BIRMINGHAM CITY UNIVERSITY

DOCTORAL THESIS

Ensembles of Pruned Deep Neural Networks for Accurate and Privacy Preservation in IoT Applications

Author:
Besher ALHALABI

Supervisor:
Prof. Mohamed Gaber
and Dr. Shadi Basura

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Artificial Intelligence and Data Analytics Group
School of Computing and Digital Technology

December 1, 2023

Declaration of Authorship

I, Besher ALHALABI, declare that this thesis titled, " Ensembles of Pruned Deep Neural Networks for Accurate and Privacy Preservation in IoT Applications " and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Besher Alhalabi

Date: 30-Nov-2023

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

BIRMINGHAM CITY UNIVERSITY

Abstract

Faculty of Computing, Engineering and Built Environment
School of Computing and Digital Technology

Doctor of Philosophy

Ensembles of Pruned Deep Neural Networks for Accurate and Privacy Preservation in IoT Applications

by Besher ALHALABI

The emergence of the AIoT (Artificial Intelligence of Things) represents the powerful convergence of Artificial Intelligence (AI) with the expansive realm of the Internet of Things (IoT). By integrating AI algorithms with the vast network of interconnected IoT devices, we open new doors for intelligent decision-making and edge data analysis, transforming various domains from healthcare and transportation to agriculture and smart cities.

However, this integration raises pivotal questions: How can we ensure deep learning models are aptly compressed and quantised to operate seamlessly on devices constrained by computational resources, without compromising accuracy? How can these models be effectively tailored to cope with the challenges of statistical heterogeneity and the uneven distribution of class labels inherent in IoT applications? Furthermore, in an age where data is a currency, how do we uphold the sanctity of privacy for the sensitive data that IoT devices incessantly generate while also ensuring the unhampered deployment of these advanced deep learning models?

Addressing these intricate challenges forms the crux of this thesis, with its contributions delineated as follows:

Ensyth: A novel approach designed to synthesise pruned ensembles of deep learning models, which not only makes optimal use of limited IoT resources but also ensures a notable boost in predictability. Experimental evidence gathered from CIFAR-10, CIFAR-5, and MNIST-FASHION datasets solidify its merit, especially given its capacity to achieve high predictability.

MicroNets: Venturing into the realms of efficiency, this is a multi-phase pruning pipeline that fuses the principles of weight pruning, channel pruning. Its objective is clear: foster efficient deep ensemble learning, specially crafted for IoT devices. Benchmark tests conducted on CIFAR-10 and CIFAR-100 datasets demonstrate its prowess, highlighting a compression ratio of nearly 92%, with these pruned ensembles surpassing the accuracy metrics set by conventional models.

FedNets: Recognising the challenges of statistical heterogeneity in federated learning and the ever-growing concerns of data privacy, this innovative federated learning framework is introduced. It facilitates edge devices in their collaborative quest to train ensembles of pruned deep neural networks. More than just training, it ensures data privacy remains uncompromised. Evaluations conducted on the Federated CIFAR-100 dataset offer a testament to its efficacy.

In this thesis, substantial contributions have been made to the AIoT application domain. Ensyth, MicroNets, and FedNets collaboratively tackle the challenges of efficiency, accuracy, statistical heterogeneity arising from distributed class labels, and privacy concerns inherent in deploying AI applications on IoT devices. The experimental results underscore the effectiveness of these approaches, paving the way for their practical implementation in real-world scenarios. By offering an integrated solution that satisfies multiple key requirements simultaneously, this research brings us closer to the realisation of effective and privacy-preserved AIoT systems.

Acknowledgements

I wish to express my deep and sincere gratitude to God for His abundant blessings and guidance throughout my PhD journey. His divine support and unwavering presence have been a constant source of strength, inspiration, and motivation, especially during the most challenging times. I am truly grateful for His grace and for granting me the opportunity to pursue my academic dreams.

Furthermore, I would like to extend my heartfelt appreciation to my supervisors (Prof.Mohamed Gaber and Dr.Shadi Basurra) for their invaluable support, guidance, and encouragement. Their expertise and mentorship have been instrumental in shaping my research work and academic career, and I am indebted to them for their generosity and patience. In particular, I would like to offer a special thanks to Professor Mohamed Gaber for his exceptional expertise, unwavering support, and insightful guidance, which have been critical to the success of my research work.

Moreover, I am immensely grateful to my family, who have been my constant source of love, understanding, and support. To my dear mother, my beloved wife Hadeel, my twin brother Ali, and my amazing daughters Jolia and Talya, I extend my deepest appreciation and admiration for their unwavering support and encouragement. Their presence in my life has been a driving force behind my achievements, and I am forever grateful for their love and care.

Lastly, I would like to honor the memory of my father, who always believed in me and encouraged me to pursue my academic goals. His unwavering love, support, and encouragement will always be cherished, and I dedicate this achievement to him.

Thank you all for being a part of this incredible journey, and for contributing to my growth and success.

Note:The research presented in this thesis was funded by Birmingham City University as part of the STEAM scholars program and supported by many individuals.

Contents

Declaration of Authorship iii

Abstract vii

Acknowledgements ix

1 Introduction 1
1.1 Preamble . . . 1
1.2 Motivation . . . 2
1.3 Problem Statement . . . 3
1.4 Aim and Objectives . . . 4
1.5 Contributions . . . 6
1.6 Publications . . . 8
1.7 Thesis Overview . . . 8

2 Artificial Intelligence of Things: A Comprehensive Review 11
2.1 Introduction . . . 11
2.2 Fundamentals of Artificial Intelligence of Things . . . 11
2.2.1 Internet of Things: Main architecture . . . 12
2.2.2 Fundamentals of Artificial Intelligence . . . 14
Supervised Learning . . . 14
Unsupervised Learning . . . 24
Ensemble Learning . . . 25
2.2.3 Graph Topology . . . 32
2.2.4 Federated learning . . . 32
The Effect of Non-independently and Identically Distributed (non-IID data) in Federated Learning 35
2.3 Bringing Machine Learning to the Edge . . . 39
2.3.1 Software Approaches . . . 39
Pruning . . . 40
Quantisation . . . 44
Fine-tuned Architecture . . . 45
Low-Rank Factorisation . . . 49
Knowledge Distillation . . . 51
Evolutionary algorithms . . . 53
2.3.2 Hardware Approaches . . . 56
2.4 Artificial Intelligence of Things Main Applications . . . 58
2.4.1 Smart Health Systems . . . 58
2.4.2 Smart Homes . . . 59
2.4.3 Smart Transportation . . . 60
2.5 Challenges of Deploying AIoT Applications . . . 61

2.6 Discussion . . . 64
2.7 Summary . . . 66

3 EnSyth: A Pruning Approach to Synthesis of Deep Learning Ensembles 67
3.1 Introduction . . . 68
3.2 Method . . . 70
3.2.1 Convolutional Neural Networks (CNNs) . . . 70
3.2.2 Feed-forward Neural Networks . . . 71
3.2.3 Net-Trim- The Pruning Method . . . 73
3.2.4 Synthesis of Deep Learning Ensembles . . . 74
3.2.5 Backward Elimination . . . 75
3.3 Experiment . . . 77
3.3.1 LeNet-5 . . . 77
3.3.2 Datasets . . . 77
CIFAR-10 . . . 77
CIFAR-5 . . . 77
MNIST-FASHION . . . 78
3.3.3 Experimental Setup . . . 78
3.3.4 Network Training and Pruning . . . 79
3.3.5 Synthesis of Compressed Deep Learning Ensembles . . . 79
3.4 Results and Discussion . . . 79
3.4.1 Results of CIFAR-10 . . . 79
3.4.2 Results of CIFAR-5 . . . 81
3.4.3 Results of MNIST-FASHION . . . 82
3.4.4 Computational Cost . . . 83
3.4.5 Backward elimination . . . 84
3.4.6 Discussion . . . 86
3.5 Summary . . . 87

4 MicroNets: A Multi-Phase Pruning Pipeline to Deep Ensemble Learning in IoT Devices 91
4.1 Introduction . . . 91
4.2 Method . . . 93
4.2.1 Pool Generation and Weight Pruning . . . 93
4.2.2 Post Training Integer Quantisation . . . 94
4.2.3 Ensemble Pruning . . . 94
4.2.4 Representative Selection Strategies . . . 97
4.3 Experimental Study . . . 99
4.3.1 Datasets and Models . . . 99
4.3.2 Implementation Details . . . 99
Model Training and Pruning . . . 99
Integer Quantisation . . . 99
Ensemble Pruning . . . 99
Deployment to a Distributed Edge Environment . . . 100
4.4 Experimental Setup . . . 100
4.5 Results and Discussion . . . 102
4.5.1 Results on CIFAR100 . . . 102
4.5.2 Results on CIFAR10 . . . 103

4.5.3 Energy monitoring . . . 105
4.5.4 Computational Complexity . . . 108
4.5.5 Discussion . . . 108
4.6 Summary . . . 111

5 FedNets: Federated Learning on Edge Devices using Ensembles of Pruned Deep Neural Networks 113
5.1 Introduction . . . 113
5.2 Method . . . 115
5.2.1 System Topology . . . 115
5.2.2 Ensemble Generation and Pruning: . . . 115
5.2.3 Model training on Local Datasets (NON-IIDs) Update Local Models Weights . . . 117
5.2.4 Graph Conversion . . . 117
5.2.5 Graph Embedding Generation . . . 117
5.2.6 Clustering of Embeddings . . . 118
Selection Criteria for Representative Models . . . 118
5.2.7 Privacy Preserving . . . 119
5.3 Experimental study . . . 120
5.3.1 Data set and Models . . . 121
Federated CIFAR100 for Simulation . . . 121
ResNetV2 . . . 121
5.3.2 Simulation Setup . . . 121
5.3.3 Results and Analysis . . . 123
Comparison With the State of the Art . . . 123
Ensembles Performance . . . 125
Preserving Privacy . . . 126
Discussion of the Results . . . 126
5.4 Summary . . . 128

6 Conclusion and Perspectives 129
6.1 Summary . . . 129
6.2 Future Direction . . . 131
6.2.1 Ensyth: Delving Deeper into Ensemble Strategies . . . 131
6.2.2 Micronets: Deep Ensemble Learning in IoT Devices . . . 131
6.2.3 FedNets: Embracing Federated Learning for Enhanced Edge Computing . . . 132

Bibliography 135

List of Figures

1.1 Onion-like Layered Model Illustrating the Three Novel Techniques Proposed in this Thesis 7

2.1 The three-layered architecture of IoT. 12
2.2 The classification Architecture. 15
2.3 The Model of an artificial neuron. 19
2.4 Architecture of a multilayer perceptron neural network. 21
2.5 Architecture of deep neural network 23
2.6 The blueprint of stacking models. 29
2.7 A typical FL-AIoT system with N participants 34
2.8 Model updates in the parameter space. Orange and green refer to the minima of global and local objectives, respectively. . . . 36
2.9 Classification of deep learning acceleration approaches. 40
2.10 Inception model with dimension reductions. 46
2.11 Net2Net vs others 52
2.12 The EANT2 algorithm with pruning during structural exploitation. 55

3.1 Visual representation of the layers in a Convolutional Neural Network (CNN). The input image is processed through convolutional layers, followed by pooling layers for spatial downsampling. The output is then fed into fully connected layers for high-level feature extraction and prediction. Each layer performs specific operations, such as convolution and pooling, which enable the network to capture spatial hierarchies and extract meaningful features from the input data.. 72
3.2 Illustration of how Net-Trim prunes unnecessary connections, making the network leaner and more efficient. 74
3.3 Synthesis of deep learning ensembles. 75
3.4 CIFAR10 82
3.5 Ensyth results using CIFAR5 83
3.6 Ensyth results using MNIST Fashion 84
3.7 Comparative Performance Metrics of LetNet5 Across CIFAR-10, CIFAR-5, and MNIST Fashion Datasets 89

4.1 MicroNets: A Multi-Phase Pruning Pipeline to Deep Ensemble Learning in IoT Devices 97
4.2 Two raspberry pi devices connected to: Google Coral Accelerator and a router 101
4.3 CIFAR100 Accuracy Comparative: “Accuracy First“ versus “Diversity First“ 102

4.4 CIFAR10 Accuracy Comparative: "Accuracy First" versus "Diversity First" 104
4.5 Energy Consumption of Micronets' Ensembles 107

5.1 *FedNets* follows a star communication topology where a server connects with all the remote clients. The server orchestrates the communications in each learning round, it starts by deploying the global ensemble to the clients. Then, each client shares a few members of their ensembles with the server. . . . 116
5.2 *FedNets* approach to convert an ANN to a graph. Each layer will be converted to a node, and the mean value of the weights/biases will be added as a feature to the node 118
5.3 *FedNets Framework involves the following steps:-Step1: server deploys deep learning ensembles to the clients and clients train them locally; step2: Each client converts the local models into graphs assuming: nodes are the layers, edges are links between layers and the attributes of the nodes are mean of weights and biases of whole layers; step3: clients generate the corresponding graph embeddings and share them with the server; step4: the server to cluster the embeddings using affinity propagation and choose a representative from each cluster, then models are deployed to clients.* 120
5.4 Accuracy comparison on two clients 123
5.5 Accuracy comparison on five clients 124
5.6 Accuracy comparison on ten clients 124

List of Tables

1.1 Comparative Analysis of the Research Challenges, Existing Approaches, and The Proposed Solutions in IoT 5

2.1 Confusion Matrix 17
2.2 Classification Matrices 18
2.3 Summary of Neuron Pruning Methods 43
2.4 MobileNets V1 vs V2 48
2.5 Summary of Finedtuned Neural Network Architectures 50
2.6 Summary of Knowledge Distillation Techniques 54
2.7 Overview of Evolutionary Algorithms in Deep Learning. . . . 56
2.8 AIoT applications in different domains 61
2.9 AIoT challenges and possible directions 64

3.1 Ensyth - hyperparameters values 80
3.2 Processing Time in μs 85

4.1 Hyperparameters, Weight pruning 100
4.2 Summary of MicroNets results on CIFAR100 103
4.3 Accuracy Comparison: "MicroNet's Clusters" versus "Random Model Selection" 103
4.4 Effect of "MicroNets' on the size of the models (in KB) 105
4.5 Inference (Milliseconds) and Temperature (Celsius) Results, Co/Reg Compilation 106
4.6 Input voltage and load current of Raspberry Pi device 106
4.7 Energy Monitoring Results for Regular Compilation Mode . . 107
4.8 Energy Monitoring Results for Co-compilation Mode 108

5.1 FedNets Hyperparameters 122
5.2 Time required by the major steps of *FedNets* in seconds. . . . 125
5.3 Changes to the number of models per client 126
5.4 Statistics related to the numbers of shared models per round . 127

List of Abbreviations

AI	**A**rtificial **I**ntelligence
ML	**M**achine **L**earning
IoT	**I**nternet **o**f **T**hings
AIoT	**A**rtificial **I**ntelligence **o**f **T**hings
DNN	**D**eep **N**eural **N**etworks
ANN	**A**rtificial **N**eural **N**etworks
CNN	**C**onvolutional **N**eural **N**etworks
DL	**D**eep **L**earning
FL	**F**ederated **L**earning
MBP	**M**agnitude **B**ased **P**runing
MI	**M**utual **I**nformation
KD	**K**nowledge **D**istillation
CPU	**C**entral **P**rocessing **U**nit
GPU	**G**raphical **P**rocessing **U**nit
TPU	**T**ensor **P**rocessing **U**nit
ACSIs	**A**pplication **S**pecific **I**ntegrated **C**ircuits
FPGA	**F**ield **P**rogrammable **G**ate **A**rrays
CIFAR	**C**anadian **I**nstitute **F**or **A**dvanced **R**esearch

Dedicated to my Family, who have been my unwavering source of support and encouragement throughout my academic journey. To my dear mother, who instilled in me a love for learning and a passion for knowledge from a young age. To my beloved wife Hadeel, whose unwavering love and constant encouragement have been a constant source of strength for me. To my twin brother Ali, whose friendship and guidance have been invaluable in helping me navigate the challenges of academia. And lastly, to my two beautiful daughters, who are a constant reminder of why I do what I do, and who motivate me to strive for excellence every day.
I also dedicate this work to the soul of my father, who instilled in me a strong work ethic, a sense of responsibility, and a belief in the power of education to transform lives. Though he is no longer with us, his memory continues to inspire and guide me.
Thank you all for your love, support, and unwavering faith in me.

Chapter 1

Introduction

1.1 Preamble

The term 'Internet of Things (IoT)' was originally coined by Kevin Ashton and other experts in 1999 [Ashton, 2009] to describe a global network of numerous and intelligent devices capable of communicating with each other through the internet. These devices are equipped with the ability to perform complex computational tasks, interact with users, exchange data with other devices, and send data to centralised data centres.

Over the years, IoT has undergone a significant evolution, driven by the advancements in the telecommunications industry such as wireless, Bluetooth, and 5G networks. As a result, a new generation of intelligent services has emerged, as highlighted by recent research [Kaur et al., 2023]. Today, IoT has transformed from old-frequency radio devices in the late 1990s to a plethora of modern, smart devices that include wearables, energy meters, smart home appliances, smart cities, healthcare systems, and much more [Kaur, Yadav, and Gill, 2023], [Bellini, Nesi, and Pantaleo, 2022].

This revolutionary development has led to a massive expansion of the IoT ecosystem, bringing about an unprecedented level of connectivity and automation. With the increasing adoption of IoT devices, the world is witnessing a new era of technological transformation, which has the potential to change the way we live and work, enabling us to achieve more efficient, sustainable, and secure societies.

With its potential to connect billions of devices and sensors, IoT has the power to impact various aspects of our daily activities and create new possibilities for improving our lives. For example, one of the most significant IoT applications is in the domain of smart agriculture. By using IoT sensors to monitor temperature, humidity, soil moisture, and other factors, farmers can obtain real-time data that enables them to optimise irrigation, fertilisation, and other factors that affect crop growth [Upadhyay, Yadav, and Gandhi, 2019]. This can lead to increased crop yields, reduced resource consumption, and improved efficiency, which are critical factors in achieving sustainable agriculture. In the manufacturing industry, IoT can be used for predictive maintenance, where sensors are used to monitor equipment performance and predict when maintenance is needed. This can help to reduce downtime, improve efficiency, and prevent equipment failures which can lead to catastrophic consequences, thus, can save significant cost, time and effort [Him, Poh, and Pheng, 2019]. Another application of IoT is in traffic management systems where sensors can be used to monitor traffic flow and optimise the

timing of traffic lights, reducing congestion, and journey times and improving safety [Elkin and Vyatkin, 2020]. With increasing number of vehicles on the road, IoT-based traffic management systems have become fundamental components of smart and sustainable cities. In healthcare, IoT devices are used to facilitate remote monitoring of patients' health, allowing doctors to track vital signs and other important data. This can help to improve patient outcomes and reduce the cost of healthcare by enabling early diagnosis and intervention [Khanna and Kaur, 2020]. Additionally, IoT has been widely used within blockchain technology to create new applications in various domains, including supply chain management, smart cities, and healthcare [Ali et al., 2019].

Another important application for blockchain IoT-based applications is Smart contracts. Smart contracts are self-executing contracts with the terms of the agreement directly written into code. They automatically enforce rules, execute actions, and facilitate transactions on a blockchain. For example, in a supply chain scenario, a smart contract could automatically trigger a payment to a supplier once certain conditions, such as delivery confirmation, are met, ensuring transparency and eliminating the need for intermediaries.[Dai, Zheng, and Zhang, 2019]. Overall, the combination of IoT and blockchain has the potential to improve the efficiency and security of various applications by enabling the automatic execution of contracts and the secure recording of data from connected devices.

1.2 Motivation

In the previous section, it was noted that the use of IoT devices is rapidly increasing in modern society, resulting in vast amounts of data being generated at the network edge. Recent forecasts indicate that the number of IoT devices in use is expected to reach 41.6 billion by 2025, generating and consuming around 79.4 ZBs of data [Badnakhe, 2021]. The conventional method for analysing the generated data involves transferring it from the devices (clients) to a central cloud server, where the analysis is conducted and the necessary insights are generated [Ghosh and Grolinger, 2020].

To analyse this data effectively, Artificial Intelligence (AI), particularly Deep Learning (DL), has become a critical aspect of IoT applications. With the ability to process large amounts of data, AI algorithms are capable of conducting complex machine learning tasks and generating valuable insights, making edge devices such as IoT devices an ideal target for these tasks. As a result, the integration of AI and IoT has given rise to a new era, the Artificial Intelligence of Things (AIoT) [Zhang and Tao, 2021], with the potential to transform various sectors, including security, transportation, healthcare, education, industry, energy, agriculture, as well as our homes and cities [Zhou et al., 2019].

However, the integration of AI and IoT also presents a range of challenges that must be addressed for the full potential of AIoT to be realised. These challenges include data privacy, statistical heterogeneity, accuracy and resource management. **Security and privacy** are critical concerns, as AIoT systems involve the transmission of large amounts of data over networks,

which makes them vulnerable to cyber-attacks and the leakage of sensitive data ["Big IoT Data Analytics" 2017]. **Accuracy**: Within the domain of the Internet of Things (IoT), clients often display a disparate distribution of class labels, referred to as statistical heterogeneity [Rashma et al., 2021]. This disparity creates hurdles in ensuring reliable and exact results. As a result, there's a pressing need for machine learning (ML) approaches, especially deep learning algorithms, that can consistently offer precise results in the presence of statistical heterogeneity. Finally, **resource management** is a critical issue, particularly when running AI applications such as deep learning on constrained devices. This is because Deep Neural Networks (DNNs) are computationally expensive, and running them on limited devices can lead to a range of issues, such as suboptimal utilisation of processing power, memory, and network bandwidth, as well as high energy consumption [Khalil et al., 2021]. To overcome these challenges, it was essential to develop innovative AI approaches in order to create efficient and privacy-enhanced AIoT networks and applications. Federated Learning (FL) is a distributed collaborative AI method that allows for the training of AI models at distributed AIoT devices without requiring data sharing [Yang et al., 2019]. Most recently, the concept of FL has been proposed as a way to build robust ML privacy-enhanced AIoT systems (could be referred as FL-IoT). For instance, multiple IoT devices can act as workers and communicate with an aggregator (e.g., a server) to conduct deep learning network training in intelligent AIoT networks [Yu et al., 2018b]. However, FL-IoT is a relatively new field and has a lot of challenges related to the distribution of the data among the AIoT network members.

1.3 Problem Statement

When designing and deploying resilient machine learning (ML) solutions for resource-constrained devices, it is imperative to consider the following key features. These features play a crucial role in ensuring the effectiveness, efficiency, and dependability of the ML system [Nguyen et al., 2021]. Therefore, to achieve optimal performance and reliability, special attention must be given to the following main features during the design and deployment phases of the ML solution.

Efficiency In order to develop machine learning solutions that are suitable for resource-constrained devices, such as Internet of Things (IoT) devices, it is crucial to design them to be resource-efficient. Deep learning algorithms, in particular, are known to be computationally intensive, making them difficult to run on devices with limited resources. Therefore, techniques such as model compression and quantization can be applied to reduce the size and complexity of the models without significantly sacrificing accuracy. Additionally, selecting the appropriate hardware and optimising the code can also help minimise resource usage [Zhang and Tao, 2021].

Accuracy In the realm of the Internet of Things (IoT), there is often an uneven distribution of class labels among clients (statistical heterogeneity)

[Rashma et al., 2021], posing challenges in delivering dependable and precise solutions. Consequently, it becomes crucial to develop machine learning (ML) solutions, particularly deep learning models, that can effectively deliver accurate outcomes. However, although providing efficient deep learning solutions for IoT devices is essential, as mentioned earlier, the accuracy of these models can be compromised due to the necessary compression techniques, leading to a potential drop in their performance.

Privacy Data privacy is a critical concern in IoT devices, as they collect and transmit a vast amount of sensitive data. The data generated by IoT devices can include personal information such as location, health, and financial data. This data can be susceptible to privacy breaches, which can result in identity theft, financial loss, and/or reputational damage. The sensitive and personal data collected by AIoT devices must be protected from unauthorised access and potential breaches. Therefore, it is crucial to design and implement deep learning models that respect the sensitive data of AIoT devices.

To craft the ideal approach for designing and deploying AIoT applications, it is crucial to ensure that it satisfies several key requirements. Firstly, it should be resource-efficient, meaning it must be designed to operate on IoT devices without significantly impacting their available resources. Secondly, it must be capable of generating accurate decisions in the presence of the statistical heterogeneity of class labels at the edge. Finally, it must respect data privacy best practices to safeguard sensitive information. However, despite the significance of these requirements, to date, there is no comprehensive approach that can effectively address all three simultaneously. Most existing works in the field tend to focus on just one aspect of these requirements, disregarding the other two [Rodríguez, Otero, and Canal, 2023], [Bi, Liu, and Kato, 2022], [Tang et al., 2017]. As mentioned above, applying deep learning to IoT environments yields several challenges, including ensuring deep learning models operate effectively on IoT devices without compromising accuracy, aptly compressing and quantising these models for resource-constrained devices, addressing the issues of statistical heterogeneity in IoT data, and upholding data privacy and security. To provide a structured overview of these challenges and the innovative solutions proposed in this research, a comprehensive table has been formulated Table 1.1. This table delineates each research challenge, the existing approaches adopted by the community to tackle them, the novel solutions proposed in this thesis, and the metrics that highlight the improvements achieved.

1.4 Aim and Objectives

The main aim of this thesis is to *develop effective and privacy-preserved computer systems for deploying AI applications on the Internet of Things (IoT) devices through the use of ensembles of pruned deep neural networks*.

To achieve this aim, we identified the following objectives:

Research Challenge	Adopted Approaches	Proposed Solution	Improvement Metrics/Measures
How deep learning models could run on IoT without compromising accuracy.	[He, Zhang, and Sun, 2017], [Romero et al., 2014], [Hu et al., 2018]	Ensyth	Higher predictability levels compared to a chosen baseline, lower number of trainable parameters, and faster CPU execution time compared to the baseline model.
Ensuring deep learning models are aptly compressed and quantised to efficiently operate on IoT.	[Sodhro et al., 2019], [Alom et al., 2018],[Han, Mao, and Dally, 2015]	MicroNets	Higher predictability levels compared to a chosen baseline, compression ratios up to 91%, and energy monitoring measures including input voltage, load current, memory footprint (size of the model), and inference time.
Tailoring models to cope with the challenges of statistical heterogeneity and the uneven distribution of class labels inherent in IoT applications.	[Konečný et al., 2017],[Tian et al., 2022], [Dao et al., n.d.,[Liang et al., 2020]	FedNets	Higher predictability levels (our approach outperforms two state-of-the-art methods).
Upholding the sanctity of privacy for the sensitive data that IoT devices incessantly generate while also ensuring the unhampered deployment of these advanced deep learning models.	[McMahan et al., 2018], [Voigt and Bussche, 2017]	FedNets	Introduces a new security mechanism to ensure privacy in federated learning.

TABLE 1.1: Comparative Analysis of the Research Challenges, Existing Approaches, and The Proposed Solutions in IoT

- Conduct a critical review of the literature on IoT, pruning techniques, deep learning ensembles, federated learning, and their applications in IoT to Identify the gaps in the existing literature.

- Design and develop a novel approach for synthesising pruned ensembles of deep learning models for better utilising of IoT resources and achieving high predictability levels.
- Design and develop a multi-phase pruning pipeline that incorporates weight pruning, channel pruning, and knowledge distillation for efficient deep ensemble learning on IoT devices.
- Design and develop a federated learning approach that enables edge devices to collaboratively train ensembles of pruned deep neural networks to tackle the statistical heterogeneity in federated learning settings while preserving data privacy.
- Evaluate the proposed approaches on benchmark datasets and compare their performance with state-of-the-art methods in terms of accuracy, efficiency, and privacy.

1.5 Contributions

The work presented in this thesis has yielded several novel contributions to the field. Specifically, we have developed novel approaches for addressing key challenges in the design and deployment of AIoT applications which directly improved the efficiency, reliability, and security of such systems.

The Synthesis of Deep Learning Ensembles To address the second objective of the thesis, our first contribution is the synthesis of an ensemble of pruned deep learning models from a baseline model, selected from a diverse space of synthesised models [Alhalabi, Gaber, and Basurra, 2019]. The ensemble's diversity leads to the outperformance of the baseline model while simultaneously eliminating the impact of compression on deep learning models by producing compressed models with better predictability measures. Moreover, the approach achieves fast inference of the pruned models through parallel processing while further boosting the accuracy through ensemble learning.

Pruning Pipelines to Deep Ensemble Learning in IoT Devices To address the third objective of the thesis, the second contribution is related to designing a novel framework that enables efficient deployment of deep ensemble learning on edge devices, maximizing the generalization of deep learning solutions at the edge end and providing robust ML solutions in complex IoT environments [Alhalabi, Gaber, and Basura, 2021]. This contribution aims to develop a multi-phase pruning pipeline that generates lightweight deep learning models capable of running on resource-limited devices without compromising their performance. The pipeline can generate individual models or ensembles and includes a new clustering-based deep ensemble pruning technique that reduces the number of models in deep learning ensembles. By applying this framework, edge devices can perform complex tasks, previously limited to high-end servers, such as real-time speech recognition, image classification, and object detection.

Federated Learning on Edge Devices using Ensembles of Pruned Deep Neural Networks To accomplish the final objective of this thesis, the third contribution aims to introduce a novel federated learning strategy for non-iid settings [Alhalabi, Basurra, and Gaber, 2023]. The proposed approach employs deep-ensemble learning to maximize the generalization of the models at the edge-end and enhance the overall performance across various distributions of the clients' local data. Furthermore, this contribution presents a unique ensemble pruning technique that minimizes the communication overheads over the network and reduces the storage footprint of ensembles. The affinity propagation clustering method is applied to the embeddings of the models, considering that a graph could represent each artificial neural network. Lastly, a novel privacy approach is described to safeguard clients' sensitive data in the applications that require running an ensemble of models.

The thesis proposes three novel techniques that can be visualised as a complete layered model similar to an onion graph model. This layered structure is used to emphasise that each component of the model is built upon the previous one. The core of the layered model is the Ensyth approach, which is surrounded by the other layers. The Micronets Approach is placed in the second layer of the onion model, reflecting that Micronets uses Ensyth in the generation of a diverse pool of models. Finally, the FetNets approach encapsulates the work that has been done in both Ensyth and Micronets and hence it is placed in the final layer of the onion model. Figure 1.1 illustrates the onion-like layered model.

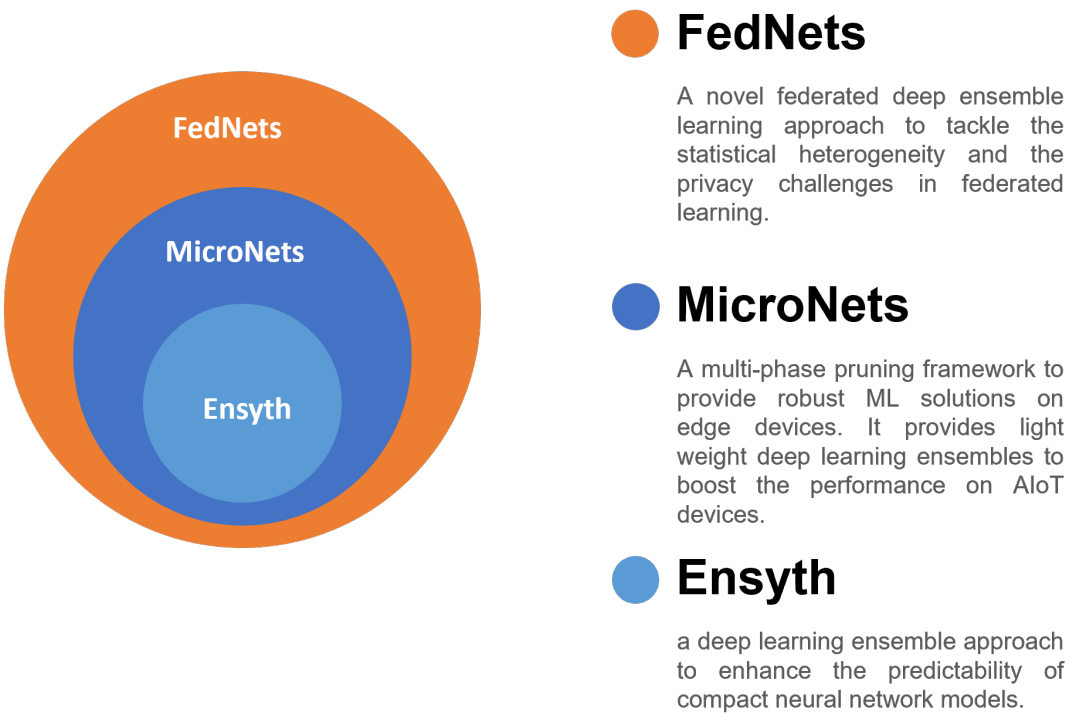

FIGURE 1.1: Onion-like Layered Model Illustrating the Three Novel Techniques Proposed in this Thesis

1.6 Publications

Our research efforts towards this thesis have resulted in the following publications:

- B. Alhalabi, M. M. Gaber and S. Basurra, "EnSyth: A Pruning Approach to Synthesis of Deep Learning Ensembles," 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), Bari, Italy, 2019, pp. 3466-3473, doi: 10.1109/SMC.2019.8913944.
- Besher Alhalabi, Mohamed Medhat Gaber, Shadi Basura, "MicroNets: A multi-phase pruning pipeline to deep ensemble learning in IoT devices", Computers and Electrical Engineering, Volume 96, Part B, 2021, 107581, ISSN 0045-7906, https://doi.org/10.1016/j.compeleceng.2021.107581.
- B. Alhalabi, S. Basurra and M. M. Gaber, "FedNets: Federated Learning on Edge Devices Using Ensembles of Pruned Deep Neural Networks," in IEEE Access, vol. 11, pp. 30726-30738, 2023, doi: 10.1109/ACCESS.2023.3261266.

1.7 Thesis Overview

The remainder of this thesis is organised as follows:

Chatper 2 [2] provides a thorough and informative overview of Artificial Intelligent Things (AIoT). The chapter presents a detailed account of the various components involved in building AIoT applications. Starting from the topology of IoT devices and their applications, the chapter concludes with an examination of the challenges associated with implementing AIoT applications in the real world.

The chapter delves into the details of the various AI algorithms that can be employed in constructing intelligent things. Specifically, the discussion covers supervised learning, which involves training a model on labelled data to predict outcomes, unsupervised learning, which entails training a model on unlabelled data to identify patterns and structures, deep learning, a subfield of machine learning that employs neural networks to learn from data, and ensemble learning, which amalgamates multiple models to improve accuracy. Furthermore, the chapter examines federated learning, a distributed learning approach that enables edge devices to collectively train a shared model without revealing raw data. The challenges and opportunities associated with each of these algorithms are discussed in detail, providing a comprehensive understanding of their capabilities and limitations. Additionally, the chapter identifies the main gaps in this field and presents the authors' contributions to addressing the identified gaps. The authors' contribution comprises innovative approaches that enhance the predictability and generalisation power of deep neural networks in compressed models.

Chatper 3 [3], we unveil our first novel approach *Ensyth*, as our first contribution towards the realisation of the second and fifth objectives that we

have set forth in this thesis. Through the utilisation of deep ensemble learning, *Ensyth* offers superior predictability levels for pruned models while also providing heightened generalisation capabilities for deep learning models operating at the edge. In addition, this chapter provides the analysis of the results obtained from subjecting our proposed approach to rigorous evaluations across various benchmarking datasets. These results provide compelling evidence of the efficacy and versatility of Ensyth.

Chatper 4 [4] introduces our second novel approach, *MicroNets*, which addresses the third and fifth objectives of this thesis. This chapter describes a multiphase pruning pipeline that enables the deployment of resource-efficient deep ensemble learning on constrained-resource devices. It also proposes a novel clustering-based pruning method for deep-learning ensembles. The chapter presents the mathematical formulation of the method and its detailed analysis on popular benchmark datasets. The experiments are performed on actual Raspberry Pi devices. The results demonstrate the effectiveness of *MicroNets* in preserving the resources of the Pi devices. The analysis covers various metrics such as accuracy, compression ratios, inference time, voltage, load current and temperature.

Chapter 5 [5] presents the culmination of our efforts towards the fourth and fifth objectives of the thesis. The novel approach, named *FedNets*, presented in this chapter is a trailblazing concept in federated learning. Unlike conventional methods that involve sharing the model weights, our approach facilitates the participating edge clients to share members of their hosted ensembles with a privacy-centric design. Notably, the proposed technique outperforms state-of-the-art federated learning approaches, including *FedAvg* and *FedYogi*. The chapter comprehensively explains the proposed algorithm, along with all experimental details related to the chosen hardware, the number of participating clients in a federated learning round, and various metrics against benchmarking federated datasets.

Chatper 6 [6], we present a succinct summary of the extensive research that has been meticulously presented in this thesis and elucidate upon the compelling conclusions that can be drawn. Furthermore, we shall contemplate and expound upon potential avenues for future research, which may serve to deepen our understanding of the subject matter at hand and contribute to the broader academic discourse.

Chapter 2

Artificial Intelligence of Things: A Comprehensive Review

2.1 Introduction

This chapter has been meticulously organised to provide readers with a comprehensive overview of the emerging field of AIoT systems. In Section 2.2, the fundamental concepts of AIoT systems are explored in detail, including the architecture of IoT systems and basic AI models, with a particular emphasis on deep learning (DL) models and federated learning (FL).

Building upon this foundation, Section 2.3 delves deeper into the advantages of combining AI and the IoT. It provides a practical example of an AIoT system that illustrates how AI and machine learning (ML) can be brought to the edge of the network to improve performance and reduce latency through a comprehensive review of acceleration techniques.

Section 2.4 shifts the focus to real-world applications of AIoT systems by providing concrete examples of how AIoT systems are being utilised in various domains, such as healthcare, agriculture, and manufacturing. These examples provide a glimpse into the potential impact of AIoT on society and the economy, underscoring the significant role this technology is set to play in shaping the future.

Finally, Section 2.5 highlights the main gaps in the AIoT field and introduces the main contribution of the thesis in addressing these gaps. Overall, this chapter provides a comprehensive and in-depth overview of the AIoT field, covering fundamental concepts, practical applications, and challenges and opportunities associated with constructing AIoT systems. As such, it is an essential resource for readers seeking to gain a thorough understanding of this rapidly evolving field.

2.2 Fundamentals of Artificial Intelligence of Things

This section offers a comprehensive overview of the fundamental concepts that underpin AIoT systems, with a specific focus on the architecture of IoT devices and the basics of Artificial Intelligence. Firstly, we provide a concise review of the general architecture of IoT devices before delving into the intricacies of AI. In doing so, we place a particular emphasis on several crucial concepts in supervised learning, including neural networks, unsupervised

learning such as clustering, and ensemble learning. Furthermore, we explore the practicalities of running deep learning models on AIoT devices.

Subsequently, we move on to expound upon federated learning, a cutting-edge approach to machine learning that is of significant relevance to AIoT systems. We discuss the most popular and advanced state-of-the-art federated learning strategies, analysing specific challenges posed by AIoT devices.

2.2.1 Internet of Things: Main architecture

The Internet of Things (IoT) is a technology that facilitates comprehensive perception and universal connectivity by leveraging sensors, wired and wireless networks, and cloud computing. Numerous published studies describe various IoT architectures [Jabraeil Jamali et al., 2020],[Mocnej et al., 2018],[Ray, 2018],[Al-Fuqaha et al., 2015]. However, to enable the connection of billions or trillions of diverse objects via the internet, flexible and adaptable layered architecture is required. The three-layer and five-layer architectures are the most commonly used in business, industrial research, and applications [Al-Qaseemi et al., 2016]. In the upcoming paragraph, we will briefly explain the IoT three-layers architecture.

As the name suggests, the three-layer architecture consists of three layers: the perception layer, the network layer, and the application layer. Figure 2.1 depicts a typical IoT architecture with three layers.

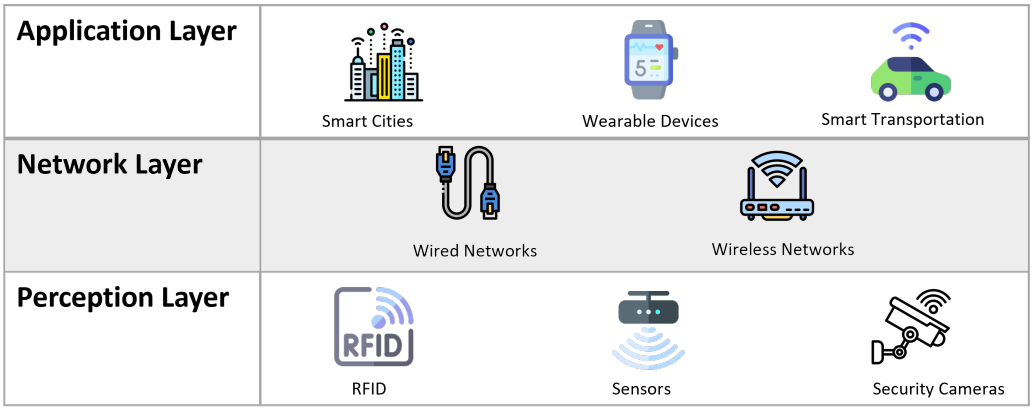

FIGURE 2.1: The three-layered architecture of IoT.

- Perception Layer: The perception layer is the bottommost layer in the IoT architecture, also known as the objects/devices layer. This layer comprises the physical devices that collect data from the environment using various perception tools, including sensors, motors and actuators and transmitters (such as temperature and humidity sensors, voltage transducers, etc.), RFID tag/EPC code scanners, webcams/microphones, and human body infrared sensors [Ray, 2018].

  After collecting information, the perception layer performs the initial processing and packaging of the data and then sends it to the application layer. Additionally, it receives control information from the network layer, which it uses to carry out the necessary control operations

[Zhong, Zhu, and Huang, 2017]. However, the perception layer is a prime target for attackers looking to exploit it, often by attacking sensors and other perception tools. Several common security threats related to the perception layer include:

1. *Eavesdropping:* This attack involves intercepting real-time transmissions, such as phone calls, text messages, and video streaming, to gain unauthorized access to private communications [Burhan et al., 2018].
2. *Timing Attack:* Attackers use this method in systems with limited computational power to identify weaknesses and uncover secrets by measuring the system's response time to various queries [Brumley and Boneh, 2005].
3. *Replay Attack:* This attack occurs when an intruder intercepts a conversation between a sender and receiver and uses the authentic information obtained to impersonate the sender. The intruder sends the same authenticated information to the victim, presenting proof of their identity, leading the victim to believe it's a valid request. As the message is in encrypted form, the receiver may unknowingly take the desired action of the intruder [Puthal et al., 2016].

- The Network Layer: Also known as the Transmission Layer, serves as a vital link between the Data Link Layer and the Application Layer, responsible for routing and forwarding data between them [Mrabet et al., 2020]. It also facilitates the connection of smart devices, networking equipment, and networks, making it an attractive target for attackers. Several security threats associated with the network layer are worth noting:

  1. *Denial of Service (DoS) Attack:* This is an attempt to prevent legitimate users from accessing devices or network resources by flooding them with excessive traffic. The attacker floods the targeted devices or resources with superfluous requests, making it challenging or impossible for genuine users to utilize them [Prabhakar, 2017].
  2. *Man-in-The-Middle (MiTM) Attack:* A type of cyberattack where an attacker secretly intercepts and modifies communication between a sender and receiver who believe they are communicating directly with each other. The attacker gains control over the communication, allowing them to alter messages as they wish. This poses a significant threat to online security, as it gives the attacker the ability to capture and manipulate information in real time [Conti, Dragoni, and Lesyk, 2016].

- Application Layer: The Application Layer is a crucial component of the IoT architecture that defines all the applications that employ the IoT technology or in which IoT has been deployed. These applications can

range from smart homes and smart cities to smart health, among others. It also serves as the control and decision-making layer, and it handles a vast amount of data and leverages it to perform a diverse range of tasks, including industrial control and health monitoring [Navani, Jain, and Nehra, 2017]. However, the Application Layer is vulnerable to several security threats and issues that can compromise the integrity, confidentiality, and availability of data. Notably, malicious code attacks pose a significant threat to the Application Layer. This type of cyber attack uses code or software to cause harm or unauthorised access to a computer system or network. Malware, such as viruses, worms, Trojans, and ransomware, are common forms of malicious code or software that can be used in such attacks [Deogirikar and Vidhate, 2017]. These attacks are difficult to detect and remove, and standard anti-virus tools may not be adequate to prevent or control them. Additionally, cross-site scripting is another type of attack that can impact the Application Layer. In this attack, the attacker injects client-side scripts, such as JavaScript, into a popular website that others view. This enables the attacker to modify the website's content for their purposes and use legitimate information for illicit activities [Gupta and Gupta, 2017].

2.2.2 Fundamentals of Artificial Intelligence

Artificial Intelligence (AI) refers to the development of algorithms and computer programs that mimic human intelligence, including the ability to think and learn. The technology has witnessed remarkable growth in recent years and has been implemented across various applications, ranging from self-driving cars to virtual assistants like Siri and Alexa and ending with IoT devices.

Machine Learning (ML), a subset of AI, is the most widely used method for systems to learn automatically from data, identify patterns, and make decisions without human intervention. ML algorithms can be classified into three main categories: supervised learning algorithms, unsupervised learning algorithms, and reinforcement learning. Supervised learning involves the use of labeled data to train models, while unsupervised learning involves training models using unlabeled data. Reinforcement learning involves training agents to interact with their environment to maximize rewards.

In this section, we focus on the most popular algorithms in supervised and unsupervised learning, including linear regression, logistic regression, decision trees, random forests, k-means clustering, and principal component analysis. We will also discuss various methods of ensemble learning, which aim to combine the predictions of multiple models to improve accuracy and robustness.

Supervised Learning

Supervised learning is a fundamental category of machine learning that is widely used in various applications, including image classification, speech recognition, natural language processing, and many more [Jiang, Gradus, and Rosellini, 2020]. It involves training a model to predict an output based

on a given input using a labelled dataset, where the output is the correct label for the input. The learning algorithm generates a function that can predict the output based on the input data [Singh, Thakur, and Sharma, 2016].

The process of building a supervised learning model consists of two key steps: training and testing. In the training phase, the model is fed with labelled training data, which comprises input samples and their corresponding labels. The learning algorithm then uses these examples to learn the patterns in the data and build a predictive model. In the testing phase, the model is applied to new and unseen data, and the algorithm leverages the learned patterns to make predictions about the output of the input data [Southwest Jiaotong University, China et al., 2015]. The learning process is explained in Figure 2.2:

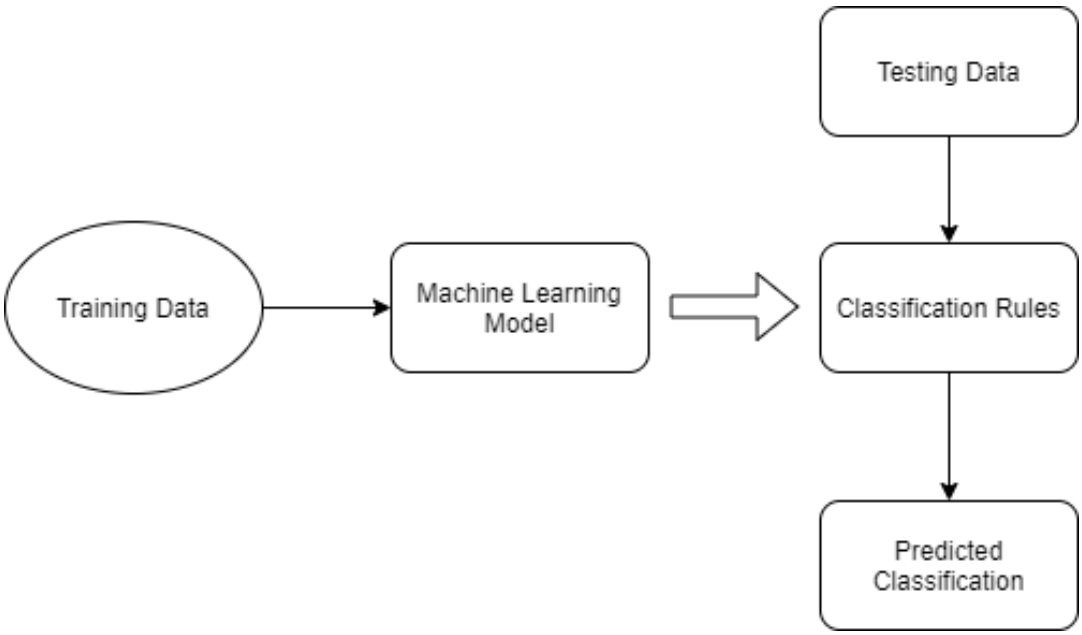

FIGURE 2.2: The classification Architecture.

In supervised learning, the classification model takes input data, represented as X, and attempts to predict the output Y using the learned parameters denoted as ζ. The process begins with initialising the parameters, ζ, and then repeatedly updates the parameters through computing the gradient of the loss function with respect to the parameters and moving in the opposite direction of the gradient. This iterative process continues until either convergence is reached or a stopping criterion is met [Goodfellow, Bengio, and Courville, 2016].

Algorithm 1 presents the pseudo-code of the supervised learning algorithm. Specifically, the algorithm defines the steps required to train a supervised learning model, starting from initialising the parameters, computing the loss function, and then updating the parameters using backpropagation. At each iteration, the algorithm adjusts the parameters based on the gradient of the loss function, moving them in the opposite direction until convergence is reached.

Algorithm 1 Supervised Learning Algorithm

Given: Training set $(X, Y) = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$
Initialise the model parameters, θ
Repeat until convergence:
for i = 1 to n **do**
  Compute the predicted output, $\hat{y} = f(x_i, \theta)$.
  Compute the loss function, $J(\theta) = L(y_i, \hat{y})$
  Compute the coefficient $\alpha_t = \frac{1}{2} ln(\frac{1-e_t}{e_t})$
  Compute the gradients, $\nabla_\theta J(\theta)$
  Update the parameters, $\theta = \theta - \alpha \nabla_\theta J(\theta)$
end for
Return the learned parameters, θ

where :

- X is the input data.
- x_i is the ith input sample.
- Y is the target output.
- y_i is the ith output sample.
- θ is the model parameter.
- $f(x, \theta)$ is the model function.
- $L(y_i, \hat{y})$ is the loss function.
- $\nabla_\theta J(\theta)$ is the gradient of the loss function with respect to the parameters.
- α is the learning rate.
- n is the number of training samples.

Supervised learning can be categorised into two types based on the type of output Y: classification and regression. In classification, the target output consists of a fixed range of categorical values. On the other hand, regression deals with continuous output values. The choice of supervised learning type depends on the nature of the problem being addressed and the desired outcome. Once the model has been trained, its accuracy needs to be evaluated.

The outcomes of Y are called predictions, and these predictions are compared to the true labels of the test data to assess the accuracy of the model. Evaluating supervised machine learning models is a crucial step in the development process, as it allows for the determination of the model's ability to make accurate predictions on new and unseen data. Therefore, selecting an appropriate evaluation technique is important for ensuring the accuracy of the model's predictions. There are several common techniques for evaluating supervised machine learning models, including:

1. Confusion matrix: A confusion matrix is a valuable tool in evaluating the performance of a classification algorithm. It provides a concise summary of the number of correct and incorrect predictions made by the classifier. It presents the predicted values of the model against the true values, providing insight into how the model is performing. The detailed definition of the confusion matrix, including its components such as true positive, false positive, true negative, and false negative, can be found in Table 2.1:

	Actual 1	Actual 0
Predicted 1	TP	FP
Predicted 0	FN	TN

TABLE 2.1: Confusion Matrix

where:

- TP (True Positive) is the number of instances that were correctly classified as positive.
- TN (True Negative) is the number of instances that were correctly classified as negative.
- FP (False Positive) is the number of instances that were incorrectly classified as positive.
- FN (False Negative) is the number of instances that were incorrectly classified as negative.

2. Metrics: Evaluating the performance of a supervised machine learning model requires the use of various metrics, such as accuracy, precision, recall, F1-score, ROC-AUC, log loss, and others [Powers, 2020], [Fawcett, 2006]. The selection of the right metric for a given problem depends on the nature of the data and the desired outcome. For instance, accuracy can be an effective measure for evaluating balanced datasets, but it may not be suitable for imbalanced ones. In Table 2.2, we provide a list of the most widely used evaluation metrics to help researchers choose the best metrics for their particular applications.

3. Train-test split: One of the simplest ways to evaluate a supervised machine learning model is to split the available data into two disjoint subsets: a training set and a test set. The model is then trained on the training set and its performance is evaluated on the test set. This approach provides an estimate of how well the model is able to generalize to unseen data. It is important to ensure that the distribution of the data is similar between the two subsets. Otherwise, the model may overfit to the training set and perform poorly on the test set. The train-test split is a widely used technique for evaluating supervised machine learning models [Sammut and Webb, 2010].

Metrics name	Definition	Formula
Accuracy	Accuracy is a common metrics used in machine learning to evaluate the performance of a classifier. It measures the fraction of instances that were correctly classified by the classifier.	$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
Precision	It measures the fraction of positive instances that were correctly classified by the classifier.	$Precision = \frac{TP}{TP+FP}$
Recall	measures the fraction of positive instances that were correctly classified by the classifier out of all positive instances.	$Recall = \frac{TP}{TP+FN}$

TABLE 2.2: Classification Matrices

4. Cross-validation: Another method of evaluating a model's performance is cross-validation. Cross-validation techniques, such as k-fold cross-validation [Hastie, Friedman, and Tibshirani, 2001], are widely used for this purpose. This technique involves dividing the data into k subsets and generating k pairs of (train, test) datasets, denoted as $D_{(\text{train,j})}$ and $D_{(\text{test,j})}$ where $j = 1$ to k. [Berrar, 2018]. The model is trained on each of the $k-1$ training sets and evaluated on the corresponding test set. This results in a more comprehensive evaluation of the model's abilities since it is tested on different subsets of the data. Cross-validation also helps in reducing overfitting and can be used to tune hyperparameters.

Neural Networks Artificial Neural Networks (ANNs) are an essential machine learning model that is inspired by the complexity and adaptability of the human brain. The architecture of ANNs consists of interconnected layers of "neurons," which are capable of processing and transmitting complex information. Each neuron within the network receives input signals, performs a series of computations on that input, and produces an output signal that is transmitted to other neurons within the network. These computations are made possible through the use of various activation functions, which can introduce non-linearities into the network's computation and enable it to model complex phenomena [Goodfellow, Bengio, and Courville, 2016].

The use of ANNs is growing at an unprecedented rate in numerous fields to address various human needs. ANNs have been shown to be highly effective in a range of applications, including image recognition, natural language processing, speech recognition, and many others. These models can also be deployed for a variety of tasks, including prediction, classification, regression, and more. [Krizhevsky, Sutskever, and Hinton, 2012], [Collobert et al., 2011], [Fridman et al., 2017]. In recent years, there has been a significant interest in the use of ANNs for operation research in the economic sector. Businesses are investing in ANNs to enhance their operational efficiency, reduce costs, and increase profitability. For instance, ANNs have been employed to forecast financial time series, including stock prices and

currency exchange rates, with impressive accuracy [Boyacioglu, Kara, and Baykan, 2009]. Moreover, ANNs have been used in credit risk analysis, fraud detection, and customer segmentation, among other applications [Dharwadkar and Patil, 2018], [Berrada, Barramou, and Alami, 2022]. These advances demonstrate the potential of ANNs to revolutionize the way businesses operate, enabling them to gain a competitive edge in a highly dynamic and complex environment.

A neural network can be represented as a directed acyclic graph, where the neurons are represented by nodes and the edges represent the flow of information. The edges have weights associated with them, which represent the strength of the connection between neurons [Li et al., 2021a]. The fundamental building block of a neural network is the artificial neuron or perceptron [Block, 1962]. A perceptron takes multiple inputs, performs a weighted sum of the inputs, and then applies an activation function to the sum to produce the output. The inputs to a perceptron are usually denoted by $x_1, x_2, ..., x_n$, the weights by $w_1, w_2, ..., w_n$, and the bias by b. The weighted sum of the inputs plus the bias is denoted by $z = w_1x_1 + w_2x_2 + ... + w_nx_n + b$. The activation function f is applied element-wise to the weighted sum z to produce the output y of the perceptron. Mathematically, a perceptron can be represented as:

$$y = f(w_1x_1 + w_2x_2 + \cdots + w_nx_n + b) \tag{2.1}$$

Also Figure 2.3 shows the model of an artificial neuron.

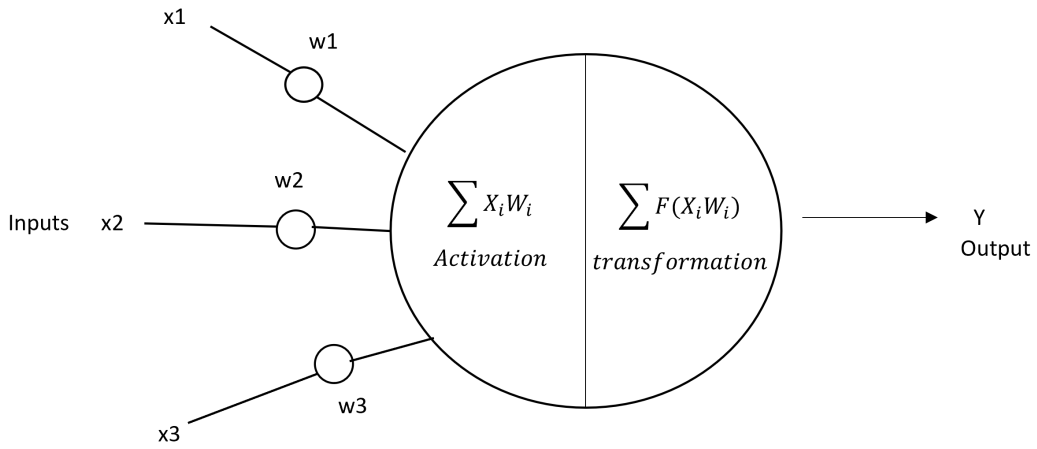

FIGURE 2.3: The Model of an artificial neuron.

The most common activation functions used in ANNs are Sigmoid, ReLU, Tanh, and softmax [Sharma, Sharma, and Athaiya, 2020]. In the next section, I will briefly explain each function and its use cases.

a) Sigmoid Activation Function: The sigmoid function is the most commonly used activation function because it is a non-linear function. It is often used in a binary classification problem, where the output should be between zero and one, representing the probability of the input belonging to a certain class [Costarelli and Spigler, 2013]. Mathematically, the Sigmoid function is

represented as:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

However, it's worth noting that the use of the Sigmoid function has been replaced by other functions like ReLU, Tanh, and others in many deep learning architectures, due to the vanishing gradient problem it can cause for deep networks [Roodschild, Gotay Sardiñas, and Will, 2020]. **B) ReLU Activation Function**: ReLu stands for Rectified Linear unit and is a non-linear activation function that is commonly used in the neural network. ReLU activation function is defined as [Costarelli and Spigler, 2013]:

$$f(x) = max(0, x) \tag{2.3}$$

The function returns zero for any input x less than 0 and returns x for any input x greater than or equal to 0. This means that for any negative input, the output is always 0, and for any positive input, the output is equal to the input. One of the main benefits of the ReLU function is that it helps to alleviate the vanishing gradient problem that can occur with other activation functions like the Sigmoid function, as mentioned earlier. This is because the derivative of the ReLU function is either zero or one, making it computationally efficient and easy to train. However, the standard ReLU activation function presents the potential drawbacks of causing a neural network to explode [Si, Harris, and Yfantis, 2018]. When a neural network experiences exploding gradients, it becomes challenging to optimize the model effectively. The weights may be updated with extremely large values, causing the network to become unstable and produce unreliable predictions [Pascanu, Mikolov, and Bengio, 2013]. **c) Tanh Activation Function**: The hyperbolic tangent (Tanh) activation function is another commonly used activation function in artificial neural networks [Sharma, Sharma, and Athaiya, 2020]. The formula is defined as:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.4}$$

This function maps the input values to a range between -1 and 1. It has a similar range as the Sigmoid but its output is symmetric around the origin. This makes the optimisation of the model easier as the gradient is more consistent [Evans and Raslan, 2005]. The Tanh activation function is often used in the hidden layers of a neural network and it's a good choice when the input is zero-centred. However, it can also suffer from the vanishing gradients problem when the inputs are too high or too low, similar to the sigmoid. **d) Softmax Activation Function**: The function is often used in the output layer of a neural network that is performing multi-class classification [Sharma, Sharma, and Athaiya, 2020]. The mathematical representation of the function is defined as:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{2.5}$$

The softmax function maps the input values to a probability distribution over the possible classes. It is a generalisation of the sigmoid function, as it's used

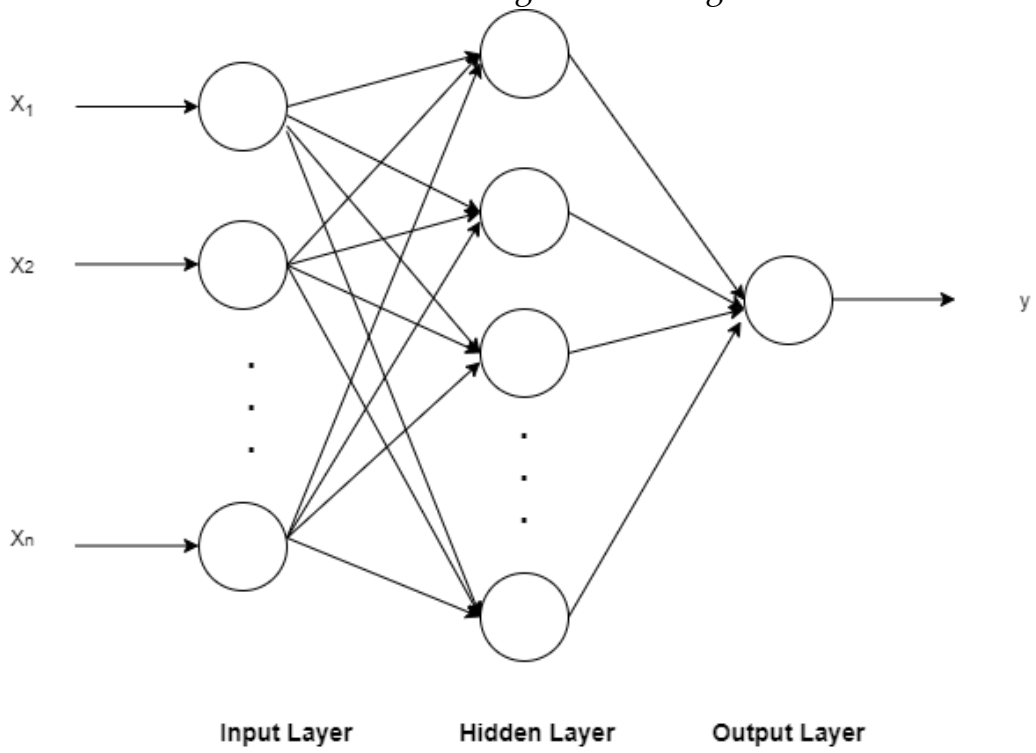

FIGURE 2.4: Architecture of a multilayer perceptron neural network.

for multi-class classification problems. The output of the softmax function for all the classes will be between zero and one, and the sum of the outputs will be equal to one. This makes it easy to interpret the output as the probability of the input belonging to each class. However, in [Yang et al., 2018c], the authors discovered that the softmax function could limit the representational capacity of language modelling.

While a single neuron can perform certain fundamental information processing operations, the true strength of neural computation lies in the interconnectedness of neurons in a network. A neural network is a collection of perceptrons organised into layers. The simplest neural network is a single-layer perceptron, which is only capable of solving linearly separable problems. To solve problems that are not linearly separable, we need to use a multi-layer perceptron (MLP) [Taud and Mas, 2018].

An MLP is a feedforward neural network that consists of an input layer, one or more hidden layers, and an output layer. The input layer receives the input data and the output layer produces the predictions. The hidden layers are used to extract features from the input data and to increase the capacity of the network to represent complex functions [Taud and Mas, 2018]; the architecture of a multilayer perceptron neural network is shown in Figure 2.3. The process of training a feed-forward neural network or MLP involves two main steps: 1)- *Forward Propagation*: where input data is passed through the layers of a network, and the output of each layer is used as the input for the next layer, until the final output is produced [Maldonado and Manry, 2002].

This can be represented mathematically as:

$$a_i^j = f(\sum_{k=1}^{n} w_{i,k}^j a_k^{j-1} + b_i^j) \tag{2.6}$$

where:

- a_i^j is the activation (output) of the i^{th} neuron in the j^{th} layer;
- $w_{i,k}^j$ is the weight of the connection between the k^{th} neuron in the $(j-1)^{\text{th}}$ layer and the i^{th} neuron in the j^{th} layer;
- b_i^j is the bias of the i-th neuron in the j-th layer and
- f is the activation function applied element-wise to the weighted sum.

2- Backpropagation) This step involves adjusting the weights and biases to minimise the error between the network's predictions and the true values [Johansson, Dowla, and Goodman, 1991]. This is typically done using an optimisation algorithm such as stochastic gradient descent (SGD) [Ketkar, 2017]. The backpropagation algorithm starts with a random initialisation of the weights and biases, and then iteratively updates the weights and biases in the direction of the negative gradient of the error function with respect to the weights and biases [Amari, 1993]. The error function is usually chosen to be a measure of the difference between the network's predictions and the true values, such as mean squared error (MSE) or cross-entropy loss. The weight update rule can be represented as:

$$w_{i,j}^t = w_{i,j}^{t-1} - \alpha \frac{\partial E}{\partial w_{i,j}} \tag{2.7}$$

where:

- $w_{i,j}^t$ is the weight of the connection between the i^{th} neuron in the input layer and the j^{th} neuron in the output layer at time step t;
- $w_{i,j}^{t-1}$ is the weight of the connection between the i^{th} neuron in the input layer and the j^{th} neuron in the output layer at time step $t-1$;
- α is the learning rate
- E is the error function

Deep Learning Deep neural networks (DNNs) are a type of artificial neural network that is composed of multiple layers. Instead of having a hidden layer, a typical deep neural network could have multiple hidden layers and an output layer. The network's architecture is shown in Figure 2.5. The layers of a DNN can consist of various types of artificial neurons, such as fully connected neurons, convolutional neurons, or recurrent neurons.

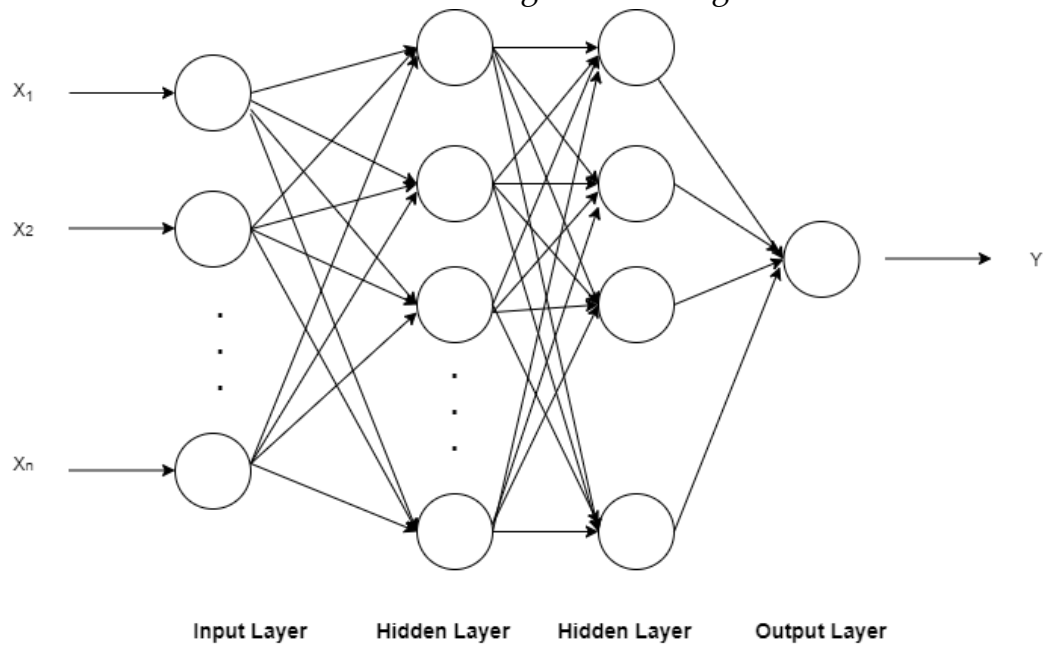

FIGURE 2.5: Architecture of deep neural network

Convolutional Neural Networks Convolutional Neural Networks (CNNs) have emerged as powerful deep learning models for various computer vision tasks. In this section, we provide an introduction to CNNs, discuss their layers and mathematical formulas, explore their applications in the Internet of Things (IoT), and analyse the advantages and disadvantages of IoT.

Definitions of CNNs Convolutional Neural Networks are a type of neural network specifically designed to process grid-like data, such as images. They are composed of multiple layers, each performing specific operations on the input data. CNNs have been widely successful in image classification, object detection, and other computer vision tasks due to their ability to automatically learn hierarchical representations from raw pixel values [Lecun et al., 1998], [Krizhevsky, Sutskever, and Hinton, 2012]

Layers of CNNs CNNs typically consist of several layers, including convolutional layers, pooling layers, and fully connected layers. Each layer serves a specific purpose in feature extraction and classification.

- Convolutional Layers: Convolutional layers apply a set of learnable filters (kernels) to the input image, convolving them over the image spatially. The filters capture local spatial patterns and generate feature maps, preserving the spatial relationships. The convolution operation can be defined as:

$$\text{Output feature map}[i,j] = \sum_{m=1}^{M}\sum_{n=1}^{N} \text{Input feature map}[i+m, j+n] \times \text{Kernel}[m,n]$$

where M and N are the dimensions of the kernel, and i and j represent the spatial position in the output feature map [Krizhevsky, Sutskever, and Hinton, 2012].

- Pooling Layers: Pooling layers downsample the feature maps generated by the convolutional layers, reducing their spatial dimensions. Common pooling operations include max pooling and average pooling, which extract the most important features while reducing computational complexity.

- Fully Connected Layers: Fully connected layers take the flattened feature maps from the previous layers and connect them to the output layer for classification or regression. These layers learn high-level representations and capture global context

The mathematical formulas involved in CNNs include convolution operations, activation functions (e.g., ReLU), pooling operations, and matrix multiplications. These formulas define the forward pass and backward pass computations that enable the network to learn and update its parameters during training [Goodfellow, Bengio, and Courville, 2016].

Applications of CNNs in IoT CNNs have found numerous applications in the Internet of Things (IoT), leveraging their ability to process visual data and extract meaningful information. Some comprehensive applications include: Smart Surveillance Systems: CNNs can analyse video feeds from surveillance cameras in real-time, detecting and classifying objects, tracking movements, and identifying anomalies [Cheng et al., 2017].

Environmental Monitoring: CNNs can analyze images from IoT devices such as drones or sensors to detect pollution, monitor wildlife, or identify environmental changes [Leong et al., 2020].

Healthcare and Telemedicine: CNNs can aid in medical diagnosis by analysing medical images such as X-rays, CT scans, or histopathology slides, assisting in disease detection and treatment planning [Litjens et al., 2017].

Smart Home Devices: CNNs can enable smart home devices to understand and respond to visual cues, such as facial recognition for personalised settings or object recognition for security purposes [Yu, Antonio, and Villalba-Mora, 2022].

Unsupervised Learning

Unsupervised learning investigates how systems can learn to categorise input patterns based on the statistical relationships among the patterns in the entire dataset [Hastie, Friedman, and Tibshirani, 2001]. In comparison to supervised and reinforcement learning, unsupervised learning lacks specific target outputs or evaluations for each input. Instead, the unsupervised learning process utilises preconceived biases to decide what aspects of the input's structure should be represented in the output [Ghahramani, 2004]. There are different unsupervised machine learning algorithms such as Association rule learning algorithms [Kumbhare and Chobe, 2014], Anomaly detection algorithms [Chandola, Banerjee, and Kumar, 2009] [Agrawal and Agrawal,

2015], Dimensionality reduction algorithms [Huang, Wu, and Ye, 2019] [Sarveniazi, 2014], and Clustering algorithms [Xu and Tian, 2015] [Rokach, 2010] [Ezugwu et al., 2022]. In the next section, I will explain *K-means* as an example of unsupervised learning algorithms as it's highly related to my contributions.

K-means: K-means is a type of unsupervised learning algorithm that is used for cluster analysis in data mining [Ahmed, Seraj, and Islam, 2020]. The goal of K-means is to partition a set of data points into k clusters, where each data point belongs to the cluster with the nearest mean. The number of clusters, k, is defined by the user [Ahmed, Seraj, and Islam, 2020]. The algorithm works by first randomly initialising k cluster centres, and then it proceeds iteratively. Algorithm .2 illustrates the main steps of the K-means algorithm.

Algorithm 2 K-Means Algorithm

1: Requires a dataset $\mathbf{X}$ with n samples and m features, number of clusters k, maximum number of iterations max_iter.
2: Initialise the centroids $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k$ randomly from the dataset $\mathbf{X}$.
3: **for** $i = 1$ to max_iter **do**
4: Assign each sample in $\mathbf{X}$ to the closest centroid.
5: Recalculate the centroids as the mean of the samples assigned to each cluster.
6: **if** The centroids do not change **then**
7: **break**
8: **end if**
9: **end for**
10: update the final clusters and centroids.

The K-means algorithm aims to minimise an objective function known as the squared error function, also known as the within-cluster sum of squares. The objective function is given by the following equation [Mahdavinejad et al., 2018]:

$$J(\pi, \mu) = \sum_{n=1}^{N} \sum_{k=1}^{K} \pi_{nk} * ||x_n - \mu_k||^2 \tag{2.8}$$

Where: π_{nk} is the binary indicator variable as explained previously; mu_k is the kth cluster centroid; x_n is the nth data point. The objective function $J(\pi, \mu)$ measures the sum of the squared distances between each data point x_n and its closest cluster centroid mu_k. The algorithm iteratively updates the cluster centroids and the assignment of data points to clusters in order to minimize the objective function $J(\pi, \mu)$. In short, the k-means algorithm aims to minimise the within-cluster sum of squares by iteratively updating the cluster centroids and the assignment of data points to clusters.

Ensemble Learning

Ensemble learning is a method that combines multiple base learners, also known as inducers, to make a decision. These techniques are widely used

in supervised machine learning, where the objective is to develop a model that can predict the output based on a set of labelled examples [Zhou, 2021]. The key concept of ensemble learning is that multiple models are combined together to form a more robust and accurate overall prediction. By leveraging the strengths of multiple models, the errors of one model can be offset by the others, resulting in improved performance compared to using a single model alone [Sagi and Rokach, 2018]. This technique could be used in the Artificial Intelligence of Things (AIoT) to improve the performance of machine learning models [Alhalabi, Gaber, and Basura, 2021]. For example, an ensemble of models could be used to classify sensor data from IoT devices in order to detect anomalies or make predictions about future events. This can help to improve the reliability and accuracy of AIoT systems [Tsogbaatar et al., 2021]. There are several ensemble learning methods will be explained in the following.

- **Bagging (Bootstrap Aggregating)**: Bagging is an ensemble learning method that aims to improve the accuracy and robustness of a single model by training multiple models on different subsets of the data and then averaging their predictions. This technique samples the data randomly (with replacement), and then trains a separate model on each sample. By training multiple models on different subsets of the data, the final predictions will be more robust and less sensitive to the specific training data [Kadiyala and Kumar, 2018]. An important aspect of bagging is that each model is trained independently of the others, which means that the models can be trained in parallel to improve computational efficiency. The bagging could mathematically be represented as [Sagi and Rokach, 2018]:

  Let $f1(x), f2(x), \cdots, fn(x)$ be the base learners or inducers trained on different subsets of the training data, and let x be the input feature vector for which we want to make a prediction. The final prediction made by the ensemble is given by:

  $$y = \frac{1}{n} \sum_{i=1}^{n} f_i(x) \tag{2.9}$$

  where y is the final prediction, n is the number of base-learners, $f_i(x)$ is the prediction of the ith base-learner, and x is the input feature vector.

  It is worth mentioning that the formula in Formula 2.9 works on regression only. For the classification task, the final prediction is usually made by voting, where each base learner votes for a class:

  $$y = argmax_c(\sum_{i=1}^{n} [f_i(x) = c]) \tag{2.10}$$

  where y is the final prediction, c is the class, n is the number of base-learners. Bagging can be applied to many different types of models, such as decision trees, neural networks, and support vector machines. An example of an ensemble learning algorithm that uses bagging is Random Forest. Random Forest is an extension of decision trees that

creates multiple decision trees by bagging and then averaging their predictions [Kadiyala and Kumar, 2018]. However, like any other method, Bagging also has its own set of disadvantages [Bühlmann, 2012], which includes: 1-) it can increase the variance of the final model;2-) Bagging can be computationally expensive, especially when the training dataset is large or when the base-learner is computationally intensive.

- **Boosting** Boosting is an ensemble learning method that aims to improve the accuracy of a single model by training multiple models in sequence, each model attempting to correct the mistakes of the previous model. The basic idea behind boosting is to iteratively train weak models and combine them to create a strong model. Each weak model is trained on the same dataset, but with different weights assigned to the data points. Data points that are misclassified by the previous model are given higher weights so that the next model will focus more on these points [Sagi and Rokach, 2018].

  The final prediction is made by combining the predictions of all the weak models through a weighted majority vote. The weights assigned to each model are based on their accuracy, with more accurate models given higher weights [Ferreira and Figueiredo, 2012]. Boosting can be applied to many different types of models, such as decision trees, neural networks, and support vector machines. Some popular examples of ensemble learning algorithms that use boosting are AdaBoost and Gradient Boosting; AdaBoost is a boosting algorithm for binary classification problems, while Gradient Boosting is a more general boosting algorithm that can be used for both regression and classification problems.

  - Adaboost: This algorithm works by repeatedly training the weak models on subsets of the data, where the subsets are chosen in such a way that the misclassifications of the previous weak models are given more weight. This results in the final ensemble model having higher accuracy than any of the individual weak models [Freund and Schapire, 1997]. AdaBoost is often used in the field of computer vision and image processing. The pseudo-code of Adaboost algorithm is shown in Algorithm 3.

Algorithm 3 Adaboost Algorithm

Initialise:
$w_i = \frac{1}{N}$ for all i, where N is the number of samples.
for t = 1 to T **do**
Train a weak classifier $h_t(x)$ using the current sample weights.
Compute the weighted error rate $e_t = \sum_{i=1}^{N} w_i I(y_i \neq h_t(x_i))$
Compute the coefficient $\alpha_t = \frac{1}{2} ln(\frac{1-e_t}{e_t})$
Update the sample weights: $w_i = w_i * e^{-\alpha_t * y_i * h_t(x_i)}$
Normalize the sample weights: $w_i = \frac{w_i}{\sum_{i=1}^{N} w_i}$
end for
The final classifier is:
$f(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$

Although Adaboost could provide a remarkable boost in accuracy [Cao et al., 2013], it has several limitations, such as: 1)- AdaBoost is computationally expensive as it requires training multiple weak models. This can be time-consuming, especially for large datasets;2)- Adaboost is unsuitable for real-time prediction because of it's sequential nature;3)- Adaboost is limited to classification problems and can't be used in the regression as the goal here would be to predict a continuous value rather than a class label. AdaBoost's approach of adjusting the weights of the training examples based on the misclassifications of the previous weak models is not applicable in this context. Additionally, the weak models used in AdaBoost are typically decision trees, which are not well-suited for regression problems [Cao et al., 2013].

- **b) Gradient Boosting:** this approach follows the same boosting concept where each model tries to correct the mistakes of the previous model. However, it uses a different mathematical formula to compute the final prediction. This includes tuning a shrinkage parameter to control the learning rate and prevent overfitting [Bentéjac, Csörgő, and Martínez-Muñoz, 2021]. The mathematical formula to compute the gradient boosting algorithm is given by:

$$F_m(x) = F_{m-1}(x) + \text{argmin}c \sum i = 1^n L(y_i, F_{m-1}(x_i) + c) \tag{2.11}$$

where $F_m(x)$ is the final prediction of the mth iteration, $F_{m-1}(x)$ is the prediction of the previous iteration, $L(y_i, F_{m-1}(x_i) + c)$ is the loss function, c is the parameters of the new tree and x, y are the input and output variables respectively. XGBoost (eXtreme Gradient Boosting) is an optimised version of Gradient Boosting. It is specifically designed for decision tree base learners and uses a more regularized model formalisation to control over-fitting, which gives it better performance [Bentéjac, Csörgő, and Martínez-Muñoz, 2021]. The specific loss function used in XGBoost is called the "gradient of the gradient" (or "hessian")

and is represented mathematically as:

$$\frac{\partial^2 L(y_i, \hat{y}_i)}{\partial \hat{y}_i^2} \tag{2.12}$$

Where $L(y_i, \hat{y}_i)$ represents the loss function, y_i represents the true label and $\hat{y}_i$ represents the predicted label. In addition to this, XGBoost uses a regularisation term called "L1" and "L2" [Moore and DeNero, 2011], which is added to the above loss function to prevent overfitting; this regularisation term can be represented as [Mienye and Sun, 2022]:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{2.13}$$

Where γ and λ are regularisation parameters, T is the number of trees in the model, and w_j is the score assigned to each leaf node.

- **Stacking** is a more complex ensemble learning method that involves training multiple/different models, then using their predictions as input to a final model. This type of ensemble learning is more powerful than bagging and boosting, as it allows the different models to complement each other and make more accurate predictions [Sagi and Rokach, 2018]. Figure 2.6 illustrates the blueprint of any stacking model.

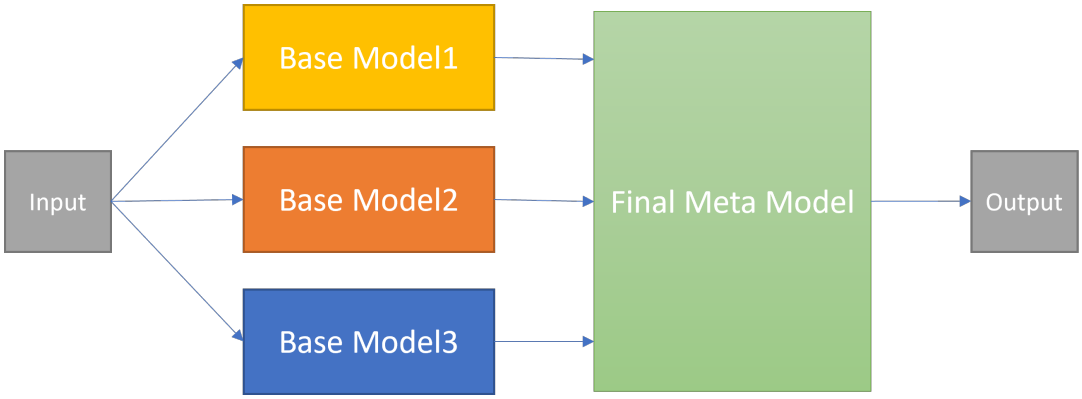

FIGURE 2.6: The blueprint of stacking models.

Stacking works by training multiple base models on the same input data, then using their predictions as input features for a higher-level model called the meta-model. The meta-model is trained to make the final prediction based on the input from the base models [Mienye and Sun, 2022]. The generic stacking algorithm is explained in Algorithm 4.

Algorithm 4 Generic Stacking Algorithm

Input: original dataset (X, y)
Output: trained meta-model
Split dataset into training and test sets (X_{train}, y_{train}, X_{test}, y_{test})
Train multiple base models on X_{train}, y_{train}
Use base models to make predictions on X_{test}
Combine predictions to generate final predictions
Train meta-model on (predictions, y_{test})
Return trained meta-model

Stacked models are known for their high accuracy and have been successful in various machine learning competitions [Ouyang et al., 2018], [Deotte et al., 2021]. Additionally, they offer improved diversity by utilising various machine learning algorithms to train the base models. An example of using stacked models would be to combine a factorisation model such as matrix factorisation with tree-based models like decision trees and random forests. This combination would provide a good diversity as the former is trained differently from the latter [Mienye and Sun, 2022]. However, a major limitation of stacked models is that they can be computationally expensive when working with large training datasets, as the entire dataset is used to train each base classifier.

In summary, Ensemble learning is a machine learning technique that combines multiple models to improve the overall performance of the system. They work together to make a final prediction. There are various ensemble learning techniques, such as bagging, boosting, and stacking. Bagging is used to reduce the variance of a single model, while boosting is used to reduce the bias. Stacking is used to combine the predictions of multiple models. Ensemble learning is widely used in practice, as it has been shown to improve the performance of many machine learning tasks.

Dynamic Ensemble of Deep Learning Approaches Following the discussion on ensemble learning, it's imperative to delve into its advanced counterpart: dynamic ensemble learning in the context of deep learning. This technique has emerged as a pivotal advancement in machine learning, offering the potential to combine the strengths of multiple deep learning models to achieve superior predictive performance. Over the past few years, several groundbreaking studies have been conducted in this domain, shedding light on its capabilities and applications. Dynamic ensemble of deep learning approaches is a new popular field of research that has the potential to improve the performance of deep learning models. In this approach, multiple deep learning models are trained on the same dataset and then combined to form a single model. The combination of models can be done in a variety of ways, such as averaging the predictions of the individual models [Zhang et al., 2014], [Ganaie et al., 2022], [Simonyan and Zisserman, 2015], or using a voting scheme [Ko, Sabourin, and Britto, 2008], [Sammut and Webb, 2010]. The advantage of dynamic ensemble of deep learning approaches is that they can

be more robust to noise and outliers than single deep learning models. Additionally, they can be more accurate than single deep learning models when the data is limited. In the upcoming section ,we review different approaches related to dynamic ensemble learning: In the realm of ensemble learning, the term "base learners" is used to refer to individual neural networks (NNs). These base learners are essentially standalone classifiers that undergo training and are then amalgamated to mitigate individual errors and enhance generalisation capabilities separately. Historically, attempts have been made to create ensembles by merging NNs, with a focus on either their accuracy or diversity, as evidenced in previous research [Hansen and Salamon, 1990], [Hashem, 1997], [Tang, Suganthan, and Yao, 2006]. It has been observed that accurate and diverse NNs have the potential to form effective ensembles that disperse errors across different regions of the input space [Brown et al., 2005, [Zhang and Zhou, 2013]. In [Alam, Siddique, and Adeli, 2020]] a new dynamic ensemble learning algorithm for designing neural network ensembles has been introduced. The proposed algorithm consists of two-step process for creating an ensemble of neural networks. The first step involves training two neural networks using different algorithms: Ash's constructive algorithm for dynamic node creation [Ash, 1989]] and Reed's pruning algorithm[Reed, 1993]. The second step involves creating separate training examples for each neural network and training them partially for a fixed number of epochs.

In [Yin, Huang, and Hao, 2015] introduced a two-stage hierarchical approach to ensemble learning known as the dynamic ensemble of ensembles (DE2). DE2 consists of component classifiers and interim ensembles, and the final DE2 is obtained through weighted averaging. [Cruz et al., 2015] implemented a two-phase dynamic ensemble selection (DES) framework. In the first phase, DES extracts meta-features from the training data, while in the second phase, DES employs a meta-classifier to evaluate the competence of the base classifier before adding it to the ensemble. [Huanhuan Chen and Xin Yao, 2009] work demonstrated that NCL treats the entire ensemble as a single machine with the aim of minimising the mean square error (MSE) and does not incorporate any form of regularisation during training. They proposed a regularised NCL (RNCL) that includes a regularisation term for the ensemble. This enables RNCL to decompose the training objectives into sub-objectives, each of which is implemented by an individual neural network. RNCL exhibits improved performance over NCL, even when dealing with datasets containing higher levels of noise. [Conroy et al., 2016] proposes a two-stage machine learning algorithm that can handle missing data without the need for data imputation. The algorithm learns a dynamic classifier ensemble from the incomplete dataset, with the goal of enhancing the accuracy of predictive models by incorporating sparsely measured features. The first step of this algorithms involves calling AdaBoost, however one potential limitation could be the reliance on a variant of AdaBoost for learning the low-dimensional classifiers. AdaBoost is known to be sensitive to noisy data, and if the missing data is imputed with noisy values, it could negatively impact the performance of the algorithm.

2.2.3 Graph Topology

As explained earlier, deep learning has revolutionised numerous fields, from image recognition to natural language processing. However, when it comes to structured data such as graphs, traditional deep learning methods fall short. This is where graph theory, a branch of mathematics, comes into play, providing the necessary tools and concepts to handle such data structures.

Graph Theory: Definitions and Mathematical Formulations Graph theory studies the properties of graphs. A graph G is defined as a pair $G := (V, E)$ comprising a set V of vertices or nodes together with a set E of edges or arcs. Each edge is a 2-element subset of V1.

The degree of a vertex is the number of edges that connect to it. For an undirected graph, the degree of a vertex v, denoted as $deg(v)$, is the number of edges incident with it. In a directed graph, we distinguish between the in-degree $deg^-(v)$ (number of incoming edges) and the out-degree $deg^+(v)$ (number of outgoing edges) [Barioli, Fallat, and Hogben, 2004].

A path in a graph is a sequence of vertices where each adjacent pair is connected by an edge. The length of the path is defined by the number of edges in the path[Barioli, Fallat, and Hogben, 2004].

Importance of Graph Theory in Deep Learning The rise of graph-structured data such as social networks, regulatory networks, citation graphs, and functional brain networks, in combination with the success of deep learning in various applications, has brought interest in generalizing deep learning models to non-Euclidean domains[ref]. In this context, graph theory plays a crucial role in developing new deep learning models that can handle such data. For instance, Graph Convolutional Networks (GCNs) are a type of neural network designed to work directly on graphs and leverage their structural information[Levie et al., 2019].

2.2.4 Federated learning

Federated learning (FL) is becoming increasingly important in the field of AIoT applications. With the proliferation of connected devices in the IoT ecosystem, federated learning offers a way to train AI models on data generated from these devices without compromising data privacy. This approach allows for the creation of AI models that can improve the performance of IoT applications, while still protecting the sensitive data generated by individual devices. Additionally, federated learning can enable the creation of personalised AI models that can adapt to the specific needs of each user or device, leading to more efficient and effective AI-based services. As such, the adoption of federated learning is crucial to realising the full potential of AIoT and unlocking the benefits of intelligent, connected devices.

The objective of the federated learning algorithm is to train a unified model using data gathered from multiple distributed devices. The algorithm works under the limitations of resources available on each device, where the model is trained locally and periodic updates of the intermediate model are sent to the cloud (server) [Imteaj et al., 2023]. The primary aim is to minimise

a loss function used for training, which can be represented by the following equation [McMahan et al., 2017]: Overall, the goal of federated learning is to optimise the following objective function:

$$\min_{\mathbf{w}} F(\mathbf{w}) = \sum_{k=1}^{K} p_k F_k(\mathbf{w}) \tag{2.14}$$

where $F_k(\mathbf{w})$ is the loss function on client C_k with model parameters $\mathbf{w}$, and p_k is the probability of selecting client C_k in each round of training. The objective is to find the global model parameters $\mathbf{w}$ that minimise the overall loss function $F(\mathbf{w})$. Algorithm 5 explains the main federated learning algorithm as explained in

Algorithm 5 Federated Learning

```
Initialize: w_0
for t = 1, 2, ..., T do
    Ct ← Randomly sample a subset of clients
    for k ∈ Ct do
        wk, t ← ClientUpdate(wk, t − 1, Dk)
    end for
    wt ← 1/|Ct| ∑ k ∈ Ct wk, t
end for

return w_T
```

In this algorithm, $\mathcal{C}t$ represents the set of clients selected at iteration t, and $|\mathcal{C}t|$ is the number of clients in the set. The `ClientUpdate` function denotes the local training step on client C_k, which updates the model parameters from $\mathbf{w}k, t-1$ to $\mathbf{w}k, t$. The final output of the algorithm is the global model parameters $\mathbf{w}_T$ at the end of T iterations.

[McMahan et al., 2017] introduced the terminology of Federated Learning (FL) along with an algorithm called Federated Averaging (FedAvg). FedAvg comprises of multiple communication rounds between clients and the server, interspersed with several local model update steps taken by each client. The main difference between the general Federated Learning algorithm and FedAvg is in the model aggregation step. In FedAvg, the model parameters are aggregated using a weighted average that takes into account the number of data points on each client. This ensures that clients with more data have a greater influence on the global model, which can lead to better performance on the overall task. The model aggregation function is presented in Equation 2.15 [McMahan et al., 2017]:

$$\mathbf{w}t \leftarrow \sum k = 1^K \frac{n_k}{n} \mathbf{w}k, t \tag{2.15}$$

where n_k is the size of client k's dataset and $n = \sum k = 1^K n_k$ is the total number of data points in all clients' datasets. FedAvg also introduces a few other modifications to the general Federated Learning algorithm, such as the use of

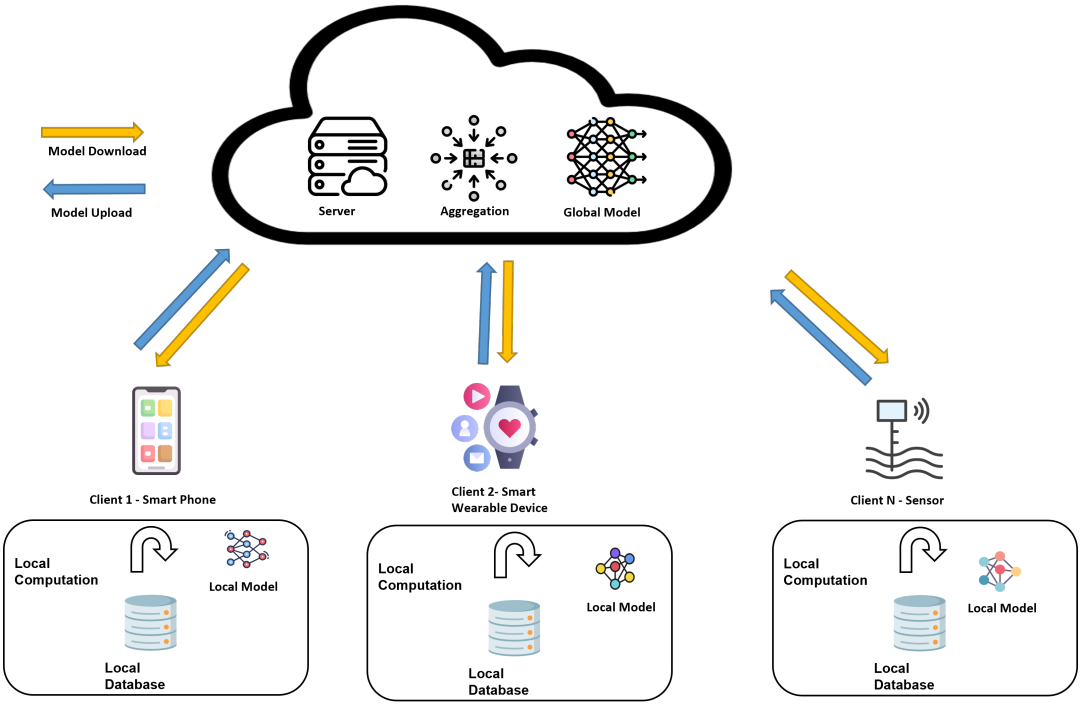

FIGURE 2.7: A typical FL-AIoT system with N participants

momentum and local clipping to improve convergence and prevent the update steps from being too large. These modifications are aimed at addressing some of the challenges that arise when training models on distributed data sources with varying degrees of heterogeneity and non-IIDness [Li et al., 2020b].

Most recently, FL has recently emerged as a promising approach for constructing IoT systems that are both intelligent and privacy-preserving [Zhai et al., 2021], [Sun et al., 2020], [Mills, Hu, and Min, 2020]. For example, different mobile devices could collaborate together in order to improve the search query suggestions (used by Google Keyboard) [Yang et al., 2018b]. There are also other benefits to applying Federated Learning in the AIoT domain, such as: a) **Privacy Enhancement:** FL allows for model training without the need for raw data at the aggregator. This minimises the risk of sensitive user information being leaked to external third parties, which in turn provides a degree of data privacy. Thus using FL for applications that work under Data Protection Regulations like (GDPR) could be very useful; b) **Low-latency Network Communication:** The use of FL technology can decrease communication latencies that arise from data transmissions because there is no obligation to transmit generated IoT data immediately to the server. This also results in conserving network resources, such as spectrum and transmit power, during data training [Chen et al., 2021]. The FL concept within IoT networks comprises two primary components: data clients, which refer to IoT devices, and an aggregation server situated at a base station as demonstrated in Figure 2.7

While federated learning offers many benefits such as privacy preservation, it also presents several challenges, particularly when dealing with non-iid (non-independent and identically distributed) datasets. In the context of AIoT, data generated by IoT devices is often non-iid, meaning that the data distribution across different devices is different. This heterogeneity poses a significant challenge in aggregating local models to create a global model that can generalise well on unseen data. Therefore, addressing the non-iid challenge is crucial to achieving the full potential of federated learning in AIoT applications. Research efforts are currently focused on developing new algorithms and methodologies that can effectively handle non-iid data in federated learning. In the following subsection, we will delve into the impact of the non-iid challenge and explore several approaches that have been proposed to tackle it.

The Effect of Non-independently and Identically Distributed (non-IID data) in Federated Learning

FedAvg is one of the first central aggregation strategies that orchestrates the distributed federated learning process [McMahan et al., 2017].

FedAvg employs SGD (Stochastic Gradient Decent) to optimise the averaged weights from the clients. However, SGD requires IID sampling of the training data sets to generate an unbiased estimate of the full gradient [Bottou, 2010]. In real-world applications, it's unrealistic to assume that the data on edge devices is following IID distribution. Actually, dealing with non-IID datasets is a key challenge in federated learning [Li et al., 2020a]. Due to the statistical heterogeneity between the clients' datasets, the distribution of each local dataset is different from the global distribution (drift in local updates). As a result, each client's objective is inconsistent with the global optima. Furthermore, large local updates (a large number of epochs [epochs refer to the number of times the entire training dataset is passed during the training process].) lead to significant differences between the averaged model and global optima, leading to low accuracy in non-IID settings [Li et al., 2021c]. Figure 2.8 explains the issue of FedAvg under non-iid settings.

As shown in the figure above, if a client performs different local updates, then the updated global model $w^{(t+1,0)}$ stays close to the local minimum w_1* rather than straying toward the true global minimum x^*.

Approaches to Deal With Non-IID Data in Federated Learning FedAvg suffers when the heterogeneity of the data is different between clients [Karimireddy et al., 2021] (non-IID datasets). This happens because the distribution of each local dataset is very different from the global distribution. Different works have been proposed to optimise FL on non-IID datasets, we divide them into different categories and summarise the most recent ones:

Data-based Approaches Data-based approaches could be divided into two main categories, namely data sharing and data augmentation. **Data sharing** is a simple, yet effective approach to tackle statistical heterogeneity in non-iid settings. [Zhao et al., 2018] proposes to create a subset of the

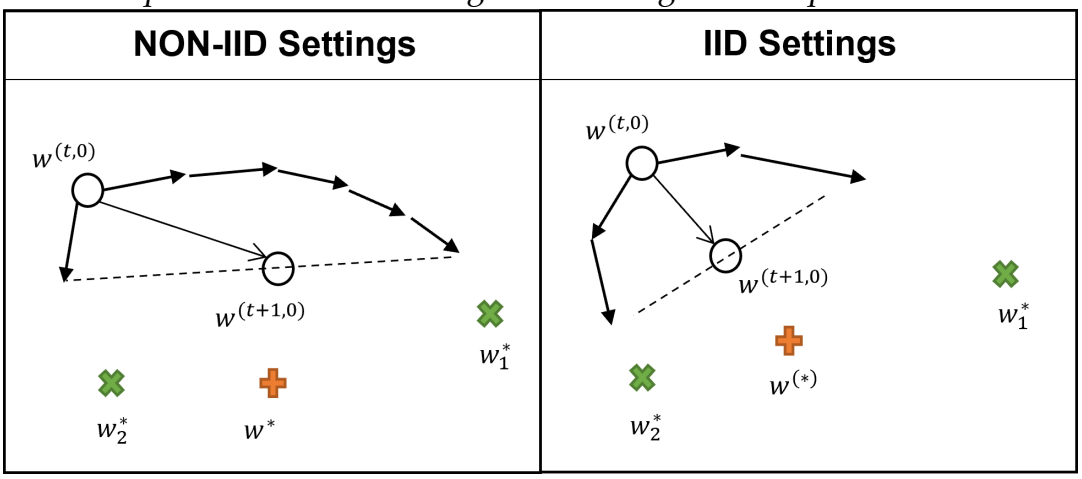

FIGURE 2.8: Model updates in the parameter space. Orange and green refer to the minima of global and local objectives, respectively.

data to be globally shared between the clients, hence generalising the learning task. The experiments on the CIFAR10 dataset show that the accuracy could be increased up to 30% while globally sharing only 5% of the dataset. Similarly, [Tuor et al., 2021] developed a mechanism to select a global subset of the client's data to be used in a federated learning task which is popular in some domains like the health sector. There are notable deficiencies in the data-sharing approaches. Initially, obtaining a uniformly distributed global dataset is challenging due to the server's lack of knowledge regarding data distributions among connected clients. Additionally, distributing segments of the global dataset to each client for model training goes against the fundamental motivation of privacy-preserving learning, which is a requirement for this process.

Data augmentation methods are a set of techniques to increase the number of data samples by applying different transformations. Its mainly used to mitigate the issues of using imbalanced datasets in ML applications [Dao et al., n.d.]. [Duan et al., 2019] uses data augmentation to develop a self-balancing federated learning framework that outperforms FedAvg on imbalanced EMNIST and imbalanced CINIC-10 datasets. To address the issue of Non-IID data, XorMixFL framework has been introduced by [Shin et al., 2020] which applies a data augmentation technique. The basic concept in XorMixFL is that each client shares its encoded seed samples (encoded through the XOR operator) with the server for decoding. A new balanced dataset is constructed by combining the decoded samples with the base data samples on the server. Subsequently, a global model is trained on this reconstructed data, which is then downloaded to each client until the training is complete. In [Yoon et al., 2021], on the other hand, the authors suggest a mean augmented method, which involves exchanging locally averaged batch data with the server. The mean data received is then combined and transferred back to each client, reducing the degree of local data imbalance. In general, the use of data augmentation techniques can greatly enhance the learning performance of models trained on Non-IID data by replenishing the imbalanced local data with augmentations. Nevertheless, as mentioned

above, many of these techniques require data sharing, which could potentially increase the risk of data privacy breaches.

Algorithm-based Approaches Several algorithm-based approaches were proposed in the literature as follows.

Local fine turning: The authors of FedProx [Li et al., 2020a], apply some modifications to FedAvg to allow partial information aggregation. This provides convergence guarantees when learning over data from non-identical distributions. [Li et al., 2021b] is another optimisation attempt to work on non-idd data. The approach uses local batch normalisation to alleviate the feature shift before averaging the clients' models. FedNova [Wang et al., 2021a] is another recent framework that relies on FedAvg; it normalises and scales the local updates of the clients according to their number of local steps before updating the global model.

Personalisation layers: edge clients are given the option to have a set of personalised layers that will not be shared with the server. A popular approach that falls under this category is FedMA [Wang et al., 2020b] which was originally designed to offer extra support for deep learning models, and it works by sharing the global model in a layer-wise manner. Furthermore, LG-FEDAVG [Liang et al., 2020] is a new federated learning framework that outperforms FedAvg in federated learning settings. In LG-FEDAVG, the shallow layers of the deep learning models are considered personalised layers, and the base layers of the networks are shared with the server. In contrast to LG-FEDAVG, FedPer [Arivazhagan et al., 2019] allows the shallow layers to be shared with the aggregation server. FedPer and LG-GEDAVG have resulted into good accuracy results, and they also reduce the communication cost since shallow layers are lightweight when shared over the network with the aggregation server. In general, Personalisation Layers are a promising approach to enhance accuracy in non-iid settings.However, one major drawback is that the clients are not able to release the personalisation layers.

Multi-task learning (MTL) methods: are inductive transfer approaches that aim to improve generalisation performance by learning multiple tasks simultaneously. [Smith et al., 2018] has developed MOCHA as a new framework that considers the issues of high communication cost, stragglers, and fault tolerance in distributed multi-task settings. MOCHA employs primal-dual optimisation to generate separate but related models for each client. However, primal-dual optimisation is unsuitable for non-convex problems and is limited to shallow networks.

Transfer learning (Knowledge distillation): Transfer learning allows knowledge exchange between different domains to achieve higher learning rates. Following this approach, [Chen et al., 2020] has developed FedHealth as the first federated transfer learning framework for wearable health devices. FedHealth adapts the inputs from different domains by replacing fully connected layers with an alignment layer. Similarly, [Wang et al., 2022] has developed another federated transfer learning framework for smart manufacturing with cross-domain applications (Fed-LTD). However, one of the main disadvantages of knowledge distillation is the negative transfer issue that can lower the clients' performance. Negative transfer occurs when data from the source

domain and the task contribute to reduced learning performance in the target domain [Pan and Yang, 2010]. **Client clustering:** In the literature, two primary types of secure data similarity evaluation methods have been introduced to address this issue. One method involves evaluating the similarity of the loss value, while the other main goal is to evaluate the similarity of model weights. The first similarity evaluation approach, reported in [Sattler, Müller, and Samek, 2019], [Ghosh et al., 2020], [Mansour et al., 2020], involves comparing the loss values of various cluster models. The fundamental concept behind this technique is simple: instead of creating a single global model, the server produces multiple global models and distributes all cluster models to connected clients for local empirical loss computation. Each client then updates the received cluster model with the lowest loss value and transmits it back to the server for cluster model aggregation. The second approach involves assessing the similarity of local data and clustering based on the local model weights. In [Briggs, Fan, and Andras, 2020], [Sattler, Müller, and Samek, 2021] FedAvg is employed first to train and warm up the global model, and then the model is downloaded locally to each client for local training. Then the models are sent back to the server to be clustered based on the weights. Client clustering is both necessary and justifiable to tackle non-iid challenges, and this is because merging local models trained on vastly different data can lead to negative knowledge transfer; hence, the overall performance of the shared model will decline. Furthermore, creating multiple global models instead of a single one improves the scalability and flexibility of FL systems, enabling system developers to select or combine different cluster models to suit specific tasks. Nevertheless, this method requires additional computation and communication resources for model training and testing. **Ensemble learning:** In Fed-ensemble [Shi et al., 2021a], the authors leverage ensemble learning to bring greater generalisation power to Federated Learning (FL). Fed-ensemble utilises random permutation to update a group of models and then produces the prediction through model averaging. By doing so, Fed-ensemble is able to achieve improved performance and accuracy compared to traditional FL methods.

Graph representation learning: most recently, graph representation has become a prominent topic in the ML community due to its wide applications. Different works have been proposed to use graph learning in FL. GraphFL [Wang et al., 2020a] is specifically designed to address the challenge of non-iid using a semi-supervised node classification approach based on graphs. First, GraphFL follows the training scheme of MAML (Model Agnostic Meta-Learning) to learn a global model on a server. Then, it leverages the traditional FL methods (e.g. FedAvg) to further improve generalisation on training sets. FedCG [Caldarola et al., 2021] is another framework to address the statistical heterogeneity in FL by means of GCN (Graph Convolutional Networks). FedCG consists of three major steps: a-identify the clusters that share the same data distributions; b-assign network components to the formed clusters; c-interaction with the GCN. [Zheng et al., 2021] and [Mei et al., 2019] preserve privacy in FL using similarity-based graph neural networks.

2.3 Bringing Machine Learning to the Edge

Deep neural networks (DNNs) have gained wide attention due to their remarkable performance in various application domains such as computer vision, natural language processing, and self-driving cars. The key factor responsible for their success is the incorporation of a large number of hidden layers that facilitate the learning of complex and abstract features. However, the use of numerous layers makes deep neural networks computationally expensive, and memory-intensive, and necessitates millions of floating-point operations (FLOPs) to be trained. This, in turn, increases energy consumption and inference time, making it challenging to deploy and train DNN models on AIoT environments and embedded systems.

To tackle preceding challenges related to the high computational cost and memory requirements of deep neural networks, researchers have proposed various methods to compress these models. One of the earliest attempts was made by Reed and others [Reed, 1993], who proposed two general themes for generating smaller neural networks: sensitivity methods and penalty terms. Sensitivity methods involve measuring the sensitivity of neurons based on an evaluation function and removing the less important neurons from the network. On the other hand, penalty terms involve applying a penalty factor to the objective function to make it smaller, which allows for the removal of weights with lower values. Since then, researchers have developed new algorithms and approaches to speed up and compress neural networks, making them more efficient for deployment in resource-limited environments.

These approaches can be broadly classified into two main themes: software-oriented approaches and hardware accelerators. Software-oriented approaches focus on optimising the neural network architecture and training process to reduce computational and memory requirements. Examples of such approaches include weight pruning, knowledge distillation, quantisation, and low-rank factorization. Hardware accelerators, on the other hand, aim to improve the performance of neural networks by offloading the computational tasks to specialised hardware. Examples of hardware accelerators include graphics processing units (GPUs), field-programmable gate arrays (FPGAs), and application-specific integrated circuits (ASICs).

Figure 2.9 provides an overview of the existing approaches to accelerate and compress deep neural networks. In the next section, we will review these techniques in detail and highlight their advantages and limitations.

2.3.1 Software Approaches

Software approaches for accelerating deep neural networks typically involve modifying the network architecture or adjusting the training process to produce a more efficient model. These methods can be classified into several categories such as quantisation, knowledge distillation, and pruning. In the upcoming section, we will delve deeper into the various techniques employed in software-oriented approaches for accelerating deep neural networks.

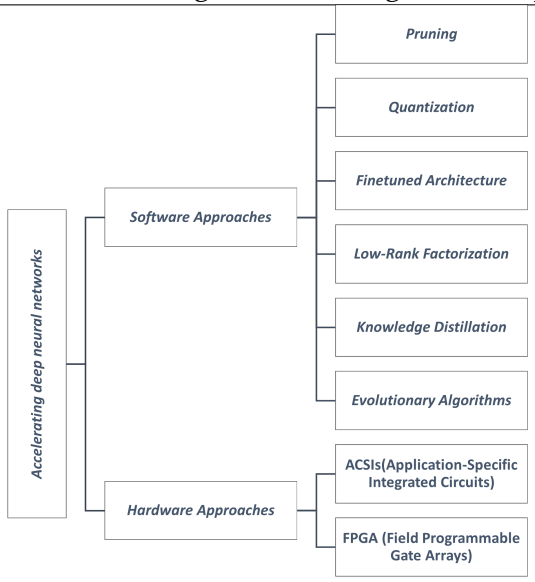

FIGURE 2.9: Classification of deep learning acceleration approaches.

Pruning

Pruning is widely used not only to reduce the complexity of a neural network but also to handle overfitting. Reed et al. [Reed, 1993] were among the first researchers who categorized the ways to generate smaller neural nets, and one of these ways is pruning. [LeCun, Denker, and Solla, 1990] and [Hassibi, Stork, and Wolff, 1993] use the Hessian matrix to prune irrelevant connections. Later [Han, Mao, and Dally, 2015] proposed a novel method to prune a network, it starts by learning the important neurons through a training process, removing less important connections, and finally fine-tuning the weights of the network. The simulation results of AlexNet show up to 9x reduction (from 61 million parameters to 6.7 million).

These techniques are further categorised into various subcategories based on their approach:

Neuron Pruning Neuron pruning is a technique utilised for compressing and accelerating deep neural networks by eliminating neurons that are considered less significant to the network's performance. The neuron pruning process can be broken down into two primary steps: Evaluation and Pruning. During the evaluation, an evaluation function is used to calculate the importance of each neuron in the network. There are various methods for evaluating the importance of neurons [Zeng and Yeung, 2006], such as:

Weight Decay Method : The approach involves the addition of a penalty value to the objective functions in order to minimise them and remove the connections with low weights. As per [Nielsen and Hansen, 2008], a new pruning algorithm for neural networks was proposed which was able to significantly reduce the dimensions of feed-forward networks while maintaining an acceptable level of accuracy. The algorithm relied on a penalty function that effectively discouraged the existence of unnecessary neurons and

minimised the weights' values. A disadvantage of the Weight Decay pruning method is that it requires careful selection of the regularisation parameter, which controls the balance between the model's complexity and the size of the weights.

Magnitude-based Pruning / MBP : MBP is a simple but effective way to remove redundant connections that are set below a certain defined value. [LeCun, Denker, and Solla, 1990], [Hassibi, Stork, and Wolff, 1993] proposed one of the very first techniques; they assumed that each connection in a neural network has a saliency (saliency refers to the importance of a neuron or a weight [Siddiqui et al., 2019]), the connections with low saliency factor are considered redundant and could be removed from the model safely . The saliency calculations depend on the Hessian matrix and derivatives of the weights. Similarly, [Hagiwara, 1994], [Wan et al., 2009] considered low weights are good candidates to be pruned away and reduce the number of connections in the network. In [Hagiwara, 1994], the authors proposed a method which involves setting a threshold value for the weights and removing those that fall below it. The authors also introduce a new criterion for removing hidden units, based on the sensitivity of the output to changes in the input. The method starts by measuring the importance of each neuron using the derivative of the network's error function with respect to the neuron's output. The neurons with low importance are then removed, and the network is retrained to fine-tune the remaining weights. The main disadvantage of MBP is that it may lead to over-pruning, where important connections may be mistakenly removed.

Cross-validation : Cross-validation is a pruning technique that relies on both magnitude-based and penalty methods, but includes an additional validation step. Initially, the dataset is divided into training and validation sets, and pruning begins. At each pruning step, the performance of the pruned model is cross-validated using the validation set. If the pruned model outperforms the original network, the pruning process continues. However, if the performance of the pruned model is worse than the original network, the pruning is stopped, and the network is restored to its previous status. The proposed approach was first introduced by [Huynh and Setiono, 2005]. The authors suggest adding a penalty value to the error function to minimise the magnitude of the weights for less important connections. The method was tested on 32 datasets, and it showed remarkable results. [Huynh and Setiono, 2005] argue that their pruning method helps the network generalise better, making it more versatile in solving diverse problems. However, Cross-validation methods are computationally expensive, especially for larger datasets or more complex models. This is because it involves training and evaluating multiple models, which can take a significant amount of time and resources [Badrinarayanan, Kendall, and Cipolla, 2017].

Mutual Information : Mutual information (MI) is as a method to control the optimal number of hidden units in neural networks. Specifically, the singular-value decomposition algorithm is used to interpret the covariance

matrix of the hidden input. A novel two-phase algorithm based on MI is proposed by [Hong-Jie Xing and Bao-Gang Hu, 2009] for pruning hidden layers and input units. In the first phase, all input features are ranked according to their relevance to the target output, and the less important ones are identified and safely removed based on their ranking and contribution to the network accuracy. In the second phase, hidden units are ranked according to their relevance measure function, and then removed one by one. Simulation results demonstrate that the proposed algorithm outperforms Support Vector Machine (SVM) and Support Vector Regression (SVR).

Similarly, [Zhang and Qiao, 2010] proposed a simple architecture for neural networks by applying Mutual Information-based methods. They suggest using the entropy of the neural net, calculated using a covariance matrix, to perform hidden node pruning. This pruning approach ensures that hidden units with low information capacity are without affecting the information capacity of the related network. One major advantage of this method is that no training is required to minimise the cost function, and pre-processing of the weights is avoided, resulting in reduced training time.

However, mutual information-based pruning can result in over-pruning of the network, leading to poor performance [Mocanu et al., 2018].

Sensitivity Analysis : A considerable amount of literature has been published on sensitivity analysis exploring its potential applications [Engelbrecht, 2001], [Yan et al., 2020], [Zeng and Yeung, 2006], [Lauret, Fock, and Mara, 2006], [Xu and Ho, 2006]. The general idea behind this approach is to measure the impact of each node or weight on the objective function and subsequently remove the least influential ones. Despite its effectiveness, one significant drawback of sensitivity analysis is the extensive training time required, even on GPU (Graphical Processing Unit). The evaluation of all neurons is a time-consuming task that could take hours, which limits its practicality in real-world scenarios. In [Yan et al., 2020], the authors proposed PIY allows for automatic pruning of neural networks by iteratively removing the least important weights until the desired network size is reached.

Table 2.3 provides a comprehensive summary of neuron pruning methods, elucidating the primary approaches behind each and presenting potential use cases that underscore their significance in model optimisation.

Channel Pruning In the channel pruning technique, the primary aim is to decrease the computational cost and accelerate the training process by substituting the over-parameterised convolutional filters with compressed filters. A (1×1) convolutional filter is used to replace a (3×3) filter to attain a top-notch performance, as reported in [Szegedy et al., 2016]. Following the same trend, [Wu et al., 2016a] presented "SqueezNet," an object detection model designed for autonomous driving, which also uses (1×1) convolutional filters instead of (3×3) filters to achieve accuracy comparable to that of the well-known AlexNet model.

While each pruning technique discussed above has its strengths and limitations, recent studies have shown that combining multiple pruning techniques can lead to better performance than using a single technique alone.

Neuron Pruning Method	**Approaches (Citations)**	**When to use**
Weight Decay Method	[Nielsen and Hansen, 2008]	Reducing dimensions of feed-forward networks for accuracy maintenance
Magnitude-based Pruning (MBP)	[LeCun, Denker, and Solla, 1990], [Hassibi, Stork, and Wolff, 1993], [Siddiqui et al., 2019], [Hagiwara, 1994], [Wan et al., 2009], [Hagiwara, 1994]	Removing redundant connections to maintain model efficiency
Cross-validation	[Huynh and Setiono, 2005], [Badrinarayanan, Kendall, and Cipolla, 2017]	Achieving better network generalisation across various problems
Mutual Information	[Hong-Jie Xing and Bao-Gang Hu, 2009], [Zhang and Qiao, 2010]	Streamlining network architectures;
Sensitivity Analysis	[Engelbrecht, 2001], [Yan et al., 2020], [Zeng and Yeung, 2006], [Lauret, Fock, and Mara, 2006]	Iterative pruning for achieving desired network sizes

TABLE 2.3: Summary of Neuron Pruning Methods

For instance, [Han, Mao, and Dally, 2015] proposed a three-step process that first identifies important neurons through network training, prunes the less important neurons, and then fine-tunes the weights through a retraining process. [Hu et al., 2016] optimises the network's architecture by removing irrelevant neurons based on statistical analysis for the neurons' activations. The approach is based on a fact: regardless of a network's input, the outputs of a significant portion of neurons in a large network are mainly zero. [He et al., 2019] proposes a filter pruning strategy relying on geometric median called FPGM, which considers the mutual relations between filters to remove the similar filter and generates compact CNNs architecture. Similarly, [Luo and Wu, 2017] suggests another filter pruning algorithm; it evaluates the importance of a deep learning model' filter using the entropy-based approach. Next, the less critical filters are ignored to generate a smaller model. [Setiono, 1997] presented a penalty function-based approach to prune deep neural networks. This approach identifies the principle nodes in one epoch using the Gram–Schmidt process and updates only the weights connected to those nodes while keeping the rest of the nodes unchanged. However, one of the major drawbacks of this method is the potential removal of important neurons, which may decrease the accuracy of the model [Van Der Baan and Jutten, 2000].

Quantisation

Quantisation is a crucial technique in the development of lightweight deep neural because it aims to reduce the number of bits required to represent the weights of neurons. This approach leads to important reductions in network size and improves the efficiency of the neural network on AIoT devices. In particular, K-means scalar quantization [Gong et al., 2014] has been demonstrated as an effective method to reduce network complexity by quantising network parameters, including weights and biases [Gong et al., 2014; Wu et al., 2016b].

Another important application of quantisation is in speeding up the training process without significant loss of accuracy. For example, [Vanhoucke, Senior, and Mao, 2011] used 8-bit linear quantisation to speed up the training process with little impact on accuracy. Another well-studied application of quantization is the work in [Han, Mao, and Dally, 2015]. The author found that applying quantisation to a pruned neural net achieves state-of-the-art performance among all quantisation methods (it reduced AlexNet from 240MB to 6.9MB). The approach presented in [Han, Mao, and Dally, 2015] depends on pruning the less important connections and then retraining the network. After that, it uses 8-bit quantization to shorten the number of bits that are used to represent the weights. Finally, it applies Huffman coding to the quantised weights to gain a more compression ratio.

Although quantisation is an essential tool to compress a neural network size, reducing the number of bits used to represent weights and activations can lead to accuracy loss. The degree of loss can depend on the complexity of the model and the level of quantization applied.

Fine-tuned Architecture

In addition to the optimisation techniques discussed earlier, deep neural networks can be accelerated by fine-tuning their architecture for better inference and performance. This is a rich area of research, with a plethora of published studies to draw upon. Noteworthy works in this field include [Lin, Chen, and Yan, 2013], [Jin, Dundar, and Culurciello, 2014], [Ioffe and Szegedy, 2015], [Wu et al., 2015], [Wu et al., 2016a], [Ghosh, 2017], [Zhang et al., 2018b]. While it is beyond the scope of this thesis to provide an exhaustive review of this literature, we highlight in the next section the most popular and recent works in this area.

Inception [Szegedy et al., 2014]: The Inception model represents a significant milestone in the development of convolutional neural network (CNN) architectures, having achieved state-of-the-art performance in both detection and classification in the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14).

Prior to Inception, most CNN designs simply stacked convolutions of the same filter size, as seen in popular models such as LeNet-5 [Lecun et al., 1998], AlexNet [Krizhevsky, Sutskever, and Hinton, 2012] and ZFNet [Zeiler and Fergus, 2013]. The Inception module's novel design involved convolutions with multiple filter sizes (1×1, 3×3, and 5×5) in a single layer, allowing for more efficient use of the input image. To reduce the computational cost of the more expensive 3×3 and 5×5 convolutions, additional 1×1 convolutions were added before them. These 1×1 convolutions also served as Rectified Linear Units (ReLU), further streamlining the network's structure and reducing computational demands. Figure 2.10 illustrate inception-v1 module. Generally, an Inception network is the composition of the previous module stacked together with max-pooling layers (stride=2) in the higher layers of the network only, leaving the lower layers as traditional convolutional layers. The main advantage of this architecture is that it allows a significant increase in the number of units without blowing up the computational complexity. GoogLeNet(the winner of ILSVRC14 competition) consists of 9 Inception modules linearly stacked, the depth is 27 layers in total(including pooling layers) and it worth to mention that the name was a homage to LetNet-5 [Lecun et al., 1998] as the work mainly inspired from it.

Inception V2 and V3 : In [Szegedy et al., 2015], the authors introduced Inception v2 and v3, which aimed to further reduce computational complexity while increasing accuracy by incorporating aggressive regularization and factorized convolutions with smaller filters. Convolutions with high filter size are considered very computationally expensive, so in **Inception v2**, the authors factorised (5×5) convolution to (3×3) convolution to boost the computational speed. In fact, the (5×5) filter is 2.78 times more computationally expensive than the (3×3) filter. Besides, they factorise the filters of size $(n \times n)$ into $(1 \times n)$ and $(n \times 1)$ convolutions. Applying this technique was 33% cheaper than applying the normal (3×3) convolution. **Inception**

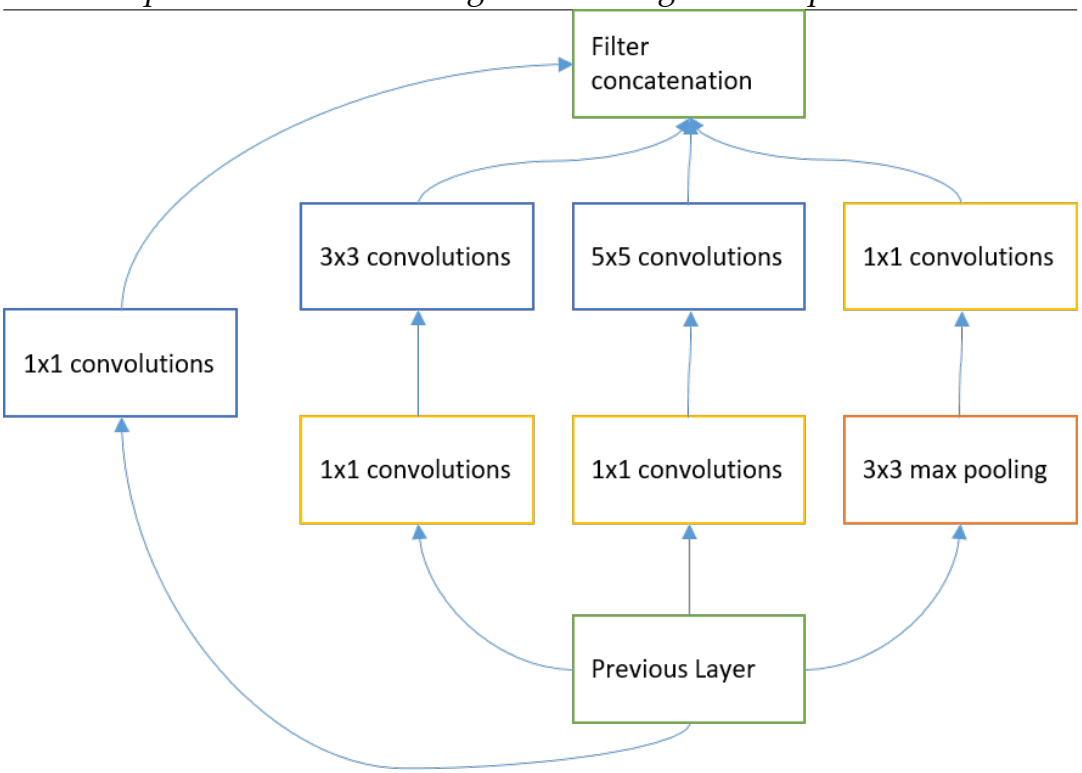

FIGURE 2.10: Inception model with dimension reductions.

v3 was provided with factorised convolutions in addition to RMSProp optimiser, label Smoothing which is a type of regularisers that is used to prevent overfitting (overfitting occurs when a model learns the training data too well, capturing noise and outliers, which leads to poor performance on unseen data due to a lack of generalization [Peng and Nagata, 2020]), and batch normalisation for the output activation.

Inception V4 [Szegedy et al., 2016]: Inception v4 is a cutting-edge deep learning architecture that combines multiple types of convolutional layers, such as standard convolutional layers, Inception modules, and residual connections. The use of residual connections also helps to improve the flow of information through the network, which can lead to better performance. One of the key innovations of Inception v4 is the "Factorised Reduction" module, which is used to reduce the number of filters in the input to the Inception module while preserving its spatial resolution. This helps to reduce the computational cost of the network while maintaining its expressive power. By using a combination of advanced techniques, Inception v4 achieves state-of-the-art performance on a variety of computer vision tasks.

Although Inception's models (V2, V3 and V4) have made a significant contribution to the field of computer vision, it is worth mentioning that the Inception family of models also have some limitations. One of the major challenges with this family of models is their interpretability. Due to their complex architectures and the use of multiple filters at different scales, it can be difficult to understand how the network is making its decisions. This lack of interpretability can be a significant obstacle in applications where transparency and explainability are important, such as in the medical or legal

fields [Wu, Sahoo, and Hoi, 2020].

ResNets [He et al., 2016a]: Residual Networks (ResNets) are a type of deep neural network that uses residual connections, also known as shortcut connections, between layers, allowing the network to learn the residual mapping between the input and output, rather than the original mapping. The "Residual Block" is a building block of the architecture and contains two or more layers with a shortcut connection that allows the input to bypass one or more of those layers. By adding the output of the block to the input and passing it through an activation function, information can flow more easily through the network, leading to improved training of deeper architectures. One issue related to ResNets is overfitting, especially in deeper architectures. This is because the additional shortcut connections can lead to more complex models that are prone to overfitting the training data [He et al., 2016b].

Xception [**chollet_xception:_2016**]: Xception is a convolutional neural network (CNN) architecture developed as an extension of the Inception architecture [Szegedy et al., 2014] for image classification and object recognition tasks. The primary innovation in Xception is the utilisation of depthwise separable convolutions that aim to reduce the computational cost of the network while maintaining its expressive power. In contrast to standard convolutional layers where a single filter is applied to all input channels, depthwise separable convolutions independently apply filters to each channel, reducing the number of parameters in the network. Xception utilises a novel variant of the Inception module called "Separable Inception" that uses depthwise separable convolutions. However, [Hendrycks and Dietterich, 2019] shows that Xception models are relatively less robust to common corruptions and perturbations, such as noise, blur, and brightness changes. This suggests that Xception may not generalise as well to real-world applications.

MobileNet [Howard et al., 2017]: MobileNet is a class of network architecture that was specifically designed for mobile and embedded devices. It allows application developers to select a small network that meets the resource constraints of their applications in terms of size and latency. Furthermore, developers can control two global hyperparameters to trade-off between latency and accuracy, allowing them to choose the optimal size of the model based on the constraints of the problem. The fundamental blocks in MobileNet are depthwise separable convolutions, which were first introduced by [SIfre and Mallat, 2014] and were later used in Inception [Szegedy et al., 2014] to reduce the number of computational operations in the first few layers of the network.

MobileNetV2 [Sandler et al., 2018]: the main difference between MobileNet V1 and V2 is the use of a new module called "Inverted Residual" which is used in the V2 version. An inverted Residuals module is a more efficient way to increase the depth and width of the network. In contrast to the traditional residual block, this module reduces the spatial resolution of the input before applying the depthwise convolution, which helps to reduce the

computation required by the network. Another key difference between MobileNet V1 and V2 is the use of linear bottlenecks. A bottleneck is a layer with a smaller number of filters than the previous layer, which helps to reduce the computational cost of the network. In MobileNet V1 [Howard et al., 2017], the bottlenecks use a 1x1 convolution to reduce the number of filters, but in MobileNet V2, linear bottlenecks are used which have no non-linearity between the depthwise and pointwise convolution, which makes the computation more efficient. Table 2.4 shows compassion between V1 and V2 in terms of the number of parameters and the number of multiply-accumulates (Macs) which indicates the number of multiplication operations to perform one inference (a larger number for MAC refers to a heavy architecture).

Network	Number Of Parameters	MAcs
MobileNetV1	4.2M	575M
MobileNetV2	3.4M	400M

TABLE 2.4: MobileNets V1 vs V2

MobileNetV3 [Howard et al., 2019]: MobileNet V3 is the third iteration of the MobileNet architecture, which uses an automated search algorithm to find the best architecture for mobile devices. Unlike its predecessors, V3 employs a different search strategy, where it uses two algorithms, MnasNet [Tan et al., 2018] and NetAdapt [Yang et al., 2018a], to achieve a better balance between computational efficiency and accuracy. MnasNet uses reinforcement learning to select the best configuration from a set of choices, and then the architecture is fine-tuned with NetAdapt to trim under-utilized activation channels in small decrements. The proposed architecture is more efficient than its predecessors and achieves state-of-the-art performance in mobile computer vision networks. MobileNetV3 is two times faster than MobileNetV2 with the same accuracy levels, which is a significant improvement. The new version also introduces the "MobileNet-EdgeTPU" module, which is optimized for edge devices. This module combines depthwise convolution, pointwise convolution, and a squeeze-and-excitation block to improve the performance of the network. The squeeze-and-excitation block allows the network to adjust the scale of the feature maps adaptively, which can lead to better accuracy. However, MobileNet family models, like many deep learning models, suffer from the interpretability problem, which makes it difficult to understand how the model reaches its decisions [Zhu et al., 2022].

AdaNet [Cortes et al., 2016]: AdaNet is a lightweight framework designed to learn neural network architecture through a combination of reinforcement learning and evolutionary AutoML techniques. Unlike other AutoML frameworks, AdaNet is not only used for finding the optimal architecture but also for learning how to create ensembles of models for better performance. The AdaNet algorithm adapts by searching for the desired neural network architecture as an ensemble of subnetworks with varying widths and depths,

creating diverse ensembles to balance performance improvement with the number of parameters. As such, AdaNet provides a flexible and adaptive approach to AutoML.

MorphNet [Gordon et al., 2017]: Rather than using AutoML to find the optimal neural network architecture, MorphNet refines an existing architecture and optimises it for the required task. It accepts a neural network as input and builds a new compressed network that is faster and able to provide better performance. MorphNet goes through a loop of shrinking and expanding phases. During the shrinking phase, it spots the unproductive neurons by calculating the cost with respect to the target resource and then applies a sparsifying regulariser. Through the expanding phase, a width multiplier [Howard et al., 2017] is used to uniformly expand all layer sizes.

EfficientNet [Tan and Le, 2019]: The EfficientNet model employs a compound scaling method that scales the network's depth, width, and resolution while maintaining computational efficiency. This is accomplished through a careful balance between these parameters and the use of cutting-edge techniques such as depthwise separable convolutions, squeeze-and-excitation blocks, and network architecture search. Each EfficientNet model is designated with a notation such as 'EfficientNet-B0', with "B" indicating the model size and "0" denoting the model's accuracy. For instance, EfficientNet-B0 represents the basic version of the architecture, while EfficientNet-B7 is the largest and most accurate model in the family. One limitation of EfficientNet is that it may not generalise well to tasks outside of image classification, as it was primarily designed and optimized for this task. This is discussed in the original EfficientNet paper [Tan and Le, 2019],

EfficientNet-EdgeTPU : Google's TPU accelerators for inferencing is becoming a very popular trend in a deep learning community. EfficientNet-EdgeTPU is a new generation of computer vision models that are derived from EfficientNet [Tan and Le, 2019] and customised to run smoothly on power-efficient hardware accelerators (eg. Google Coral). It's able to provide a real-time image classification performance and yields high accuracy levels that could only be seen in sophisticated neural network architectures running in data centres. To summarise, numerous architectures and techniques have been proposed to address the challenges of model compression, performance, and generalisation. Each technique introduces a unique approach or modification, aiming to advance the state-of-the-art and provide efficient solutions to real-world problems. Table 2.5 provides a detailed overview of the most popular fine-tuned neural network architectures.

Low-Rank Factorisation

The acceleration of Convolutional Neural Networks (CNNs) is a topic of significant interest in the field of deep learning. A number of factors can be considered in this regard, including the critical impact of convolutional operations on the training process. Due to the need to convolve all neurons,

TABLE 2.5: Summary of Finedtuned Neural Network Architectures

Model	Key Features	Remarks
Inception [Szegedy et al., 2014]	Multiple filter sizes in a single layer.	State-of-the-art in ILSVRC14.
Inception V2 & V3 [Szegedy et al., 2015]	Reduced computational complexity, aggressive regularization, and factorized convolutions.	
Inception V4 [Szegedy et al., 2016]	Integration of standard convolutions, Inception modules, and residual connections.	Introduced "Factorised Reduction".
ResNets [He et al., 2016a]	Residual connections between layers.	Prone to overfitting in deeper architectures.
Xception **[chollet_xception:_2016]**	Extension of Inception uses depthwise separable convolutions.	Might not generalize well to real-world scenarios.
MobileNet [Howard et al., 2017]	Designed for mobile and embedded devices.	Uses depthwise separable convolutions.
MobileNetV2 [Sandler et al., 2018]	"Inverted Residual" module and linear bottlenecks.	Efficiency-focused.
MobileNetV3 [Howard et al., 2019]	Uses MnasNet and NetAdapt.	Introduced "MobileNet-EdgeTPU" module.
AdaNet [Cortes et al., 2016]	Uses reinforcement learning and evolutionary AutoML techniques.	For architecture search and ensemble creation.
MorphNet [Gordon et al., 2017]	Refines existing architectures.	Uses a loop of shrinking and expanding phases.
EfficientNet [Tan and Le, 2019]	Compound scaling method.	Designed for image classification.
EfficientNet-EdgeTPU [Tan and Le, 2019]	Derived from EfficientNet.	Optimized for Google's TPU accelerators.

compressing convolutional layers is an effective approach for reducing training time. The work presented in [Jaderberg, Vedaldi, and Zisserman, 2014] aimed to achieve this goal by constructing low-rank filters in one dimension. Another approach for accelerating CNNs is to consider the convolution kernels, which are typically expressed as four-dimensional tensors [1] It is believed that these tensors contain a large number of redundant connections, and thus reducing the number of neurons in them can improve compression

[1] a tensor is a mathematical object that generalises matrices to higher dimensions. It's a container which can house data in N dimensions, often used as a fundamental data structure by neural networks [Panagakis et al., 2021].

ratios [Rigamonti et al., 2013]. In addition to the convolutional layers, the input and output layers (fully connected) can also be represented as two-dimensional matrices. This low-rank representation can be useful for reducing calculation time, as shown in [Denton et al., 2014]. It is worth noting that all low-rank factorisation methods require decomposing filters or kernels, which can be computationally expensive. Finally, It is worth mentioning that factorisation approaches may not always be effective. For example, if the filters or kernels in a CNN already have a low-rank structure, then using low-rank factorisation approaches may not result in significant improvements in compression ratios. In fact, some studies have shown that for certain types of CNNs, low-rank factorisation can even reduce the accuracy of the final model [Dong, Gong, and Zhu, 2019].

Knowledge Distillation

is a widely used technique for compressing deep neural networks into shallower ones, which aims to transfer the knowledge learned by the weights in the original network (teacher network) into a smaller network (student network). The idea of using knowledge transfer to produce compact models was first introduced by [Buciluǎ, Caruana, and Niculescu-Mizil, 2006], and has since been adopted by many researchers as a common approach for accelerating and compressing deep neural networks. Generally, knowledge distillation approaches aim to compress large models into smaller ones by learning target features using the softmax function, which transfers knowledge from a teacher model (large network) to a student model (small network). [Hinton, Vinyals, and Dean, 2015] extended this idea by developing a new compression framework following the teacher-student paradigm. This framework compresses the teacher model (an ensemble of deep neural networks) into a student model by training the student model to predict the output of the teacher model in addition to the original classification labels. Another approach to knowledge distillation is the *FitNet* framework proposed in [Romero et al., 2014]. *FitNet* allows for the training of deeper but thinner student models using target outputs from the teacher model, guided by intermediate representations called "hints" to improve training and accuracy. It distils the teacher model's knowledge by minimising its feature map and passing it to the student model. The experiments show that *FitNet* achieves better or comparable performance to the teacher model [Hinton, Vinyals, and Dean, 2015].

In [Yim et al., 2017], the authors identify the important information in the teacher model, which is then transferred to a student model in terms of direction flows between layers. Mathematically, to calculate the flow between two layers, an FSP matrix is generated from the feature map of both layers like the following:

$$G_{i,j}(x;W) = \sum_{s=1}^{h} \sum_{t=1}^{w} \frac{F^1_{s,t,i}(x;W) \times F^2_{s,t,j}(x;W)}{h \times w} \tag{2.16}$$

The learning phase of this method consists of two steps: 1- match FSP matrices between the teacher and the student by minimising the loss function

L_{FSP},2- training for the original task.

"Net2Net" [Chen, Goodfellow, and Shlens, 2015] is another technique that was developed to accelerate the training process of deep neural nets using knowledge transfer. This acceleration tool utilises function-preserving transformations between the model's architectures to achieve the required; first, the student's model is initialised to represent the same as the teacher's function, and then function transformation is applied. There are two types of transformations; the first one is"Net2WiderNet", where the model is replaced with an identical one but with more neurons in the hidden layers (wider model); the second one is "Net2DeeperNet", which replaces the model with a deeper model that matches some of the features in the teachers. The main difference between this approach and similar pre-training approaches is the altered function that is used to represent the added layers to a neural net's model. Figure 2.11 shows the difference between "Net2Net"'s workflow and other knowledge transfer approaches. In [You et al., 2017] and

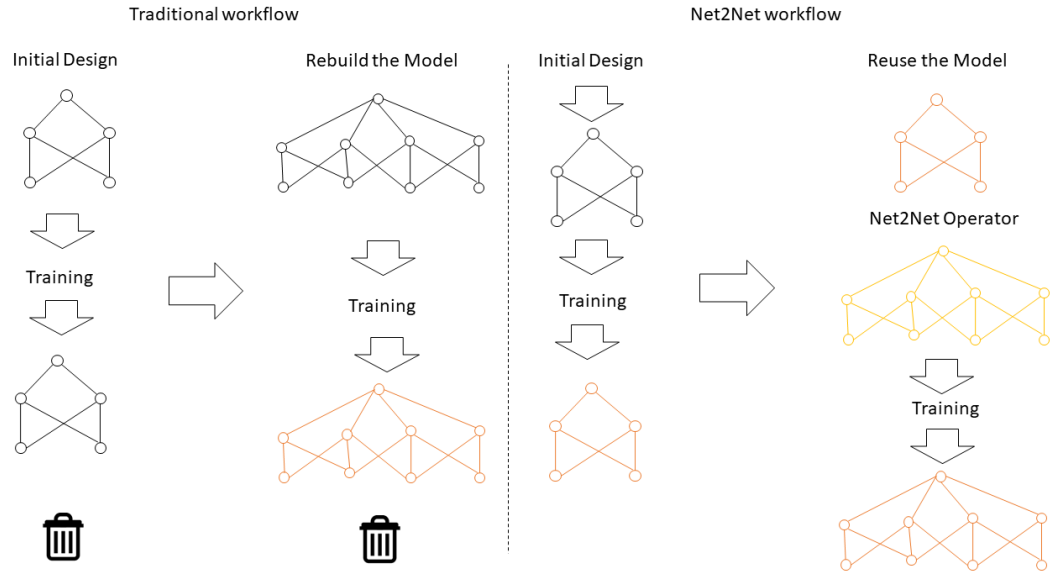

FIGURE 2.11: Net2Net vs others

instead of using one teacher model to train the student's network, multiple teacher's networks are collaborated to transfer their informative knowledge to train a thin but deep neural network. The proposed method not only averages the softened weights in the output layer for all teacher's models (dark knowledge) but also in the intermediate layers by applying dissimilarity constraint's among the available examples. Although this approach achieved remarkable results on benchmarking data sets, the computational complexity could be a critical issue when compared with the previous approaches. In a similar manner, [Sau and Balasubramanian, 2016] developed a simple approach "logit perturbation" to transfer the knowledge from multiple noisy teachers to a single student. In their experiment, two teacher models have been used: Teacher1(Network in a network Lin, Chen, and Yan, 2013), Teacher2(modified Alexnet [Krizhevsky, Sutskever, and Hinton, 2012]), after that the geometric mean of the logist for both of them has been calculated and passed to the logit value of the student's model. The error

rate results show 20.44% for the student's model while 21.94% for the student that trained using Teacher1 only and 22.62% for the student that trained using Teacher2 only, the previous results support that learning from multiple teachers could have better performance. Another interesting work in this area is introduced in by [Zhang et al., 2018c], in this work "Deep Mutual Learning (DML)" strategy based on ensemble learning is proposed. The knowledge distillation process in DML starts with an ensemble of untrained students that works together to learn a specific task and learning in such a way is significantly better than learning alone in a supervised environment.

Unlike the previous works which rely on teacher/student models to be trained separately, [lan, Zhu, and Gong, 2018] introduced an online distillation algorithm "On-the-fly Native Ensemble (ONE)" that do a batch-wise knowledge transfer in one phase only where both the student and the teacher are built simultaneously without the need for an isolated training process for both. In the training process, "ONE" builds a multi-branch target network by adding supplementary branches, and then a teacher model is constructed on-the-fly using the ensemble of all branches. In the evaluation process, the supplementary branches are removed and the trained model has converted from a multi-branch to a single-branch again with zero-cost in time increase. The experiences on CIFAR10/100 show that "ONE" was not only able to improve the generalisation and the quality of the distilled model, but also reduce the computational complexity.

However, knowledge distillation techniques have their own limitations. Recent studies have shown that knowledge distillation can result in a loss of diversity in the learned representations and reduce the generalisation performance of the student model [Furlanello et al., 2018]. The subsequent Table 2.6 offers a detailed summary of the leading knowledge distillation methods.

Evolutionary algorithms

Evolutionary algorithms have been widely used for finding optimal solutions in solution spaces through the application of biological evolution operators, such as crossover, mutation, and selection [Deb et al., 2002]. There are different types of evolutionary-based techniques; however, genetic algorithms (GAs) are the most popular ones [Hrstka et al., 2003] and they are mainly used to evolve candidate solutions, followed by a selection process that eliminates weak candidates and preserves good ones [Beasley and Chu, 1996].

Evolutionary-based techniques have been extensively employed in the area of deep learning [Schaffer, Whitley, and Eshelman, 1992], [Angeline, Saunders, and Pollack, 1994], [Yao and Liu, 1997], [Stanley and Miikkulainen, 2002], [Ding et al., 2013], [Xie and Yuille, 2017], including the use of genetic algorithms for pruning neural networks, as demonstrated by Whitley [D. Whitley, 1990], who generated multiple versions of pruned models through reproduction, mutation, and crossover.

Similarly, [(Zhang, 1993)] developed an approach to simplify the architecture of feed-forward neural networks by applying GA (Genetic Algorithms). The problem of finding the optimal architecture for a neural network model is considered a multi-objective optimisation problem, and the solution space

TABLE 2.6: Summary of Knowledge Distillation Techniques

Technique	Description	Remarks
Initial Idea [Buciluǎ, Caruana, and Niculescu-Mizil, 2006]	Compression by knowledge transfer to produce compact models.	Introduced the concept of transferring knowledge.
Hinton's Approach [Hinton, Vinyals, and Dean, 2015]	Teacher-student paradigm where student predicts teacher's output and original labels.	Extended knowledge transfer idea.
FitNet [Romero et al., 2014]	Trains deeper, thinner student models using hints from teacher's intermediate representations.	Improved training and accuracy.
Directional Flow [Yim et al., 2017]	Transfers important info from teacher to student in terms of direction flows between layers.	Utilizes FSP matrices.
Net2Net [Chen, Goodfellow, and Shlens, 2015]	Accelerates deep net training using function-preserving transformations.	Includes "Net2WiderNet" and "Net2DeeperNet" transformations.
Multi-Teacher Approach [You et al., 2017]	Multiple teachers transfer knowledge to a deep student network.	Averages weights in output and intermediate layers.
Logit Perturbation [Sau and Balasubramanian, 2016]	Transfers knowledge from multiple noisy teachers to a single student.	Learning from multiple teachers can improve performance.
Deep Mutual Learning (DML)Zhang et al., 2018c	Ensemble of untrained students learn a task together.	Better than learning alone in a supervised environment.
On-the-fly Native Ensemble (ONE) [lan, Zhu, and Gong, 2018]	Online distillation with batch-wise transfer, building student and teacher simultaneously.	Improves generalisation and reduces complexity.

is built using all the possible combinations of hidden nodes and layers, then GA is applied to search for the optimal compressed architecture through the existing solutions. In [Siebel, Botel, and Sommer, 2009], an evolutionary pruning method "Neuro-Evolution" has been introduced to reduce the complexity of a neural network by removing the less important neurons; the process of pruning is done in conjunction with an evolutionary reinforcement learning algorithm "EANT2" [Siebel and Sommer, 2007]. As an evolutionary-based method, "EANT2" has an optimisation process that selects the best

candidates among the existing neurons to be part of the next optimal model; during this process, the less important connections are identified and pruned without any extra calculations. The pruning strategy is inspired by the work in Optimal Brain Surgeon [Hassibi, Stork, and Wolff, 1993] and it depends on calculating the covariance of the network's parameters using Hessian matrix and then CMA-ES [Hansen and Ostermeier, 2001] is adopted for parameters' optimisation. The main algorithm of "EANT2" is presented in Figure 2.12. In [Hu et al., 2018], a novel approach to pruning convolutional neural net-

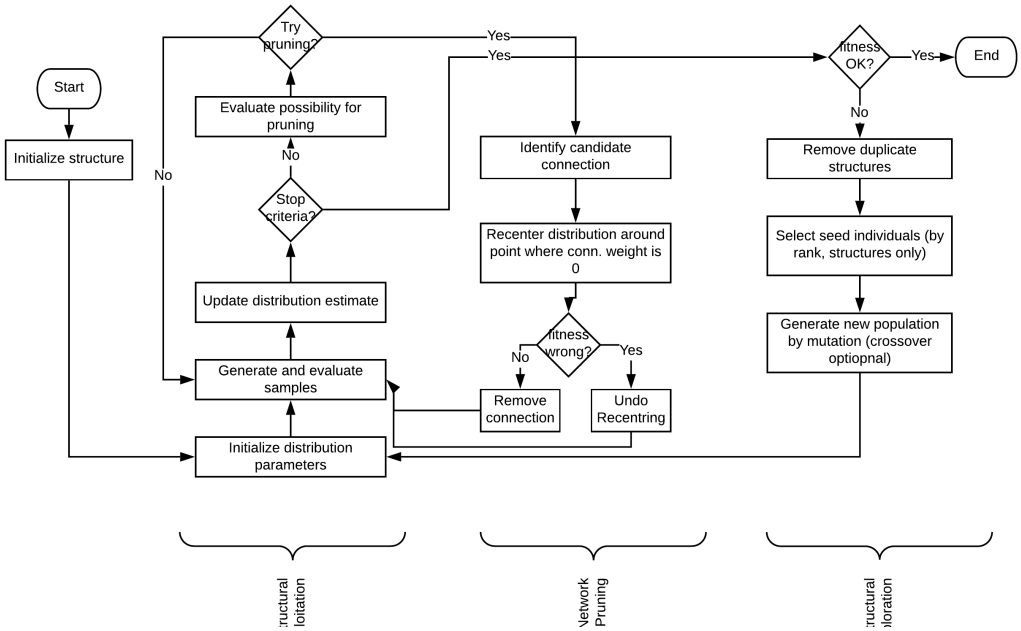

FIGURE 2.12: The EANT2 algorithm with pruning during structural exploitation.

works (CNNs) using genetic algorithms is proposed. The approach involves layer-by-layer pruning for a CNN model according to each layer's sensitivity, tuning the pruned model using the knowledge distillation framework, formulating the channel selection process as a search problem solvable efficiently using genetic algorithms, and using a two-step approximation fitness function to add extra efficiency to the genetic process.

One major drawback of using Genetic Algorithms (GA) for deep learning pruning is the high computational cost required to search for the optimal solution in a large solution space. As the number of possible network architectures and parameters increases, the computational complexity of the GA algorithm grows exponentially, which limits the scalability of the approach. This can lead to long training times and significant computational resources, which may not be feasible for large-scale deep learning models. For example, [Hu et al., 2018] noted that the computational cost of the GA-based approach is much higher than that of traditional pruning methods, due to the need to search for the optimal network architecture in a large solution space. The authors proposed a number of techniques to mitigate this issue, including using a layer-by-layer pruning approach and a two-step approximation fitness function, but noted that these techniques only partially address the scalability issue.

In order to provide a consolidated view of the discussed evolutionary algorithms and their application in deep learning, Table 2.7 is presented below. This table delineates the primary techniques, their brief descriptions, and the associated references.

Technique	Description	Reference
Genetic Algorithms (GAs)	Used to evolve candidate solutions; applies selection to eliminate weak candidates and preserve strong ones.	[Hrstka et al., 2003], [Beasley and Chu, 1996]
Neural Network Pruning with GAs	Multiple versions of pruned models generated using reproduction, mutation, and crossover.	[D. Whitley, 1990]
Simplified Architecture of Feed-forward Neural Networks with GA	Solution space constructed using all possible combinations of nodes and layers; GA used to find optimal compressed architecture.	[(Zhang, 1993)]
Neuro-Evolution (EANT2)	Evolutionary pruning method to reduce neural network complexity by removing less important neurons. Uses an evolutionary reinforcement learning algorithm.	[Siebel, Botel, and Sommer, 2009], [Siebel and Sommer, 2007]
Pruning CNNs using GAs	Layer-by-layer pruning of CNN according to each layer's sensitivity, channel selection as a search problem using GAs.	[Hu et al., 2018]

TABLE 2.7: Overview of Evolutionary Algorithms in Deep Learning.

2.3.2 Hardware Approaches

As the field of deep learning has advanced, the demand for processing power has become increasingly high. In order to achieve faster training and inference of deep neural networks, hardware acceleration techniques have been developed to leverage specialised hardware resources. These hardware approaches have the potential to significantly speed up the computations involved in deep learning and allow for larger and more complex models to be trained in a reasonable amount of time. Some of the most common hardware acceleration approaches include:

Graphics Processing Units (GPUs) : GPUs are specialised processors designed to handle the massive amount of parallel computation required for Deep Neural Networks (DNNs). GPUs have thousands of small cores that can perform many simple operations in parallel, making them well-suited for the matrix and vector operations that are common in DNNs [Nickolls and Dally, 2010]. As a result, GPUs have become the de facto standard for training large-scale DNNs. The development of GPU-based frameworks for DNNs, such as TensorFlow [Abadi et al., 2016], PyTorch [Paszke et al., 2019], and Caffe [Jia et al., 2014], has played a critical role in the widespread adoption of DNNs, making it easier for researchers and practitioners to design and train complex models efficiently. The use of GPUs has led to significant speedups in training times for DNNs, enabling the development of more accurate and sophisticated models that were previously impossible to train in a reasonable amount of time. However, GPUs have limitations in terms of consuming huge power, memory bandwidth, and requiring massive memory size, which can limit their scalability for larger DNN models.

Tensor Processing Units (TPUs) : Various hardware approaches have been developed to accelerate the training and inference of DNNs. One of the most promising approaches is the use of Tensor Processing Units (TPUs) developed by Google [Jouppi et al., 2017]. TPUs are application-specific integrated circuits (ASICs) designed to accelerate the inference and training of DNNs. They are particularly suited for large-scale, highly parallel DNN workloads, as they can achieve up to a 15x speedup over GPU-based solutions [Jouppi et al., 2017]. In addition, TPUs can be easily integrated with popular DNN frameworks, including TensorFlow and PyTorch [Paszke et al., 2019]. Google has also released Coral, a TPU-based USB accelerator that enables developers to integrate TPU acceleration into edge devices such as cameras and robots [*Products* n.d.]. This makes it possible to perform high-performance machine learning tasks locally without relying on cloud-based solutions such as latency and security [Xu et al., 2018].

Application-specific Integrated Circuits (ACSIs) : ASICs refer to a specialized category of integrated circuits (IC) that are tailored to perform a specific function, as opposed to being designed for general-purpose use. A noteworthy example of the ASIC approach was presented by [Farabet et al., 2010], who proposed a highly scalable hardware accelerator for processing large convolutional neural networks. This accelerator consists of a Central Processing Unit (CPU) that controls the configuration buses, a group of processing tiles for performing macroscopic operators, and a dual-port memory streaming engine that facilitates multiple and parallel operations on photos. This research work inspired [Pham et al., 2012] to follow the "NeuFlow" architecture proposed by [Farabet et al., 2011]. The aim of this system is to efficiently detect, classify, and locate objects in complex scenes, leveraging a Xilinx Virtex 6 FPGA platform. The system consumes only 10 watts of power, which is significantly less than a laptop computer, while providing a speed increase of up to 100 times in real-world applications.

Field Programmable Gate Arrays (FPGA) The traditional CPUs and GPUs are not the optimal solutions when it comes to handling massive parallel computations because it's leading to high energy consumption and performance bottlenecks. To overcome these challenges, researchers have explored the use of specialised hardware accelerators such as Field Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs). FPGA is rising as a promising hardware accelerator for accelerating deep learning algorithms due to its special characteristics (customisation, high computational capabilities, low energy consumption) [Mittal, 2018]. Many researchers investigated the effect of FPGA on neural nets, studies showing promising results [Ordoñez-Cardenas and Romero-Troncoso, 2008], [Peemen et al., 2013], [Rice, Taha, and Vutsinas, 2009], [George and Hawkins, 2005]. [Farabet et al., 2011] develop a CoVNet processor using low-end DSP FPGA (field programmable gate array). The system consists of one FPGA and an external memory unit; the processor uses a network compiler software to compile a trained CNN (Convolutional Neural Net) into a sequence of instructions, this processor was employed to recognise faces and could be easily applied on embedded devices. However, one of the main disadvantages of FPGAs is their higher development cost and longer time-to-market compared to ASICs. This is due to the need for hardware designers to program and verify the functionality of the FPGA, which can be a time-consuming and complex process [Mittal, 2016].

2.4 Artificial Intelligence of Things Main Applications

The adoption of AIoT applications has expanded significantly, with a significant impact on daily life quality and economic growth. This section lists scenarios for AIoT applications in different domains such as smart homes, smart healthcare, smart agriculture, smart grids, smart environment and the Internet of Vehicles. Table 2.4 summarises the different application categories with recent studies.

2.4.1 Smart Health Systems

The emergence of AIoT enabled by edge computing introduces a novel avenue for exploration in the medical and healthcare sector. This innovation has already yielded various applications that include health monitoring systems, and disease diagnosis systems. **Health Monitoring Systems**: AIoT presents a remarkable prospect for creating intelligent health monitoring systems that can monitor patients' health conditions. Most recently, we have started to witness remarkable advances in the smartwatches industry that integrates casual watch functions and smart sensors. Different sensors are used to capture various measurements about our bodies like blood pressure, ECG, oxygen level and even our sleep cycles [Falter et al., 2019]. Such devices are also equipped with diagnosis functions and sounding warnings in real-time to provide early warnings for potential health problems [Ahad, Tahir, and Yau, 2019]. The authors in [Zhang et al., 2020] have developed smart socks to

detect early falling and monitor the symptoms of Parkinson's disease [Bloem, Okun, and Klein, 2021]. The socks are equipped with sensors that capture body signals and transmit the data to nearby smart mobile phones in real-time. This allows other applications to perform advanced analysis for the submitted data using AI. **Disease Diagnosis Systems**: The recent advances in AIoT has to lead to a new generation of AI-Based applications that help doctors in medical diagnosis. In [Dilsizian and Siegel, 2013], (AI) is being used in medicine and cardiac imaging to provide personalised diagnosis and treatment. HealthFog [Tuli et al., 2020] is another smart healthcare system that uses ensemble deep learning to automatically diagnose heart diseases in integrated IoT and fog computing environments. HealthFog integrates multiple data sources, including medical images and patient data, to make accurate predictions. [Sood and Mahajan, 2018] has proposed another fog-based healthcare system for the diagnosis and treatment of Chikungunya (a mosquito-borne viral disease that can cause severe joint pain and fever). The framework uses a combination of sensors, fog nodes, and cloud resources to collect and analyse patient data in real-time. Nowadays, patients and surgeons have embraced Robot-assisted surgery (RAS) as a form of adjunctive therapeutic technology. The surgical instruments, which are robotic in nature, are controlled by the surgeon's hand movements in real-time and operate with small-scale precision. [Liu et al., 2020] proposes an anchor-free convolutional neural network for real-time surgical tool detection. The CNN model is trained to detect surgical tools in a live video stream without relying on pre-defined anchor boxes. The proposed approach achieves high detection accuracy while maintaining real-time performance, making it suitable for integration into robot-assisted surgical systems. In a similar manner, [Qiu et al., 2020] has developed a real-time tracking system using deep regression networks for robotic ophthalmic surgery. The system tracks the position of the iris in the eye, allowing for precise and safe surgical procedures.

2.4.2 Smart Homes

The utilisation of intelligent technologies such as sensors, actuators, and artificial intelligence (AI) within our homes and buildings, has increased the quality of our lives, safety and productivity [Shi et al., 2021b], [Sepasgozar et al., 2019], [Hong, Shin, and Lee, 2016], [Marikyan, Papagiannidis, and Alamanos, 2019], [Sovacool and Furszyfer Del Rio, 2020].

[Berrezueta-Guzman et al., 2020] describes the development and evaluation of a smart home environment designed to support homework activities for children. The environment consists of various technologies such as smart lighting, interactive surfaces, and speech recognition. The study found that the smart home environment positively impacted children's engagement with homework and improved their academic performance. [Wang, Gong, and Liu, 2019] proposes a deep learning-based approach to improve human activity recognition using WiFi-based data. Specifically, the authors investigate the use of spatial diversity, which involves collecting data from multiple WiFi access points to improve activity recognition accuracy. They compare the performance of their approach to existing methods and show that spatial diversity can significantly improve recognition accuracy. Additionally, [Zou

et al., 2018] has developed an occupancy-driven lighting control system for smart buildings. The system uses WiFi signals to detect and track the occupancy of a room and adjusts the lighting accordingly. The authors rely on a machine learning algorithm for occupancy detection and a cloud-based platform for control and monitoring. [Shi et al., 2020] introduces a novel floor monitoring system that uses smart mats and deep learning technology to detect and classify human activities. The system is scalable and can be used in various applications, such as healthcare and home automation. In the same vein, a lot of attention has been directed to developing smart home appliances as they offer the potential to improve energy efficiency and increase home security. According to [Zhou et al., 2016], smart home appliances can improve energy efficiency and reduce household energy consumption. Also, [Zhou et al., 2016] have found that smart appliances, such as smart thermostats and lighting, can significantly reduce energy consumption without compromising comfort. However, some security and privacy risks are associated with smart home appliances [Panwar et al., 2019].

2.4.3 Smart Transportation

AIoT applications in transportation systems aim to improve road safety, minimise crash collisions and relieve traffic congestion. Autonomous driving, also known as self-driving cars, refers to the technology in smart transportation that enables a vehicle to operate without human intervention. This technology is based on a combination of advanced sensors, software, and machine learning algorithms that allow the vehicle to sense its environment and make decisions based on the data it collects [Talebpour and Mahmassani, 2016]. HydraMini [Wu et al., 2020] and HydraOne[Wang et al., 2019] serve as typical examples of how embedded computing platforms are used to enhance the decision-making capabilities of autonomous driving technology. HydraMinin and HydraOne are equipped with embedded computing platforms that enable them to support AI (e.g., CNN) inference and traditional computer vision analysis. This enables the vehicles to make real-time decisions, such as shifting, braking, throttling, and steering based on the data they collect. In addition, there have been advancements in developing advanced driver assistance systems (ADASs) aimed at helping drivers. For instance, EdgeDrive is a cloud-based system that operates at the network's edge and is designed to provide drivers with real-time ADAS applications such as smart navigation while driving [Maheshwari et al., 2019]. The development of such systems marks a significant step forward in enhancing road safety and reducing the risk of accidents on the road. In [Villanueva et al., 2019] another assistance system is developed for detecting somnolence (drowsiness) in drivers using a deep neural network. The proposed system uses a camera to capture images of the driver's face, which are then analysed by the deep neural network to determine the level of drowsiness.

Domain	References	Primary Objective
Health Care	[Falter et al., 2019], [Ahad, Tahir, and Yau, 2019], [Zhang et al., 2020]	Health monitoring systems that utilise advanced sensors to monitor patients' conditions.
	[Tuli et al., 2020], [Sood and Mahajan, 2018], [Dilsizian and Siegel, 2013]	Medical Diagnosis systems to support doctor in taking decisions.
	[Qiu et al., 2020], [Liu et al., 2020]	Robot-assisted Surgery systems
Smart Homes	[Vita et al., 2020], [Wang, Gong, and Liu, 2019]	Smart home Monitoring Systems
	[Zou et al., 2018], [Shi et al., 2020]	Control System for Buildings
	[Zhou et al., 2016], [Zhou et al., 2016]	Smart Home Appliances - Energy Management
Smart Transportation	[Wu et al., 2020], [Wang et al., 2019]	Autonomous Vehicles
	[Maheshwari et al., 2019],[Villanueva et al., 2019]	Driver Assistance Systems

TABLE 2.8: AIoT applications in different domains

2.5 Challenges of Deploying AIoT Applications

The AIoT is characterised by intelligent and autonomous devices that communicate with each other and with the cloud to perform complex tasks. However, the practical deployment of AIoT presents several challenges that must be addressed for successful implementation including data privacy, heterogeneity and interoperability and resource management [Chang et al., 2021]. In the next section, I will explain the main challenges of deploying AIoT applications; Table 2.9 summarises the key challenges along with potential solutions.

Security and Privacy One of the primary challenges facing the AIoT is ensuring the **security and privacy** of the vast amounts of data generated by these systems. As the number of connected devices grows, so does the risk of cyber-attacks and data breaches. Effective security mechanisms must be implemented to protect the privacy and integrity of sensitive data. Researchers have proposed various security solutions, including blockchain-based approaches that provide secure and decentralised storage of data [Zhao et al., 2019]. Some researchers are combining blockchain technology with the Internet of Things (IoT) in the context of 6G communication networks. This approach has shown great potential for data storage and analytics [Sekaran et al., 2020]. An alternative approach is to rely on deep learning algorithms; for

example in [Wang et al., 2021b] a novel deep learning data privacy protection scheme based on homomorphic encryption is proposed. The scheme aims to provide secure data sharing and analysis while preserving data privacy. Experimental results demonstrate that the proposed scheme achieves high accuracy and low computational overhead. In the same direction, [Xiong et al., 2022] has developed the 2DP-FL algorithm that incorporates differential privacy by adding noise during both the training of local models and the distribution of the global model. Most recently, Federated Learning (FL) has emerged as a promising solution to develop intelligent and privacy-preserving IoT systems. As discussed earlier in this chapter, FL is a decentralized and collaborative method of artificial intelligence, enabling data training by coordinating numerous devices with a central server without sharing actual datasets [Konečný et al., 2017].

Cooperative Mode of Operation Another critical challenge related to deploying AIoT applications is finding a cooperative mode of operation among the diverse devices involved in the AIoT ecosystem, including end devices, edge servers, and cloud services. Each device has unique strengths and limitations; balancing its capabilities is crucial to achieving efficient and effective operation. To address this challenge, researchers are exploring new approaches such as edge computing, which allows for processing data closer to the end devices, reducing latency and increasing efficiency [Khan et al., 2020].

Interoperability is also a significant challenge facing the AIoT ecosystem. As the number of manufacturers and vendors increases, ensuring compatibility and interoperability becomes increasingly complex. Standardisation of communication protocols and interfaces is necessary to ensure interoperability. Researchers have proposed various standardisation solutions, including open-source frameworks that enable interoperability among different devices and platforms [Latifi, 2022], [Souza, Souza, and Ciferri, 2022].

Resource management Resource management is also considered a major challenge when it comes to deploying IoT applications. Not all IoT devices can allocate their storage and computing resources for data training, as some of them have limited computational capabilities that pose significant resource constraints[Tsukada, Kondo, and Matsutani, 2020]. Training deep learning models like DNN on IoT devices may not be feasible because of the high CPU and battery demands involved in solving training tasks, particularly when dealing with image and audio data [Chauhan et al., 2018]. As an example, the VGG-16 model [Simonyan and Zisserman, 2015] comprises about 138 million parameters and necessitates nearly 550 MB of memory capacity, making it impractical to make an inference on resource-limited devices like AIoT. The research community has introduced numerous techniques for compression and acceleration; The study presented in [Xu et al., 2019] suggests a new Federated Learning architecture for mobile devices that could be used to train deep neural networks taking into consideration the available computational resources. Another approach to cope with the limited

resources of IoT devices is to rely on AI accelerators. The study presented in [Lane et al., 2016] proposes a software-based deep learning accelerator that facilitates AI/DL training on mobile hardware. The main concept involves employing a range of heterogeneous processors (such as GPUs), where each computing unit utilises unique computational resources to handle various inference phases of DL models. Other techniques aim to reduce the complexity of DNNs by removing the redundant neurons [Han, Mao, and Dally, 2015], [Wan et al., 2013], [Wu et al., 2016b]. Another effective approach to reducing the complexity of DNNs is to apply quantisation. Quantisation refers to the process of reducing the number of bits needed to represent the weights of neurons. To simplify the neural network, both [Gong et al., 2014] and [Wu et al., 2016b] utilised k-means scalar quantization on the network's parameters. An alternative approach to preserve the resources of AIoT devices is Knowledge transfer or transfer learning. Knowledge Transfer is a technique that has been employed to reduce the complexity of deep neural networks. This approach involves reusing pre-trained models or their learned representations for a new task, thereby reducing the amount of training data and computation required. [Yosinski et al., 2014] demonstrated that transferring knowledge from pre-trained models can lead to better performance on tasks with limited data. Furthermore, [Pan and Yang, 2010] provides a comprehensive survey of transfer learning methods, including their applications and challenges.

Genetic Algorithms (GA) also could be utilised to prune neural networks. In [D. Whitley, 1990], it involved generating multiple versions of the pruned model using reproduction, mutation, and crossover. Similarly, Zhang [Zhang, 1993] used genetic algorithms to simplify the architecture of feed-forward neural networks. The problem of finding the optimal network architecture is a multi-objective optimization task, with the solution space containing all possible combinations of hidden nodes. The genetic algorithm is then used to search for the optimal architecture. In [Hu et al., 2018], a novel approach was presented for pruning convolutional neural networks (CNNs) using genetic algorithms. The approach involved layer-by-layer pruning based on each layer's sensitivity, followed by tuning using a knowledge distillation framework. The channel selection process was formulated as a search problem, which was efficiently solved using genetic algorithms. GA-based pruning approaches are generally superior in performance as they enhance the generalization of the pruned network and use fewer parameters.

Ethics Finally, there are ethical implications of AIoT, particularly in relation to data usage and decision-making. Ensuring that AIoT systems are transparent and accountable is critical for building trust in these technologies. Researchers have proposed various ethical solutions, including ethical guidelines and regulations that govern data collection, processing, and use in AIoT systems [Floridi, 2016]. [Allhoff and Henschke, 2018] examines foundational ethical issues related to the Internet of Things (IoT), including issues related to privacy, security, and autonomy. The authors argue that the design and deployment of IoT technologies should prioritise ethical considerations and recommend ethical guidelines for AIoT systems.

AIoT Challenge	Possible direction	References
Security and Privacy	Blockchain based approaches	[Zhao et al., 2019], [Sekaran et al., 2020]
	Deep learning	[Wang et al., 2021b]
	Federated Learning	[Konečný et al., 2017]
	Differential Privacy	[Xiong et al., 2022]
Resource Management	AI Accelerators	[Lane et al., 2016]
	Tuned Architectures	[Xu et al., 2019]
	Pruning	[Han, Mao, and Dally, 2015], [Wan et al., 2013], [Wu et al., 2016b]
	Quantisation	[Han, Mao, and Dally, 2015], [Wan et al., 2013], [Wu et al., 2016b]
	Genetic Algorithms	[Hu et al., 2018], [Hu et al., 2018]
	Knowledge Transfer	[Yosinski et al., 2014], [Pan and Yang, 2010]
Interoperability	communication standardisation	[Latifi, 2022], [Souza, Souza, and Ciferri, 2022]

TABLE 2.9: AIoT challenges and possible directions

2.6 Discussion

As highlighted in Section 2.5, the deployment of AIoT applications in real-world scenarios is fraught with challenges. Table 2.9 provides a summary of the main challenges and potential solutions to overcome them. In this section, we turn our attention to identifying the critical gaps in the existing literature that our contributions aim to address.

In recent years, deep learning algorithms have garnered significant attention owing to their ability to achieve state-of-the-art results in a wide range of domains [Krizhevsky, Sutskever, and Hinton, 2012], [Collobert et al., 2011], [Fridman et al., 2017]. However, the use of deep neural networks (DNNs) in constraint-resource devices such as AIoT can pose severe performance issues due to their high computational complexity. To mitigate this challenge, researchers have proposed various methods to compress DNNs, including using regularisers [Nowlan and Hinton, 1992],[Girosi, Jones, and Poggio, 1995], drop connect [Wan et al., 2013], pruning [He, Zhang, and Sun, 2017], [Han, Mao, and Dally, 2015], quantisation [Gong et al., 2014], [Wu et al., 2016b], and fine-tuned network design [Ghosh, 2017], [Howard et al., 2017].

Despite these compression techniques, deploying compressed DNNs in complex IoT environments remains a challenge. The noisy data generated at the edge can negatively impact the prediction accuracy of pruned models and lead to inadequate performance for IoT applications that require stable and consistent model performance. Therefore, there is a need for new approaches that can optimize DNNs for deployment on AIoT devices and

improve their robustness to noisy data generated at the edge. In the following sections, we present our contributions that address these key gaps in the literature.

Furthermore, in the traditional approach to analysing IoT data, a user's data is transferred to a central cloud server for analysis and generation of insights, as discussed in [Ghosh and Grolinger, 2020]. This method has several drawbacks, including potential security breaches and the risk of violating data privacy regulations such as the GDPR (General Data Protection Regulation) [Voigt and Bussche, 2017]. Sensitive information is at risk when transferred to a remote server, which increases the risk of a data breach. Additionally, data protection regulations are becoming increasingly strict, and organisations that fail to comply with them risk significant legal consequences. As a result, new approaches to analysing IoT data are necessary to maintain data privacy and security. To address theses limitations, we have provided three sequential novel contributions that aim to improve the deployment and performance of AIoT applications.

In Chapter 3, we propose a novel solution to address the accuracy drop of compressed models running on AIoT devices. Our approach leverages deep ensemble learning to enhance the predictability of compact DNNs. We evaluate our proposed approach on several benchmark datasets, and our results demonstrate its exceptional performance. In Chapter 4, To reduce DNNs demand in terms of memory space, we proceed to minimize the memory usage of executing *EnSyth*'s deep learning ensembles on constraint-limited devices by introducing *MicroNets*. *MicroNets* is a comprehensive framework for operating deep ensemble learning on AIoT devices. It is devised not only to run deep learning ensembles seamlessly on AIoT devices and enhance the precision of pruned deep learning models but also to amplify the generalization capability at the edge. Augmenting the generalization power at the edge is crucial because IoT devices produce noisy data owing to the surrounding environment, which can lead DNNs to overfit.

In the final chapter of our work, Chapter 5, we present *FedNets*, a solution that addresses security and privacy concerns when running sensitive applications on the edge. *FedNets* is a cutting-edge deep ensemble federated learning approach that offers robust and secure machine learning solutions on AIoT devices.

Unlike the traditional federated learning approach, where clients share their parameters over the network to update a single global model, *FedNets* proposes a different approach. Here, each client runs a set of diverse lightweight deep-learning models, similar to the approach presented in Chapter 4. The participating clients work together by sharing ensemble members instead of parameters. However, running deep ensemble learning and sharing complete models in such an environment may pose a challenge. Therefore, *FedNets* employs graph embedding theory to simplify calculations. This approach optimizes resource usage and enhances the model's accuracy without compromising the system's performance.

In non-iid settings, *FedNets* has achieved exceptional predictability performance, outperforming two of the state-of-the-art federated learning strategies. Overall, *FedNets* offers an innovative solution to security and privacy

challenges, enabling the deployment of sensitive applications on AIoT devices with confidence.

2.7 Summary

In this chapter, we present an extensive and informative overview of Artificial Intelligent Things (AIoT). We provide a detailed account of the different components involved in building AIoT applications, starting from the topology of IoT devices and their various applications, and concluding with the challenges associated with implementing AIoT applications in the real world.

We delve into the details of the various AI algorithms that can be employed in constructing intelligent things. Our discussion covers supervised learning, which involves training a model on labelled data to predict outcomes. We also explore unsupervised learning, which entails training a model on unlabelled data to identify patterns and structures. Moreover, we delve into deep learning, a subfield of machine learning that employs neural networks to learn from data, and ensemble learning, which amalgamates multiple models to improve accuracy.

Furthermore, we examine federated learning, a distributed learning approach that enables edge devices to collectively train a shared model without divulging raw data. We discuss the challenges and opportunities associated with each of these algorithms in detail, providing a comprehensive understanding of their capabilities and limitations.

We also identify the main gaps in this field and present our contribution to filling the identified gaps. Our contribution comprises innovative approaches that enhance the predictability and generalisation power of deep neural networks in compressed models.

Overall, this chapter provides a comprehensive and in-depth overview of AI and intelligent things, enabling readers to gain a better understanding of the opportunities and challenges involved in constructing intelligent devices for the AIoT ecosystem. Additionally, it highlights our contribution to this field, which seeks to address some of the identified gaps and provides novel approaches to enhance the accuracy, generalisation power, and data privacy of AIoT applications.

Chapter 3

EnSyth: A Pruning Approach to Synthesis of Deep Learning Ensembles

In the previous chapter, we undertook a comprehensive review of the rapidly evolving field of Artificial Intelligence of Things (AIoT). This review began with an exploration of the fundamental definition of IoT devices and continued on to a detailed discussion of the many challenges and gaps that currently exist in the practical application of AIoT solutions in the real world.

Building upon this foundation, in this chapter we introduce a novel deep ensemble learning technique that has been specifically designed to enhance the predictability of compact neural network models at the edge end of IoT systems. This innovative approach represents a significant breakthrough in the development of AIoT solutions, as it addresses the second and the fifth objectives outlined in Chapter 1: namely, the need to optimise performance at the network's edge, and the importance of developing robust and effective methods for dealing with the complexities of real-world data sets.

Our deep ensemble learning technique offers a range of important benefits and advantages over traditional AIoT models. By leveraging the power of ensemble learning, we are able to improve the accuracy and reliability of our models, while also minimising the impact on system resources (memory footprint). Furthermore, this approach is highly scalable, allowing it to be adapted to a wide range of different IoT devices and use cases.

Overall, our novel deep ensemble learning technique represents a significant step forward in the development of AIoT solutions. By addressing key challenges related to the practical application of AIoT technologies (resource efficiency, reliability), we are helping to pave the way for a future in which IoT devices are able to function more efficiently, effectively, and seamlessly than ever before.

The technique can be summarised as follows: first, we generate a set of diverse compressed deep learning models using different hyperparameters for a pruning method. Next, we utilize ensemble learning to synthesize the outputs of the compressed models to create a new pool of classifiers. Finally, we apply backward elimination on the generated pool to explore the best-performing combinations of models. On CIFAR-10 and CIFAR-5 datasets, the approach we propose, called *EnSyth*, outperforms the predictability of the baseline model (LetNet-5).

The approach presented in this chapter is published as a full paper titled "*EnSyth*: A Pruning Approach to Synthesis of Deep Learning Ensembles" in the 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC) [Alhalabi, Gaber, and Basurra, 2019].

3.1 Introduction

Deep learning has been the focus of significant attention in the last decade due to its impressive achievements in diverse domains [Krizhevsky, Sutskever, and Hinton, 2012], [Collobert et al., 2011], [Fridman et al., 2017]. It has emerged as a leading tool for learning and solving complex problems. The key to its success lies in its ability to leverage many hidden layers to learn different features hierarchically, with classification performed by the softmax function at the output layer. However, the complexity of deep neural networks arises from their numerous layers, making them computationally expensive. For instance, the VGG-16 model [**simonyan_very_2014**] consists of approximately 138 million parameters and requires about 550 MB of memory space. To put this into perspective, consider an average IoT device with a memory of 1 GB. Deploying the VGG-16 model on such a device would consume more than half of its entire memory space. This is just for a single model. If we consider ensemble learning, which involves training and deploying multiple models to improve accuracy, the memory requirements increase substantially. Given the limited memory of IoT devices, deploying complex models like VGG-16 or implementing ensemble learning approaches becomes impractical and often impossible. This highlights the need for more efficient models and learning techniques that can operate within the memory constraints of IoT devices.

Researchers have introduced various compression and acceleration strategies to overcome the complexity of deep neural networks generating large model sizes. Some of the compression and acceleration methods include the use of simple regularisers such as L1 and L2 to control the complexity of neural networks during training [Demir-Kavuk et al., 2011], [Cortes, Mohri, and Rostamizadeh, 2012], [Kamalov and Leung, 2020], [Zhang et al., 2021].

Another approach is drop connect methods, where a subset of the network's weights is randomly dropped, resulting in pruned networks [Wan et al., 2013]. Neuron pruning methods increase the sparsity of neural networks by removing irrelevant connections [Yu et al., 2018a], [Molchanov et al., 2019], [Hong-Jie Xing and Bao-Gang Hu, 2009], [Zhang et al., 2018c], while weight pruning removes less-contributing weights [Zhang et al., 2018a], [Han, Mao, and Dally, 2015].

Another strategy is channel pruning, where entire channels are removed instead of a single neuron's output [He, Zhang, and Sun, 2017]. Knowledge distillation transfers knowledge from a larger teacher model to a smaller student model, thereby reducing its complexity while maintaining its accuracy [Romero et al., 2014], [You et al., 2017], [Yim et al., 2017]. Network quantisation reduces the number of bits required to represent the network's weights, but it may require special hardware for acceleration [Gong et al., 2014], [Wu et al., 2016b].

Fine-tuned network design is another acceleration strategy that focuses on reducing the complexity and improving the accuracy of the entire neural network. This approach is achieved through architecture optimisation [**chollet_xception:_2016**], [Howard et al., 2017]. Lastly, genetic algorithms have been widely applied to accelerate deep neural networks [Siebel and Sommer, 2007], [Xie and Yuille, 2017], [Hu et al., 2018].

Despite the numerous compression and acceleration strategies proposed to mitigate the complexity of deep neural networks, they often come with a trade-off between accuracy and model size. In particular, pruning techniques have been successful in reducing the size of deep neural networks, but at the cost of accuracy. This has motivated research into techniques to address the accuracy drop of pruned models, such as iterative pruning and retraining [Han, Mao, and Dally, 2015]. However, these techniques can be computationally expensive and may not always lead to the desired level of accuracy. To tackle this issue, we propose a novel approach that synthesises deep learning ensembles from a baseline model, boosting its accuracy and inference time performance. This approach, called *EnSynth*, automatically generates a set of diverse models that complement each other in terms of their strengths and weaknesses. By combining these models, EnSynth achieves higher accuracy than individual pruned models and even some unpruned models, with a negligible increase in model size. It achieves this by applying multiple sets of pruning methods with varying hyperparameters, resulting in a diverse pool of pruned deep learning models. Using this pool to form ensembles can significantly boost accuracy due to the diversity awarded using different hyperparameters values. The potential number of ensembles that can be formed from a large pool of pruned models is $2^m - 1$, where m is the number of pruned models. Exploring such a large solution space for a large m can be computationally intensive, but can be accomplished using metaheuristic optimisation methods such as Genetic Algorithms [Zemouri et al., 2020], [Wang et al., 2020c], [Xu et al., 2021]. To select the best models, EnSyth uses a simple backward elimination method, where models of varying sizes $\{m, m-1, \ldots, 1\}$ are formed by sequentially eliminating the worst-performing pruning model. Inference acceleration is achieved through parallel processing, where all models are used to infer class labels independently, followed by a computationally efficient fusion for majority voting. EnSyth ensembles are instrumental in collaborative machine-learning environments that operate on resource-constrained devices, where the performance of a single model can be limited by the available resources [Gaber, Stahl, and Gomes, 2014]. The main contributions of this chapter are:

- synthesis of an ensemble of pruned deep learning models from a baseline model from a diverse space of synthesised models. The diversity of the ensemble makes it possible to outperform a baseline model, and
- eliminates the impact of compression on deep learning models by producing compressed models with better predictability measures. Through parallel processing, the approach achieved fast inference of the pruned models while boosting the accuracy through ensembling.

The chapter is organised as follows: In Section 3.2, we provide a detailed explanation of the proposed technique, *EnSyth*, which leverages ensemble learning to improve the performance of deep neural networks. We also include theoretical justifications for the proposed approach, highlighting its effectiveness in mitigating overfitting and improving the generalization capability of deep learning models.

In Section 3.3, we describe the experimental setup used to evaluate the effectiveness of *EnSyth* on three different datasets: CIFAR-10, CIFAR-5, and MNIST-FASHION. We present comprehensive results and analysis, comparing the performance of *EnSyth* with other state-of-the-art deep learning models. Specifically, we demonstrate that our proposed approach achieves better accuracy and generalisation performance than baseline models, while requiring fewer parameters.

In the subsequent section, we critically discuss the results obtained from the application of *EnSyth*, highlighting the key strengths and limitations of our approach.

In particular, we discuss the trade-offs between model complexity and performance, as well as the challenges associated with the selection of optimal ensemble size and diversity.

Finally, in Section 3.5, we present the chapter's conclusions and summarise the main contributions of our work. We emphasise that *EnSyth* represents a novel and effective approach to improve the performance and generalisation capability of deep learning models, particularly in the context of limited training data. We also identify potential future directions for research, including the exploration of ensemble learning in more complex and diverse datasets, as well as the development of efficient algorithms for selecting optimal ensemble models.

3.2 Method

In this section, we provide a comprehensive overview of the approach taken to generate compressed deep learning models. Firstly, we introduce the topology of the feed-forward neural network models used in our study. Next, we delve into the pruning method [Aghasi, Abdi, and Romberg, 2018] used to generate compressed models with smaller network sizes and improved efficiency. We further explain how this method retains the most important network connections and achieves better performance compared to traditional compression techniques.

Furthermore, we elaborate on our approach to synthesise the compressed models from a diverse range of pruned models generated. We detail how the synthesis approach creates an ensemble of models with greater diversity than the baseline model, leading to improved performance. Finally, we describe the selection mechanism used to filter the best-compressed models from the ensemble based on performance metrics.

3.2.1 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) have emerged as a powerful tool in the field of deep learning, particularly for image and video processing tasks. CNNs are specifically designed to automatically learn and extract features from raw input data, making them well-suited for tasks such as image classification, object detection, image segmentation, and more [Lecun et al., 1998]. The architecture of CNNs consists of multiple layers, including convolutional layers, pooling layers, and fully connected layers. In convolutional layers, local receptive fields are convolved with the input data, allowing the network to capture spatial hierarchies and patterns [Krizhevsky, Sutskever, and Hinton, 2012]. The output of a convolutional layer is obtained through the following mathematical formula:

$$y(i,j) = \sigma\left(\sum_m \sum_n x(i+m, j+n) \cdot w(m,n) + b\right)$$

$x(i,j)$: input data

$w(m,n)$: weight of the convolutional filter at position (m,n)

$y(i,j)$: resulting feature map

b : bias term

$\sigma(\cdot)$: activation function

Pooling layers are then used to downsample the feature maps, reducing the spatial dimensions and providing translational invariance (Zeiler and Fergus, 2014) [3].

A commonly used pooling operation is max pooling, which selects the maximum value within each pooling region. This can be represented mathematically as:

$$y(i,j) = \max\left(\{x(m,n)\}\right)$$
$$x(m,n) : \text{input values within the pooling region}$$

The output from these layers is flattened and linked to generate fully connected layers, also known as dense layers, which perform high-level feature extraction and produce the final predictions. In a fully connected layer, each neuron is connected to every neuron in the previous layer. The mathematical formula for the fully connected layer can be expressed as:

$$y = \sigma\left(\mathbf{W} \cdot \mathbf{x} + \mathbf{b}\right)$$
$$\mathbf{x} : \text{input vector}$$
$$\mathbf{W} : \text{weight matrix}$$
$$\mathbf{b} : \text{bias vector}$$
$$\sigma(\cdot) : \text{activation function}$$

The comprehensive architecture of a CNN is illustrated in Figure 3.1, showcasing the intricate framework that encompasses various layers and their interconnections.

Convolutional Neural Networks (CNNs) can be viewed as a specific type of feed-forward neural network architecture. In a traditional feed-forward neural network, each neuron in a layer is connected to all neurons in the previous layer. Similarly, in a CNN, the fully connected layers at the end of the network operate in the same way as in a standard feed-forward neural network, where each neuron is connected to every neuron in the previous layer. This allows the CNN to perform high-level feature extraction and make predictions based on the learned features. The convolutional layers, on the other hand, utilize local receptive fields and shared weights, enabling them to capture spatial hierarchies and patterns in the input data. This characteristic makes CNNs particularly well-suited for image and video processing tasks [Lecun et al., 1998]. In the upcoming section, we delve into the discussion of feed-forward neural networks, with particular attention to the Convolutional Neural Network (CNN) as a specialized variant within this category. This exploration is crucial as it enables us to present the essential mathematical formalisation required for the pruning method employed in our study.

It is important to note that all forthcoming experiments in this chapter, as well as the remaining sections of the thesis, will exclusively focus on the utilization of Convolutional Neural Networks (CNNs).

3.2.2 Feed-forward Neural Networks

Feed-forward neural networks are a type of artificial neural networks that are commonly used in supervised learning applications, where the aim is to learn a function that maps input data to output labels. Assuming that the network is trained using x_p training example where:

1. $p = 1, \cdots, P$;
2. $x_p \in \mathbb{R}^N$: $\mathbb{R}^N$;

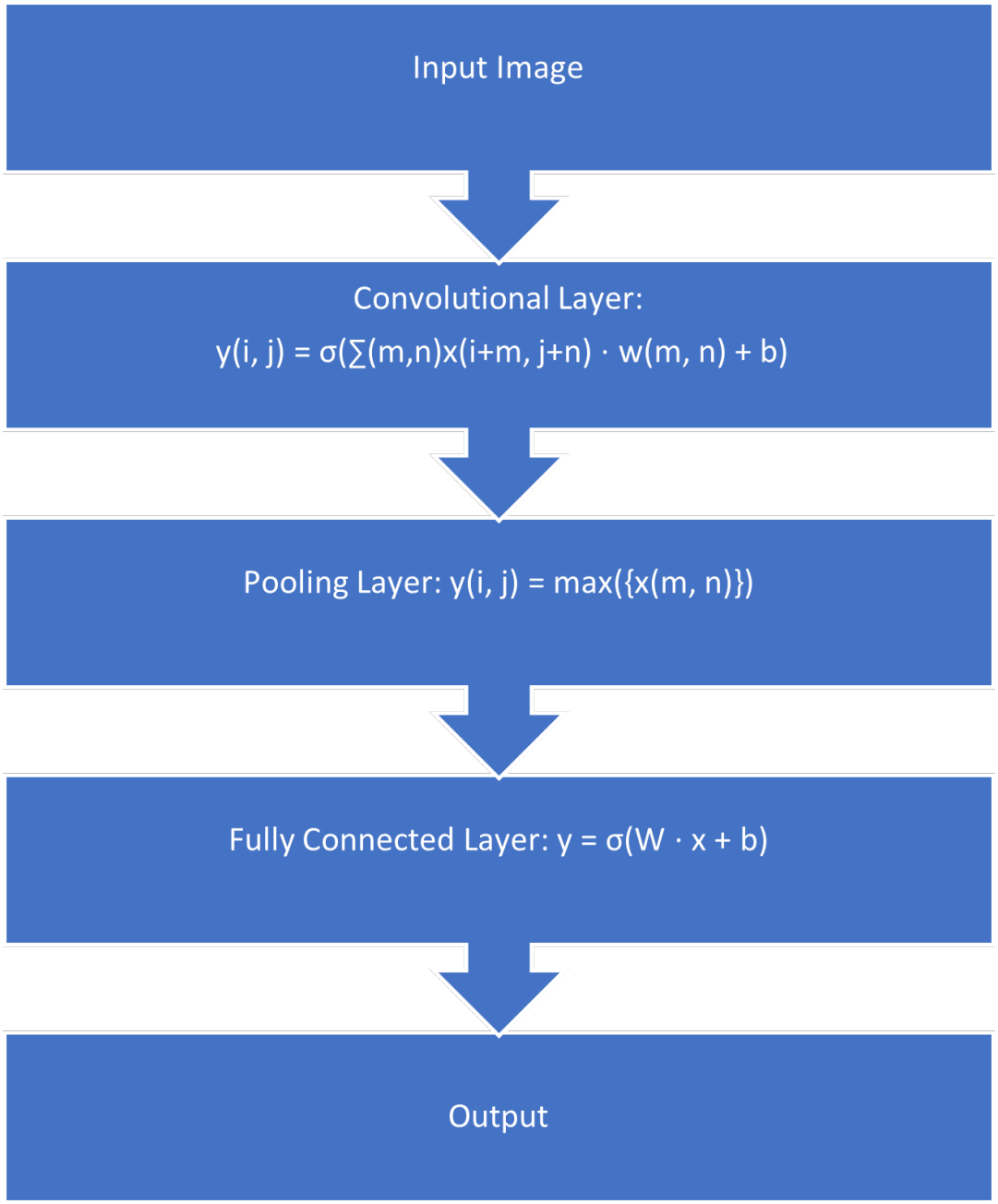

FIGURE 3.1: Visual representation of the layers in a Convolutional Neural Network (CNN). The input image is processed through convolutional layers, followed by pooling layers for spatial down-sampling. The output is then fed into fully connected layers for high-level feature extraction and prediction. Each layer performs specific operations, such as convolution and pooling, which enable the network to capture spatial hierarchies and extract meaningful features from the input data..

Suppose $X \in \mathbb{R}^{N \times P}$ is a one-dimensional matrix representing the training samples as $X = [x_1, \cdots, x_P]$, L is a layer in the network; the network's output at the last layer is represented by $X^{(L)} \in \mathbb{R}^{N_L \times P}$ where each column in $X^{(L)}$ is a response to the corresponding training column in X. In ReLU neural network, the output of ℓ^{th} layer is defined as :

$$X^{(\ell)} = ReLU\Big(W_\ell^T X^{(\ell-1)} + b^{(\ell)} 1^T\Big) \tag{3.1}$$

where $\ell = 1, \cdots, L$..

If we add an additional row to both of $X^{(\ell-1)}$ and W_ℓ the previous formula could be written as:

$$X^{(\ell)} = ReLU\Big(W_\ell^T X^{(\ell-1)}\Big) \tag{3.2}$$

where $\ell = 1, \cdots, L$.. A neural network which follows one of the two previous formulas(3.1,3.2) will be an ideal candidate for the pruning method, which will be described next.

3.2.3 Net-Trim- The Pruning Method

Neural networks, especially deep ones, are composed of multiple layers. Each layer is like a processing block, with a combination of linear and nonlinear operations. These networks often have a vast number of parameters, sometimes in the millions. This large number of parameters can lead to challenges:

- Storage: Storing such large models can be problematic.
- Interpretation: Understanding the role of each parameter can be complex.
- Manipulation: Adjusting or optimising the model can be cumbersome.

To tackle these challenges, a method called Net-Trim was introduced [Aghasi et al., 2017]. Net-Trim is a post-training method applied to already trained neural networks. Its main goal is to simplify the network by reducing the number of parameters without significantly affecting the network's performance. It does this by "pruning" or removing unnecessary connections in each layer. **How Does Net-Trim Work?**

- Layer-by-Layer Pruning: Net-Trim goes through each layer of the network and decides which connections (or parameters) are essential and which can be removed.
- Convex Optimisation and LASSO Comparison: The decision on which connections to prune is made using a mathematical approach called convex optimisation. This ensures that a large number of parameters in the network are set to zero, effectively removing them. If you're familiar with the LASSO method used in linear models [Girosi, Jones, and Poggio, 1995], Net-Trim operates on a similar principle. Essentially, Net-Trim can be thought of as applying the LASSO concept to neural networks.

The process of NetTrim is summarised in Figure 3.2.

Several benefits could be obtained using Net-Trim:

- Faster Predictions: With fewer parameters, the network can make predictions more quickly.
- Memory and Storage Efficiency: A pruned network requires less memory and storage.

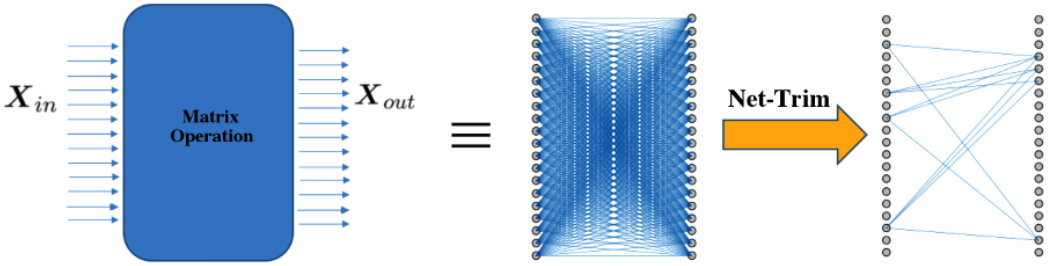

FIGURE 3.2: Illustration of how Net-Trim prunes unnecessary connections, making the network leaner and more efficient.

- Potential for Higher Accuracy: By removing redundancies, Net-Trim can sometimes improve the network's test accuracy.
- Better Interpretability: A simpler network is easier to understand, and with Net-Trim, we can better identify which features are most important for the network's decisions.

Technical Details:
In a typical ReLU neural network, the output of each layer is given by:

$$X_{out} = ReLU\Big(WX_{in}\Big) \tag{3.3}$$

Where: X_{out} is the output matrix. W represents the weight matrix of the network. X_{in} is the input matrix, with each column corresponding to a training sample.

Net-Trim introduces modifications to this structure, optimising the weight matrix and potentially replacing it with a more efficient version. Furthermore, Net-Trim has hyperparameters that can be adjusted. By tweaking these, we can obtain different models with varying accuracy, size, and inference time. This flexibility is crucial for our approach, where we aim to find an optimal balance for deep learning model ensembles.

3.2.4 Synthesis of Deep Learning Ensembles

Synthesising sets of diverse compressed models into ensemble predictions is a critical element in our approach because ensemble learning will not only allow the creation of better classifiers but also overcome potential overfitting issues and provide more generalisation to the final solution. Let m a pruned model generated by Net-Trim and the decision of the model m_i about a class w_j be defined as: $y_{i,j} \in \{0,1\}$ where:

- $i = 1, 2, \cdots, N$: N is the number of the classifiers.
- $j = 1, 2, \cdots, C$: C is the number of classes.

if m_i predicted correctly a class w_j then $y_{i,j} = 1$ otherwise $y_{i,j} = 0$. I use plurality voting [Zhou, 2012] as a simple ensemble learning technique to synthesise the classifiers. The prediction of each compressed model is considered as a vote, and then the predictions which get the majority of votes will be found in the final ensemble's prediction. Let $P = m_1, \cdots, m_N$ an ensemble of pruned models m_i, the final ensemble decision about a test class in plurality voting is a class w_j that receives the maximum support $\eta_{final}(P)$ from all the classifiers which form the ensemble. Thus

the output of the ensemble could be defined as:

$$\eta_{final}(P) = argmax_{j\in\{1,\cdots,C\}} \sum_{i=1}^{N} y_{i,j}$$

Figure 3.3 provides a brief overview of the concepts being illustrated.

Although ensembling multiple models is a powerful technique that can significantly boost the performance of a deep learning model, it is important to note that not all models in the ensemble contribute equally to the final predictions. Some models may have lower predictability levels, which can negatively impact the overall performance. To ensure that we only consider models that are truly beneficial to the ensemble, we propose a backward elimination scheme.

The backward elimination scheme involves iteratively removing models from the ensemble that have the lowest predictability levels until we arrive at an optimal combination of models that achieve the best results. By eliminating models with reduced predictability levels, we can improve the overall performance of the ensemble and ensure that only the most effective models are included. In the next section, we present our proposed back-elimination technique.

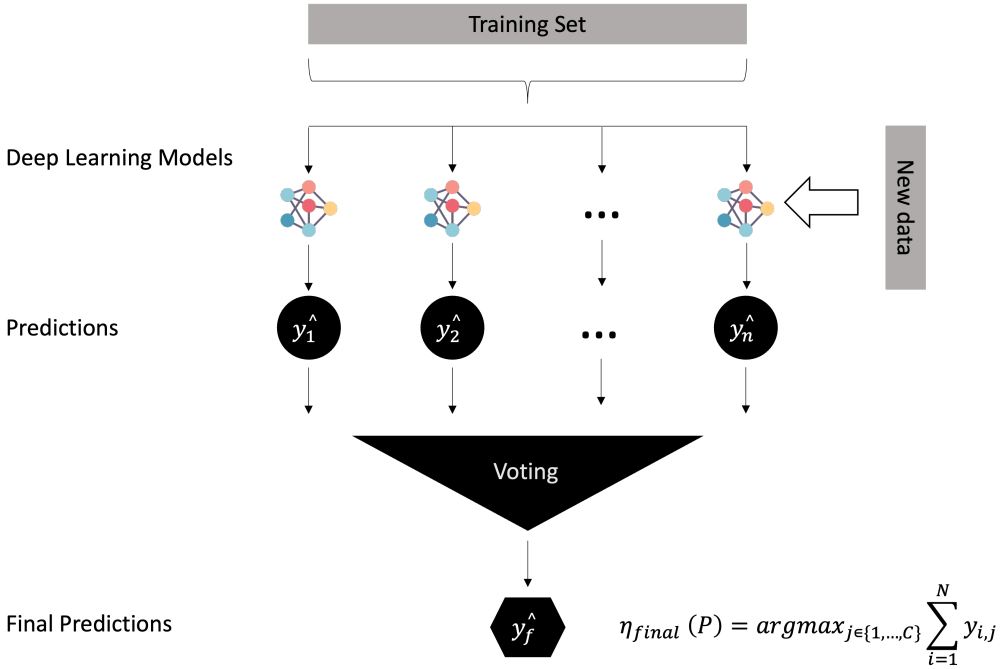

FIGURE 3.3: Synthesis of deep learning ensembles.

3.2.5 Backward Elimination

The primary aim in constructing ensembles for deployment on resource-constrained devices is to maintain high accuracy while minimizing the ensemble's size to conserve computational resources. This balance is critical, especially for real-time applications where inference speed is paramount.

Our backward elimination strategy is akin to feature selection techniques in machine learning, where non-contributing features are pruned to improve model performance. We initiate this process with a comprehensive set of pruned models, each contributing to the ensemble's predictive power. The backward elimination process iteratively assesses the ensemble's performance, removing the least accurate model in each cycle. This iterative reduction continues until we are left with the smallest ensemble that still meets our accuracy threshold.

The term "lowest accuracy" refers to the model within the ensemble that contributes the least to correct predictions in the current iteration. This is a dynamic measure, as the contribution of each model can change depending on the composition of the ensemble. The "high predictability levels" we seek are not just high accuracy in isolation but the ability of the ensemble to generalize well to new data, which is a hallmark of robust models.

The backward elimination process is designed to identify the core set of models that are most effective. The absolute minimum number of models in the ensemble is not a fixed number but rather the smallest subset that can achieve accuracy comparable to the full ensemble. This is contingent upon the diversity and complementarity of the models within the ensemble. In one scenario, a few highly diverse models may suffice, while in another, more models may be required to capture the complexity of the data.

Algorithm 6 delineates the backward elimination procedure. The algorithm's objective is to distill the ensemble to its essence, retaining only those models that are imperative for maintaining high accuracy. This process is not merely about discarding models but about understanding the synergy between them and preserving the collective intelligence of the ensemble.

Algorithm 6 Backward Elimination for Optimal Ensemble Selection

$N = Size(Ensemble)$
while $N > 0$ **do**
  $predict \leftarrow \eta_{final}(P)$
  Evaluate the contribution of each model to the ensemble's accuracy;
  Identify and remove the model m with the lowest incremental accuracy gain;
  $N = N - 1$;
  Reassess the ensemble's performance without model m;
end while
end while
Return the optimally reduced ensemble that satisfies the predefined accuracy criteria;

In this process, we consider not only the model with the lowest accuracy but also its incremental value to the ensemble's performance. This nuanced approach ensures that each model's removal is justified not only by its standalone performance but also by its contribution to the collective outcome. The potential scenarios for the minimum ensemble size vary, ranging from a single, highly predictive model to a combination of models that together provide a comprehensive understanding of the data space. The backward elimination algorithm is thus a tool for achieving the most efficient ensemble, tailored to the specific needs and constraints of the deployment environment.

3.3 Experiment

In this section, we aim to provide a detailed and comprehensive explanation of the experiment conducted to evaluate the performance of our proposed method. To begin with, let us describe the benchmarking datasets and the baseline model used in our simulation. Next, we explain the pruning method used in our experiment. We adopt a systematic pruning approach where we eliminate connections with the least absolute weight magnitude iteratively. We prune the LeNet-5 model with different values of hyperparameters. Finally, we present the results of our experiment. We evaluate the accuracy of our method on the CIFAR10 and MNIST-FASHION datasets and compare it with the baseline model.

3.3.1 LeNet-5

LeNet-5 [Lecun et al., 1998] is a widely-used convolutional neural network (CNN) model known for its success in image classification tasks. The architecture consists of two convolutional layers, each with a filter size of (3255) and (6455), respectively. Following each convolutional layer is an average pooling layer with a filter size of (2*2) and a stride of two. The network ends with two fully connected layers. The overall architecture of the network can be represented as $INPUT => CONV => POOL => CONV => POOL => FC => FC$, where CONV represents the convolutional layer, POOL represents the average pooling layer, and FC represents the fully connected layer. This architecture has been proven to be effective in image recognition tasks, making it an ideal baseline model for our experiments.

3.3.2 Datasets

CIFAR-10

The CIFAR-10 [Krizhevsky, n.d.] dataset is widely used for benchmarking image classification models. It consists of 70,000 32×32 colour images that are categorised into ten distinct classes, each containing 6,000 images. The dataset is divided into two sets: a training set of 50,000 images and a testing set of 10,000 images. Each image is labelled with one of ten categories, which include airplanes, automobiles, birds, cats, deer, dogs, frogs, houses, ships, and trucks. The dataset is challenging due to its relatively low resolution and the small size of the images, making it difficult for models to accurately distinguish between similar object categories. However, its popularity stems from its ability to provide a standardised benchmark for image classification algorithms, allowing researchers to compare the performance of different models on the same dataset.

CIFAR-5

To diversify our benchmarking dataset, we introduced a subset of CIFAR-10 consisting of images from five animal categories, which we refer to as CIFAR-5. This subset contains 30,000 images in total, with 25,000 images used for training and 5,000 for testing. Each example in the dataset is assigned to one of the following labels: [2: bird, 3: cat, 4: deer, 5: dog, 6: frog]. It's worth noting that our goal in introducing this subset is not to reduce the complexity of the dataset, but rather to provide an alternative benchmark for evaluating the performance of our method.

MNIST-FASHION

MNIST-FASHION [Xiao, Rasul, and Vollgraf, 2017] is another benchmarking dataset that comprises 70,000 grayscale images of fashion products categorized into ten classes, with 7,000 images per class. The training set consists of 60,000 images while the testing set has 10,000 images. The images in the dataset have a resolution of 28×28 pixels. Each example in the training and testing sets is labeled as one of the ten classes: [0: T-shirt/top, 1: Trouser, 2: Pullover, 3: Dress, 4: Coat, 5: Sandal, 6: Shirt, 7: Sneaker, 8: Bag, 9: Ankle boot]. This dataset is an alternative to the traditional MNIST dataset, which has been widely used in the machine learning community for benchmarking image classification models. MNIST-FASHION presents a more challenging task due to the complex visual features of the fashion product images, making it a suitable dataset for evaluating the performance of image classification models.

In this following section, we provide a rationale for the selection of the LeNet-5 model and the CIFAR-10 and CIFAR-5 datasets for our deep learning model synthesis.

The choice of the LeNet-5 model and the CIFAR-10 and CIFAR-5 datasets for our deep learning model synthesis was carefully considered based on several key factors. Firstly, the selection of the LeNet-5 architecture is rooted in its historical significance as a pioneering convolutional neural network (CNN) design, which has laid the foundation for modern CNNs. The work in [Lecun et al., 1998] introduced the LeNet architecture, marking a crucial milestone in the evolution of deep learning for image analysis. Its relatively shallow architecture, which comprises a series of convolutional and pooling layers followed by fully connected layers, is well-suited for tasks involving image classification. This choice not only pays homage to the fundamental work in the field but also serves as a benchmark to assess the effectiveness of our proposed synthesis approach on a widely recognised architecture. Secondly, the CIFAR-10, Mnist Fashion dataset was chosen due to its diverse set of 10 object classes, making it a standard benchmark for image classification tasks. Thier small image resolution and complex object recognition challenges make it ideal for evaluating the generalisation capabilities of deep learning models. In addition to CIFAR-10, we introduced the CIFAR-5 dataset, a novel dataset that includes a subset of five classes from CIFAR-10. The CIFAR-5 dataset was designed to explore model generalisation across diverse image classification tasks. Our choice of these datasets and the LeNet-5 model is rooted in a well-balanced consideration of historical significance, benchmarking opportunities, and the desire to explore the generalisation and versatility of deep learning models. By evaluating our synthesis approach on these datasets and architectures, we aim to showcase its adaptability and robustness in tackling diverse image classification challenges, thereby contributing to the broader field of deep learning research.

3.3.3 Experimental Setup

The experiment was conducted on a system with the following specifications: Ubuntu Desktop 18.04.1 LTS operating system, Intel KVM 64-bit 2.4 GHz CPU, 16384 KB cache, and 32 GB RAM. The software used for the experiment includes Python 3.6.7, TensorFlow 1.10.0, and Keras 2.2.4.

3.3.4 Network Training and Pruning

To establish baseline models, we trained the LeNet-5 model on the proposed data sets. The LeNet-5 baseline model achieved an accuracy of 78.4% on CIFAR-10, 73.3% on CIFAR-5, and 90.3% on MNIST-FASHION. We then employed NetTrim [Aghasi, Abdi, and Romberg, 2018] to prune and fine-tune the baseline models. NetTrim offers four hyperparameters: L1, which applies L1 regularisation on the weights of the models; L2, which applies L2 regularisation on the weights of the model; dropout, a factor used to randomly ignore neurons during training; and Epsilon-gain, which directly affects the accuracy and sparsity of the pruned model.

We generated 36 compressed models by varying the values of these parameters. Table 3.1 presents a detailed overview of the combinations used to create a pool of compressed models.

The above table represents the solution space for our method, but the real solution space may contain up to $2^{36} - 1$ models where different combinations and permutations could be obtained. The following is a summary of statistics about the size of the 36 pruned models where only the weights and biases are saved as compressed Numpy arrays [Harris et al., 2020]:

1. CIFAR-10: Max:4.2 MB, Min:1.6 MB, Avg: 2.94 MB (baseline: 4.8 MB);
2. CIFAR-5: Max: 4.2 MB, Min: 1.3 MB, Avg: 2.89 MB (baseline: 4.8 MB);
3. MNIST-FASHION: Max:7.7 MB, Min: 0.88 MB, Avg: 2.56 MB (baseline: 7.7MB);

As seen above, the model sizes have been significantly reduced, resulting in a remarkable decrease in memory footprint. In the case of CIFAR-10, the average model size was reduced by approximately 39% compared to the baseline, enabling more efficient utilization of IoT resources. Similarly, for CIFAR-5, the average model size was reduced by around 40%. For MNIST-FASHION, the reduction was even more substantial, with the average model size shrinking by approximately 67% compared to the original size. This significant compression not only minimises the memory requirements but also enhances the overall utilisation of IoT resources, making the models more lightweight and feasible for deployment in resource-constrained environments.

3.3.5 Synthesis of Compressed Deep Learning Ensembles

The models' weights in the current solution space are combined to form new classifiers using majority voting. Next, the classifier's predictability is evaluated against the testing set. Then a simple backward elimination is applied to exclude the weakest model (i.e. the model with the lowest accuracy on the testing set). Ideally, the elimination process should be performed on the validation set rather than the testing set. However, as the aim of this work is to prove the existence of pruned ensembles that could be synthesised from the same baseline model, the testing set was selected.

3.4 Results and Discussion

3.4.1 Results of CIFAR-10

The Figure 3.4, provides a visual representation of the performance comparison between the proposed method and the baseline model for the ensembles. The results

TABLE 3.1: Ensyth - hyperparameters values

Solution Space	**Hyperparameters**			
	ϵ	L1	L2	dropout
Set1	0.01 0.02 0.04 0.06 0.08 0.1 0.2 0.3 0.4 0.5 0.6 0.7	0	1	1
Set2	0.01 0.02 0.04 0.06 0.08 0.1 0.2 0.3 0.4 0.5 0.6 0.7	0	[0,0,0.004,0.004,0]	1
Set3	0.01 0.02 0.04 0.06 0.08 0.1 0.2 0.3 0.4 0.5 0.6 0.7	0	[0,0,0.004,0.004,0.004]	0.5

are depicted in a clear and concise manner, highlighting the superiority of the proposed method in terms of predictability.

One of the key observations that can be made from the results is that the predictability of the ensemble models improves as the number of models in the ensemble increases. But this escalation in accuracy usually stops at a point. In particular, the results show a noticeable trend of enhanced predictability when the number of models in a prediction ensemble ranges between 6 and 30. This suggests that an ensemble with a moderate number of models can offer a better trade-off between prediction accuracy and computational cost.

Additionally, the figure reveals that a significant proportion of the ensemble models outperform the baseline model. Upon closer inspection of the data, it becomes apparent that 23 out of 36 models in the ensembles surpass the performance of the baseline model. This is a promising result, as it indicates that the proposed method can consistently improve the accuracy of the ensemble models.

Notably, the figure highlights that the highest accuracy achieved was 78.86% from an ensemble comprising eight models. This result is particularly noteworthy, as it demonstrates the potential of the proposed method to boost the accuracy of prediction ensembles significantly. The empirical results from our experiments with CIFAR-10 reveal a nuanced relationship between ensemble size and performance, characterized by a point of inflection where the benefits of adding more models begin to taper off. This trend is indicative of the principle of diminishing returns in ensemble learning, where each additional model contributes less to predictive accuracy relative to the computational cost incurred [Oza and Tumer, 2008]. Achieving the optimal balance between prediction accuracy and computational expense is thus realised with a moderate-sized ensemble. Such a balance is crucial not only for efficiency but also for maintaining a diversity that captures various data aspects without introducing redundancy that could lead to computational waste.

This concept of balance segues into the critical role of diversity among ensemble members, which significantly influences ensemble methods' performance. The literature on ensemble learning robustly supports the notion that a collection of diverse models is more effective in covering the solution space than a uniform group, thereby enhancing the generalisation capabilities on unseen data [Kuncheva et al., 2003]. Diversity is key because it ensures that the errors of individual models are less likely to be correlated; hence, their aggregation tends to cancel out the errors, leading to an improvement in overall accuracy. However, the addition of models to an ensemble does not guarantee an increase in diversity. As the ensemble grows, the probability of integrating models that genuinely contribute to diversity decreases, particularly if the pool of available models lacks variety. The number of models is less significant than the uniqueness of the perspectives each model brings to the ensemble. Our findings demonstrate that ensembles comprising 6 to 30 models strike a practical balance, offering robust performance while avoiding the computational extravagance of larger ensembles. This 'sweet spot' effectively leverages the advantages of diversity without succumbing to the diminishing returns of oversized ensembles, thus substantiating the premise that a moderate ensemble size can indeed present the most advantageous trade-off between prediction accuracy and computational cost [Dietterich, 2000].

3.4.2 Results of CIFAR-5

Upon examining the data presented in Figure 3.5, it becomes evident that the proposed *EnSyth* method has achieved slightly better accuracy than the baseline model

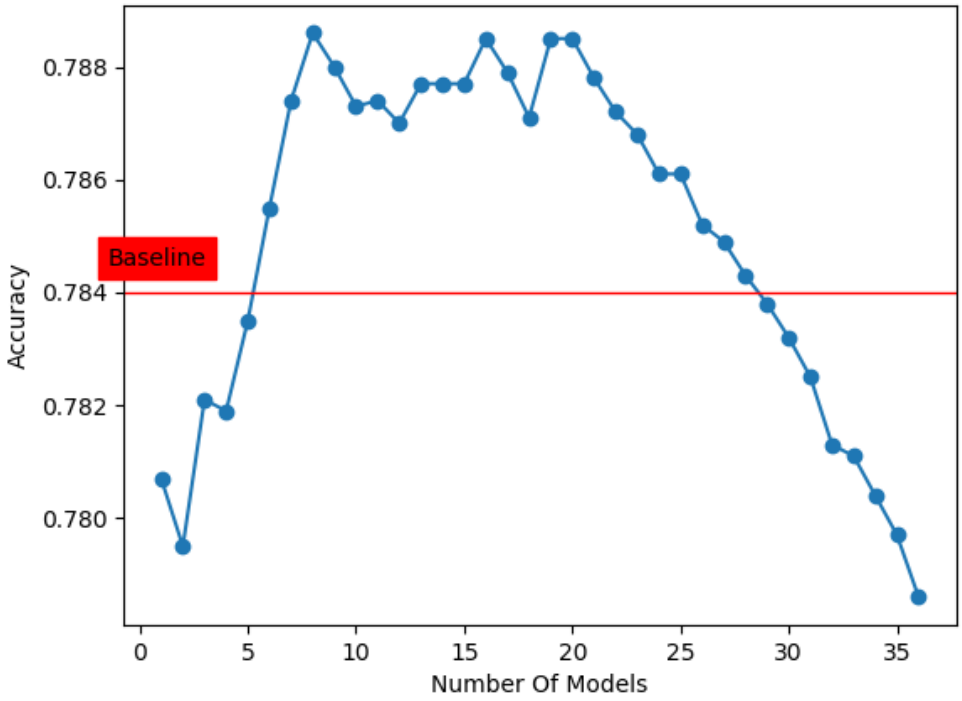

FIGURE 3.4: CIFAR10

by leveraging the power of synthesising deep learning ensembles. Notably, the results showcase a total of 12 ensembles that outperform the baseline model, out of the vast solution space of $2^{36} - 1$ potential solutions that were explored.

It is worth highlighting that the highest accuracy achieved through the proposed method was 73.82%, which was attained by combining three models in an ensemble. This result demonstrates the potential of the *EnSyth* method to enhance the accuracy of prediction ensembles significantly.

In summary, the results presented in Figure 3.5 emphasise the effectiveness of the *EnSyth* method in boosting the accuracy of deep learning ensembles. By synthesising the ensembles, the method has achieved superior performance over the baseline model, with a considerable number of outperforming solutions found in the explored solution space.

3.4.3 Results of MNIST-FASHION

The results presented in Figure 3.6 shed light on the performance of the proposed *EnSyth* method on the MNIST-FASHION testing set. Unlike the findings on the CIFAR-10 dataset illustrated in Figure 3.4, none of the ensembles composed with the *EnSyth* method was able to surpass the baseline model's accuracy on this particular dataset.

This outcome can be attributed to the fact that the baseline model already achieves a high level of accuracy on the MNIST-FASHION testing set, with a performance of 90.3%. Therefore, the proposed combinations of models failed to achieve superior results on this dataset.

However, it is worth noting that some of the ensembles that consisted of a small number of models (1-5) were able to achieve comparable accuracy to the baseline model (90.21%). This observation is a clear indication that the search space of $2^{36} - 1$

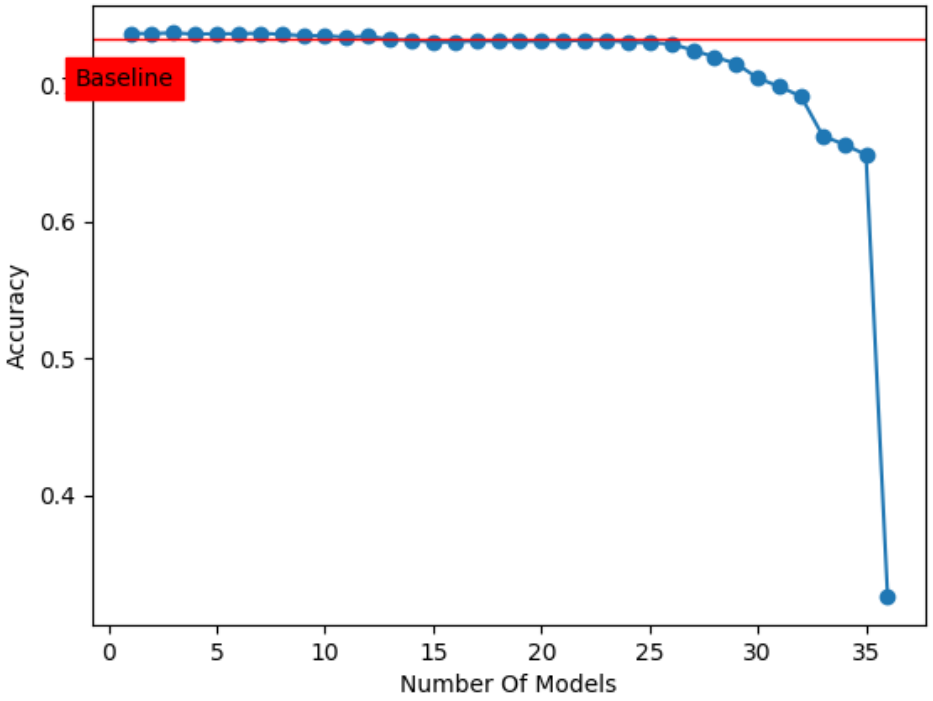

FIGURE 3.5: Ensyth results using CIFAR5

potential solutions is likely to contain a set of ensembles that can produce promising results.

In summary, Figure 3.6 provides insights into the performance of the *EnSyth* method on the MNIST-FASHION testing set. While none of the ensembles were able to outperform the baseline model, the results suggest that the search space of potential solutions is vast and contains a set of ensembles that can produce promising results. Ultimately, the results underscore the importance of exploring the solution space to identify the best-performing ensembles.

3.4.4 Computational Cost

Further analysis for *EnSyth* shows that the ensembles generated by the method require fewer mathematical operations to make a prediction, as evidenced by the trainable parameters count. It is assumed that the number of trainable parameters for an ensemble is equal to the model that has the maximum number of parameters in that ensemble. In addition, the ensembles also exhibit faster CPU execution time, making them more efficient compared to the baseline model.

To provide further insight into this finding, Table 3.2 presents the number of trainable parameters and the average CPU execution time for the solution space explored during the experiments. The CPU execution time was calculated by feeding each model generated from a set of multiple solution spaces with 50 randomly selected images from CIFAR10, MNIST-FASHION, and 25 images from CIFAR5. The experiments were repeated nine times to obtain reliable measurements, and the averages were calculated.

The results displayed in Table 3.2 demonstrate that the ensembles generated by the *EnSyth* method exhibits a lower number of trainable parameters and faster CPU

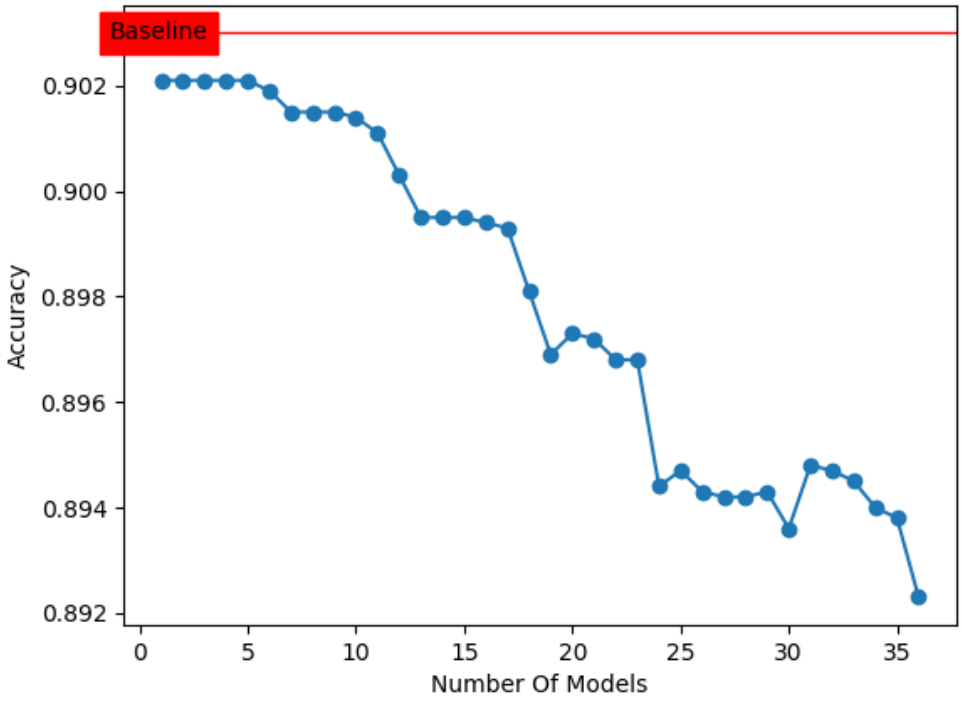

FIGURE 3.6: Ensyth results using MNIST Fashion

execution time compared to the baseline model. This finding highlights the potential of the proposed method to not only enhance the accuracy of prediction ensembles but also to make them more efficient in terms of computational resources.

The stacked bar charts represented in Figure 3.7 serve as a visual representation of the numerical data detailed in the LaTeX table provided earlier. It illustrates the processing times (CPU) and the number of parameters (PARAMS) for various models across the CIFAR10, CIFAR5, and MNIST-FASHION datasets. For each dataset, the models show a varied number of parameters and processing times. In general, as the model numbers increase, there is a trend of decreasing parameters, which is consistent across all datasets. CPU times also tend to decrease but not as consistently, indicating potential optimisations in model complexity without a proportional increase in processing time. The Baseline model, present in all datasets, has the highest parameters and CPU times, serving as a reference point from which subsequent models deviate. These visualisations effectively highlight the trade-offs and balances between model complexity (as indicated by PARAMS) and efficiency (as reflected by CPU times).

3.4.5 Backward elimination

The "lowest accuracy" within the backward elimination process is determined by a model's relative contribution to the ensemble's predictive capability. This is not solely a measure of individual model accuracy but also considers the model's redundancy and the uniqueness of its contribution to the ensemble's diversity. A model is identified for elimination when it offers the least incremental benefit in terms of ensemble accuracy or when its removal does not significantly diminish the ensemble's performance.

TABLE 3.2: Processing Time in μs

Solution Space	**CIFAR10**		**CIFAR5**		**MNIST-FASHION**	
	PARAMS	CPU	PARAMS	CPU	PARAMS	CPU
Baseline	1663367	91389	1663367	65342	1068298	60462
model1	449125	88712	495739	55458	690727	64838
model2	434727	97337	462392	57717	515694	63037
model3	409820	92000	413032	53179	349340	54986
model4	389077	104112	377410	63897	261155	63275
model5	372406	104244	350174	54509	216367	55309
model6	357104	95805	327820	58771	186753	57215
model7	300644	89662	262180	57328	124971	71198
model8	264190	87792	230782	59406	103398	68252
model9	238728	90467	210362	50445	89496	62281
model10	220266	98415	191688	63527	81341	68114
model11	202498	96473	173373	58674	73196	71749
model12	183802	87632	156328	58764	67678	60940
model13	449591	84103	495625	49144	690725	61207
model14	435068	87470	462176	61775	515985	71292
model15	409863	81002	412918	62938	349510	57813
model16	389414	87186	377683	53196	263364	66171
model17	372045	82882	350011	54519	216602	65960
model18	357415	89725	328173	57192	186321	63897
model19	125306	87472	262669	63158	300819	65514
model20	264433	91569	230848	59841	103440	65623
model21	238866	83084	210599	65157	89636	80378
model22	220109	85930	191402	62459	81384	72368
model23	203311	94199	173371	64266	73173	71940
model24	182823	82880	155159	47629	67589	69643
model25	449578	89749	495692	65576	690728	64918
model26	435326	84622	462402	57962	516312	70715
model27	410650	93787	413068	65927	349515	88424
model28	389634	92072	377807	56507	262603	69605
model29	372273	92882	350256	61985	216909	64090
model30	357602	87173	327886	57042	186680	78473
model31	300925	88574	262159	60170	125250	61816
model32	264239	91445	230685	58797	103346	67042
model33	239254	90179	210282	58426	91338	67711
model34	219892	89763	190855	52055	81142	68332
model35	202639	95498	173020	63604	73265	72857
model36	184180	94243	157021	56694	67622	67416

"High predictability levels" refer to the ensemble's ability to maintain high accuracy consistently across various datasets and scenarios. This involves evaluating the ensemble's performance stability under different conditions, such as class imbalances, noise, and data perturbations. Predictability is thus a multifaceted measure that encompasses not only accuracy but also the ensemble's robustness and reliability.

In our results, we observed that ensembles with a moderate number of models, specifically between 6 and 30, provided the most significant gains in predictability without incurring excessive computational costs. This finding aligns with the backward elimination process, which seeks to identify the smallest ensemble that maintains high predictability levels. The absolute minimum number of models in the ensemble is thus determined by the point at which further reductions would lead to a notable decrease in accuracy or an increase in prediction variance.

The efficacy of the backward elimination algorithm is further substantiated by the experimental results. For instance, in the CIFAR-10 dataset, the highest accuracy was achieved with an ensemble of eight models, indicating that beyond this point, additional models contribute less to predictive performance. This supports the use of backward elimination to systematically identify and remove models that are less contributory, ensuring an optimal ensemble size for deployment, especially in resource-constrained environments such as edge devices or real-time applications.

In light of these results, the backward elimination process is validated as an effective method for synthesizing compact and highly predictive ensembles, as demonstrated by the performance improvements over the baseline model in the CIFAR-10 and CIFAR-5 datasets. While the MNIST-FASHION dataset did not see ensembles surpassing the baseline, the process still identified ensembles with comparable accuracy, emphasising the potential of the explored solution space and the backward elimination method's role in navigating it efficiently.

3.4.6 Discussion

As shown in Figure 3.5, *EnSyth* was successful in improving the accuracy of the compressed classifier by synthesizing deep learning ensembles. Specifically, our approach achieved marginally better accuracy compared to the baseline model. Out of the 36 solutions explored in the solution space (out of a possible $2^{36} - 1$ solutions), we identified 12 ensembles that outperformed the baseline model. The highest accuracy of 73.82% was achieved by combining three models in the ensemble. Additionally, the results of the CIFAR-10 testing presented in Figure 3.4 demonstrate superior predictability for the ensembles compared to the baseline model. A noticeable trend is observed here, indicating that the predictability of the ensembles increases when the number of models in the ensemble is between 6 to 30. A detailed analysis of this figure reveals that 23 out of the 36 models outperform the baseline model. Notably, the highest accuracy achieved by the ensemble consisting of eight models is 78.86%. Moving to the results obtained on the MNIST-FASHION testing set are presented in Figure 3.6, it is worth noting that some ensembles with a small number of models [1–5] achieved similar accuracy to the baseline model (90.21%). This indicates that the real solution space of $2^{36} - 1$ is likely to contain a set of ensembles that can produce promising results. Further examination for Table 3.2 shows that the ensembles require fewer mathematical operations for prediction compared to individual models, based on the number of trainable parameters. Additionally, ensembles were found to have lower CPU execution time compared to individual models.

Besides our aim to generate better-compressed classifiers, we are particularly interested in exploring the space of the existing solutions in depth to identify the optimal combination of candidates that could be synthesised to produce better results. With only 36 models and a simple selection technique like backward elimination, *EnSyth* was able to search a small subset of the space (only 36 out of $2^{36} - 1$ possible solutions) to find one or more ensembles that outperform the baseline model. As such, there is a great opportunity for further investigation, this includes: (1) expanding the space of the pruned models by applying different pruning methods with different hyperparameters; and (2) dealing with the selection of the optimal ensemble as a multi-objective optimisation problem to find the minimum number of models (a small ensemble) while achieving the highest possible accuracy. Furthermore, the results from CIFAR10, CIFAR-5, and MNIST-FASHION indicate that with a small number of models in an ensemble, the classifier can achieve high predictability levels. This is particularly interesting when it comes to deploying those ensembles on smartphones and Internet of Things (IoT) devices because the ensemble size is small compared to the baseline model and the performance in terms of inference time and accuracy is even better.

3.5 Summary

In this chapter, we describe *EnSyth*, a method to synthesise deep learning ensembles that results in improved classifiers in terms of both inference time and accuracy. The approach involves applying multiple pruning methods with varying hyperparameters to generate a diverse pool of pruned models that can be used to form ensembles. The number of possible ensembles is $2^m - 1$, where m is the number of pruned models. The approach consists of several stages. First, we train a baseline model and then prune it using different hyperparameters to form a pool of compressed models. After that, we employ ensemble learning to compose new predictors and search the solution space for outperforming models.

The simulation results on CIFAR-10 and CIFAR-5 datasets reflect the effectiveness of the proposed method in two ways: 1) improving the performance of compressed models, and 2) providing better utilisation of edge resources by reducing the number of required math operations to make an inference. The results of experiments on CIFAR-10, CIFAR-5, and MNIST-FASHION datasets reveal that ensembles with a small number of models can achieve high predictability levels compared to the baseline models. This finding is significant for deploying these ensembles on smartphones and AIoT devices because they could work effectively in any resource-limited environment without causing performance issues.

In the upcoming chapter, we present an innovative approach to further reducing the memory usage of running deep learning ensembles generated by *EnSyth* on resource-limited devices. Our solution is MicroNets, a comprehensive framework that offers a complete pipeline for deploying deep learning ensembles on AIoT devices with limited resources. The pipeline comprises multiple stages, including a novel multi-phase pruning method that enables the generation of a diverse pool of compressed models with superior generalisation power. Furthermore, our pruning approach includes a novel technique for ensemble pruning, which reduces the memory required for inference using deep ensemble learning. The proposed approach not only reduces the memory footprint of ensembles but also improves accuracy and efficiency, making it suitable for deployment on resource-constrained AIoT devices.

MicroNets also includes additional features, such as efficient weight quantisation and selective activation pruning, to further reduce memory consumption and

computation costs. Our framework is specifically designed to enable the effective deployment of deep learning ensembles on edge devices while meeting resource constraints.

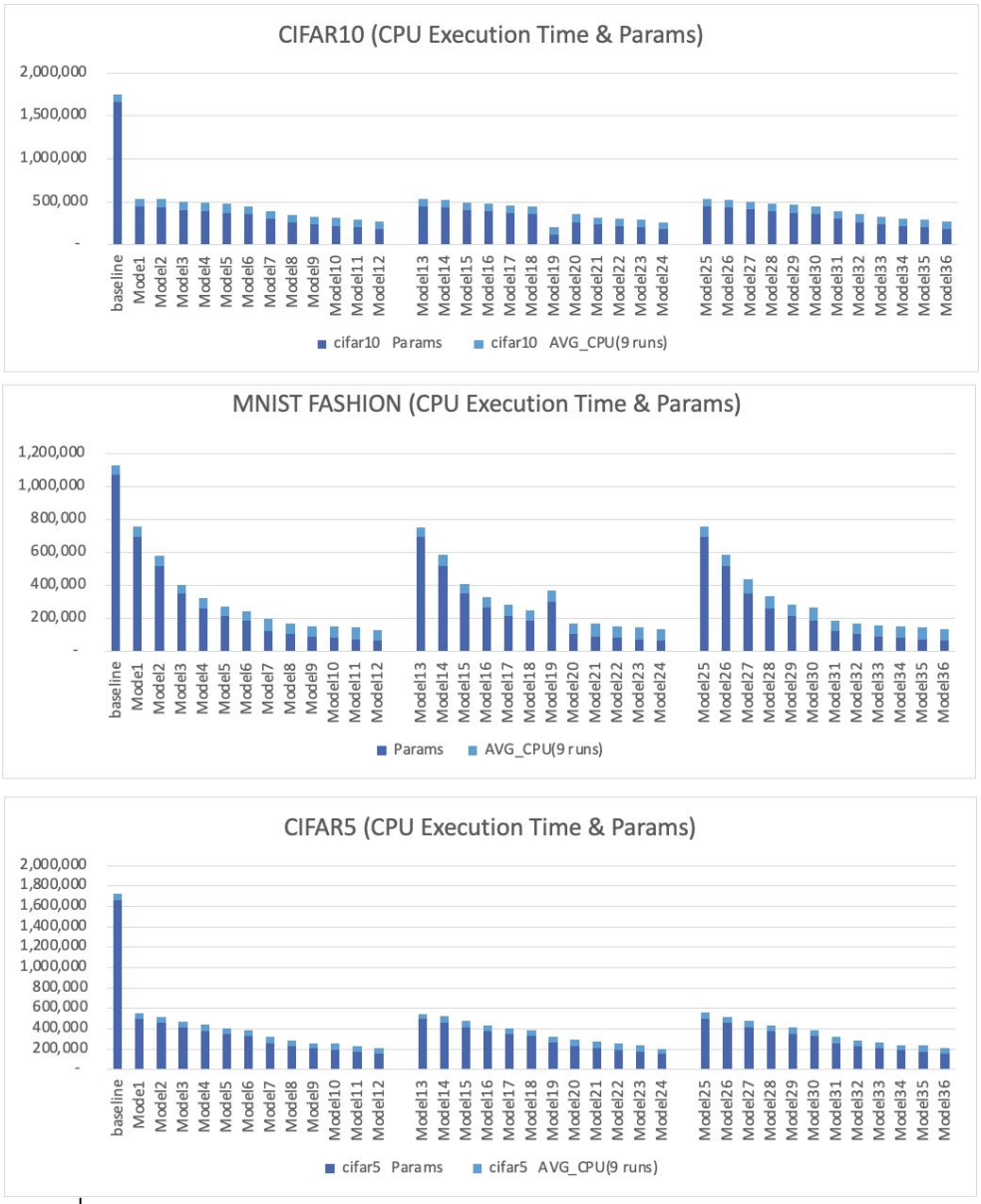

FIGURE 3.7: Comparative Performance Metrics of LetNet5 Across CIFAR-10, CIFAR-5, and MNIST Fashion Datasets

Chapter 4

MicroNets: A Multi-Phase Pruning Pipeline to Deep Ensemble Learning in IoT Devices

In the previous chapter, we introduced a novel technique called *EnSyth: A Pruning Approach to Synthesis of Deep Learning Ensembles*. This technique addresses the third and the fifth objectives of this thesis outlined in Chapter 1.

This chapter introduces the *MicroNets* framework, which builds upon the research discussed in the previous chapter. The framework aims not only to further reduce the memory usage of deep learning ensembles but also to enhance their generalisation capabilities to handle the noisy data generated by AIoT (Artificial Intelligence of Things) devices.

Experimental evaluations conducted on Raspberry Pis demonstrate the effectiveness of the *MicroNets* approach. It successfully generates lightweight models and leverages ensemble learning techniques, resulting in predictability levels that surpass those of ResNet and CIFAR10CNN baseline models by up to 7

The proposed approach has been published as a comprehensive research paper titled "MicroNets: A multi-phase pruning pipeline for deep ensemble learning in IoT devices" in the Computers and Electrical Engineering Journal by Elsevier. The paper can be found in volume 96, with the identification number 107581 [Alhalabi, Gaber, and Basura, 2021].

4.1 Introduction

The Internet of things (IoT) has emerged as one of the major technological advances that will contribute to forming the future of humanity [Sodhro et al., 2018], providing its potential applications, affecting our daily activities [Sodhro et al., 2021]. Those tiny devices are regarded as an optimal target for Machine Learning (ML) and deep neural networks (DNN) applications [Ahmad et al., 2020] because they require massive amounts of data for the generation of accurate prediction models. However, these AI techniques are known for their thirst for substantial computational power and cannot effectively run on these computationally constrained IoT devices. For example, deep learning models consist of millions of trainable parameters, requiring extremely powerful workstations to be trained. A VGG-16 has almost 138 million parameters and requires around 550 MB of memory [Simonyan and Zisserman, 2015]. On the other hand, a typical RAM for an IOT device does not exceed 500 MB.

Therefore, optimising IoT-driven intelligent systems [Sodhro et al., 2019] and pruning DNNs becomes paramount in the research community. Deploying DNNs

on edge devices brings numerous advantages, such as preserving privacy and achieving low latency. Many compression and acceleration strategies have been introduced, including neuron pruning [Yu et al., 2018a], network quantisation [Alom et al., 2018], etc. Most recently, researchers have shown great attention to deep learning compression pipelines as combining different pruning techniques leads to higher compression ratios [Luo, Chi, and Deng, 2019]. However, similar approaches are not optimised to run on complex IoT environments. The nature of the noisy data generated at the edge affects the prediction levels of the pruned models. It leads to poor performance on IoT applications that require stable model performance. To overcome those challenges, We propose "MicroNets" as the first deep learning framework that enables deep ensemble learning on edge devices. "MicroNets" lead to high compression ratios while not affecting the pruned model's accuracy by applying pruning and quantization. More importantly, synthesising diverse classifiers eliminates the high variance of deep learning models. Motivated by the lottery ticket hypothesis [Frankle and Carbin, 2019], this approach starts by generating a diverse set of pruned deep learning models using different hyperparameters of pruning algorithms. The aim is to find a diverse ensemble of subnetworks (a coalition of lottery tickets) instead of one to be deployed in a resource-limited environment. In general, The framework is as a wrapper that encapsulates any deep neural network model, providing a heuristic to generate an effective ensemble of pruned deep learning models, irrespective of the pruning method used. The only criterion that the pruning method should provide hyperparameters that could be set with different values to generate a diverse set of models. Overall, the main contributions of our proposed approach are:

- a novel framework designed to enable efficient deployment of deep ensemble learning on edge devices through deep ensemble learning, which maximises the generalisation of deep learning solutions at the edge-end, and provides robust ML solutions in complex IoT environments;
- a multi-phase pruning pipeline to generate light-weight deep learning models, which runs (as individuals or ensembles) on resource-limited devices without draining its resources; and
- a new clustering-based deep ensemble pruning technique to reduce the number of models in deep learning ensembles.

The remainder of this chapter is organised as follows: In Section 4.2, we present our proposed approach in detail, which is designed to address the third and the fifth objectives of this thesis outlined in Chapter 1. We discuss the underlying principles of our approach in great detail. In Section 4.3, we provide a comprehensive overview of the experimental setup we used to evaluate the effectiveness of our approach. We describe the datasets we used for training and testing our models and explain how we selected the models, edge clients, and other components of our distributed edge environment. Moving on to Section , we present the results of our experiments on the CIFAR10 and CIFAR100 datasets. We demonstrate that our approach can achieve comparable or better accuracy than A CNN baseline model while using fewer resources, making it a promising solution for edge computing scenarios. We also describe the deployment of our *MicroNets* on a distributed edge environment, showing that it can be easily integrated with existing systems and can improve the system's overall performance.

Finally, in Section 4.6, we summarise our proposed approach and its results, highlighting its advantages and limitations. We also discuss the implications of our work for future research in the field of edge computing. We conclude by introducing

our next contribution, which builds on the insights and techniques presented in this thesis to address new challenges related to data privacy and security.

4.2 Method

In this section, we describe in detail the proposed approach. We start by justifying the rationale behind generating a pool of diverse deep learning classifiers (ensemble of deep learning models). Then, We move to illustrate the ensemble's journey through the pruning pipeline. The first step in this journey aims to reduce network complexity by applying weight pruning. The next step involves using integer quantisation to enable efficient deployment in resource-limited environments. The final step in the pruning pipeline reduces the number of models in the ensemble. Here, we propose a new method to prune a deep learning ensemble by adopting a clustering-based pruning method.

4.2.1 Pool Generation and Weight Pruning

Deep learning models can solve non-linear problems by learning via a stochastic training algorithm. This enables them to learn complex relationships between inputs and outputs and approximate any mapping function. However, One major drawback of deep learning models is high variance which makes them highly dependent on (1) the training set; and (2) the conditions that have been applied to the training process (initial weights values, loss function, optimiser function, etc.). This could affect the network's ability to generalise and consequently produce a model that makes different predictions when the same conditions apply. To overcome these challenges, we follow a similar approach applied in [Alhalabi, Gaber, and Basurra, 2019]. We start with a baseline model, and then a weight pruning method [Zhu and Gupta, 2017] is applied. During the training process, a binary mask is added to each elected layer for pruning. The mask has the same size and shape as the layer's tensor, determining the weights actively participating in the forward execution graph. Then the weights in each mask are sorted according to their absolute values. The weights with the smallest magnitude values are set to zero, leading to initial sparsity levels (s). During the backpropagation, the weights that had been marked in the forward execution are not updated. This process will be executed automatically and gradually where the sparsity is increased from an initial value (s_i) (typically 0) to final sparsity (s_f) over several pruning steps (n), starting at a training step (t_0) with a pruning frequency (Δt):

$$s_t = s_f + (s_i - s_f)\left(1 - \frac{t - t_0}{n\Delta t}\right)^3, \tag{4.1}$$

where $t \in \{t_0, t_0 + \Delta t, ..., t_0 + n\Delta t\}$, ($s$,$n$,$t$,$\Delta t$) are the hyperparameters of this pruning technique. We vary these values to generate a pool of diverse pruned deep learning models. However, compressing the model's size using such a pruning technique is not adequate for AI edge devices, they do not support floating-point operations. Thus, integer quantisation is applied as shown in the following step.

4.2.2 Post Training Integer Quantisation

We adopt 8-bit integer quantisation to approximate the floating-point values using the following formula:

$$real\_value = (int8\_value - zero\_point) \times scale \tag{4.2}$$

All weights are represented by $int8$ two's complement values in the range $[-127, 127]$ with $zero\_point$ equals to (0). Similarly, activations/inputs are represented by $int8$ two's complement values in the range $[-128, 127]$ with $zero\_point$ in the range $[-128, 127]$. After applying integer quantisation to the proposed pool, we get a set of pruned-quantised classifiers that have different characteristics. Some models could have excellent accuracy and inference time performance, while others may not. In the next step, we reduce the number of learners in the pool by applying ensemble pruning assuming the pool as an ensemble of deep-learning models.

4.2.3 Ensemble Pruning

According to [Zhou, 2012] ensemble pruning techniques could be classified into the following categories: ordering-based pruning, optimisation-based pruning and clustering-based pruning.

Clustering-based pruning involves a two-step process. Initially, it segregates base models into distinct groups according to a specific criterion, aiming to positively influence the overall ensemble's efficacy. To achieve this segregation, a variety of clustering techniques have been utilized. These include hierarchical agglomerative clustering [Kuncheva, 2014], deterministic annealing [Bakker and Heskes, 2003], k-means clustering [Qiang, Shang-xu, and Sheng-ying, 2005] [Lazarevic and Obradovic, 2001], and spectral clustering [Zhang and Cao, 2014], most of which apply some form of diversity-oriented criteria. For instance [Giacinto, Roli, and Fumera, 2000] computed the likelihood of classifiers erring differently on a distinct validation set. Similarly, [Lazarevic and Obradovic, 2001] considered the Euclidean distance within the training set. Additionally, [Kuncheva, 2014] suggested the use of a diversity matrix for both hierarchical and spectral approaches.

The subsequent phase involves the selection of a representative base learner from each group. For example, a novel model was trained to represent each cluster's center in [Bakker and Heskes, 2003]. Conversely, Giacinto and colleagues [Giacinto, Roli, and Fumera, 2000] opted for the classifier furthest from other clusters. The approach in [Lazarevic and Obradovic, 2001] involved sequentially excluding models from the cluster, starting with the least precise. The selection criterion in [Qiang, Shang-xu, and Sheng-ying, 2005] was the model's classification accuracy.

An important consideration is the determination of the optimal cluster quantity, which can be guided by the validation set's performance metrics [Qiang, Shang-xu, and Sheng-ying, 2005]. With fuzzy clustering approaches, the number of clusters may be decided using indices that rely on membership values and data characteristics, or on statistical indexes for an automatic determination [Krawczyk and Cyganek, 2017].

As seen above, the clustering-based methods aim to partition the classifiers into different clusters where the classifiers that behave similarly will be grouped. As a result, a set of clusters with varying levels of diversity is composed. Finding such clusters is critical to our approach as it maximises the generalisation of the final solution and overcomes any potential overfitting issues. Although many studies offer clustering-based ensemble pruning [Qiang, Shang-xu, and Sheng-ying, 2005],

they do not support deep learning models and computer vision data sets. Hence, we propose a new clustering-based pruning technique for deep learning ensembles that aims to maximise the accuracy and diversity of the final ensemble. The following illustrates how our approach tackles the accuracy and diversity of a deep-learning ensemble.

Ensemble accuracy: The accuracy of an ensemble is generally improved by choosing individuals with high predictability levels [Zhou, 2012]. Based on this, we do descend-ranking to the members of the cluster. The ranking is done according to the accuracy against a validation set (a subset of the training set). Then we pick up the top-performing models from each generated cluster based on two strategies, which will be explained further in (subsection- 4.2.4). This ensures only high-performing models are part of the final deep learning ensemble that runs on the edge devices.

Ensemble diversity: Diversity in ensemble learning refers to the differences in the predictions made by the individual classifiers within the ensemble. A diverse ensemble is one where the classifiers make independent errors, which can lead to better generalisation and improved performance on unseen data. The rationale is that diverse classifiers can capture different aspects of the data, thus reducing the likelihood of making the same errors [Kuncheva, 2014]. Different measure has been proposed to calculate the diversity between the classifiers of an ensemble. For example, The disagreement measure is one way to quantify diversity, calculating the proportion of instances where two classifiers disagree in their predictions, as given by [Kuncheva, 2014]:

$$Disagreement(C_i, C_j) = \frac{1}{N} \sum_{n=1}^{N} I(C_i(x_n) \neq C_j(x_n)) \tag{4.3}$$

Where:
C_i, C_j: Two individual classifiers within the ensemble.
N: The total number of instances in the dataset.
I: An indicator function that returns 1 if the condition is true and 0 otherwise.
x_n: The nth instance in the dataset.

The Q-statistic further explores this by measuring the degree of similarity between pairs of classifiers, with its formula below:

$$Q(C_i, C_j) = \frac{N_{11}N_{00} - N_{10}N_{01}}{N_{11}N_{00} + N_{10}N_{01}} \tag{4.4}$$

Where:
N_{11}: The number of instances where both classifiers C_i and C_j correctly predict.
N_{00}: The number of instances where both classifiers C_i and C_j incorrectly predict.
N_{10}: The number of instances where classifier C_i is correct and C_j is incorrect.
N_{01}: The number of instances where classifier C_i is incorrect and C_j is correct.
The above formula indicates how often classifiers make the same decision, either correct or incorrect [Webb, 2000]. Entropy measures bring a different perspective by assessing the unpredictability in the predictions across the ensemble ([Zhou, 2012] encapsulated in the formula

$$E = -\sum_{k=1}^{K} p_k \log_2 p_k \tag{4.5}$$

Where:
E: The entropy measure for the ensemble.

p_k: The proportion of classifiers that predict the k-th class.
K: The total number of classes.

Lastly, generalised diversity measures combine multiple pairwise diversity measures into a single framework, providing a comprehensive assessment of diversity within the ensemble [Kuncheva, 2014], the formula is introduced as:

$$GD = f(D_1, D_2, \ldots, D_M) \tag{4.6}$$

Where:
GD: The generalised diversity measure for the ensemble.
$D_1, D_2, \ldots, D_M$: Different diversity measures used in the ensemble.
f: A function that combines the different diversity measures into a single framework.
M: The total number of diversity measures considered.

These measures of diversity are instrumental when pruning the initial pool of models using different hyperparameter values to achieve both diversity and sparsity within the ensemble.

Let S be the pruning set (a subset of the training set) $\in \mathbb{R}^{m \times n}$ where m is the number of samples and n is the number of labels. Let v be a deep learning model $\in P$, the pool of pruned-quantised models, the prediction of the model $v_i \in P$ for a class $c_j \in S$ is $y_{i,j} \in [0, 1]$ where

- $i = 1, 2, \cdots, m$ where m is the index of models.
- $j = 1, 2, \cdots, n$ where n is the index of classes.

We construct the output of the pool P for each sample c_i as:

$$A = \begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,n} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m,1} & y_{m,2} & \cdots & y_{m,n} \end{bmatrix}$$

As the output of the pool has been defined, we move now to define how clustering is applied to the output of the pool A. Let's consider x as a row vector $\in A$, the value of x is defined as: $x = [y_{b,1} y_{b,2}, \cdots, y_{b,n}]$; such that the index b is defined as:

$$b = argmax \begin{bmatrix} max(y_{1,1}, y_{1,2}, \cdots, y_{1,n}) \\ max(y_{2,1}, y_{2,2}, \cdots, y_{2,n}) \\ \vdots \\ max(y_{m,1}, y_{m,2}, \cdots, y_{m,n}) \end{bmatrix}$$

Based on this, a clustering matrix C could be defined on a total number of images o in the pruning set as:

$$C = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_o \end{bmatrix}$$

Next k-means method is applied on C and the output is represented by R, $R = kmeans(C, k)$ where k is the number of clusters and $R = [r_1, r_2, \cdots, r_o]$: r_i is the index of the cluster where x_i belongs to. Once clusters are created, the best candidates from each cluster will be selected to compose the final ensemble that will be deployed to the distributed constraint-limited environment.

4.2.4 Representative Selection Strategies

At this stage, different clusters are generated, and the classifiers with similar characteristics are grouped into one cluster. However, each cluster may contain a large number of models; thus, we propose two strategies to pick up the best representatives from the resulting clusters using r that was defined before.

1. **Accuracy first**: in this strategy, we consider the accuracy of a cluster to be equal to the average accuracy of all models belonging to this cluster. Based on this, we rank the clusters according to their accuracy and then pick the cluster with the highest accuracy. Next, we sort the chosen cluster's models in descending order (according to the accuracy against the pruning set), and pick the top 5 models. Those models will be deployed to the target IoT devices;

2. **Diversity first**: here, we give priority to the diversity of the final solution. Thus, we select one classifier (best accuracy) from each generated cluster to be deployed on the resource-limited devices.

The next step involves deploying the models to a distributed resource-constraint environment, where each node in this environment uses max-voting [Zhou, 2012] as an ensemble learning method. The predictions of the models are considered as votes, and the class that receives the maximum number of votes is considered the ensemble's forecast. Let's consider $W = m_1, \cdots, m_N$ as an ensemble of the deployed models m_i. The prediction of the ensemble for a test example using max-voting is the class that receives the maximum support $\eta_{final}(W)$ from all models in the ensemble. As such, we define the ensemble output as:

$$\eta_{final}(W) = argmax_{j \in \{1,\cdots,C\}} \sum_{i=1}^{N} y_{i,j}$$

;

The detailed steps of the *MicroNets* discussed above are elaborated further in the Algorithm 7. Figure 4.1 depicts the process of applying *MicroNets*, as elaborated in the details of this section. In the next section, we move to the experimental setup and the results of applying this method on bench-marking datasets.

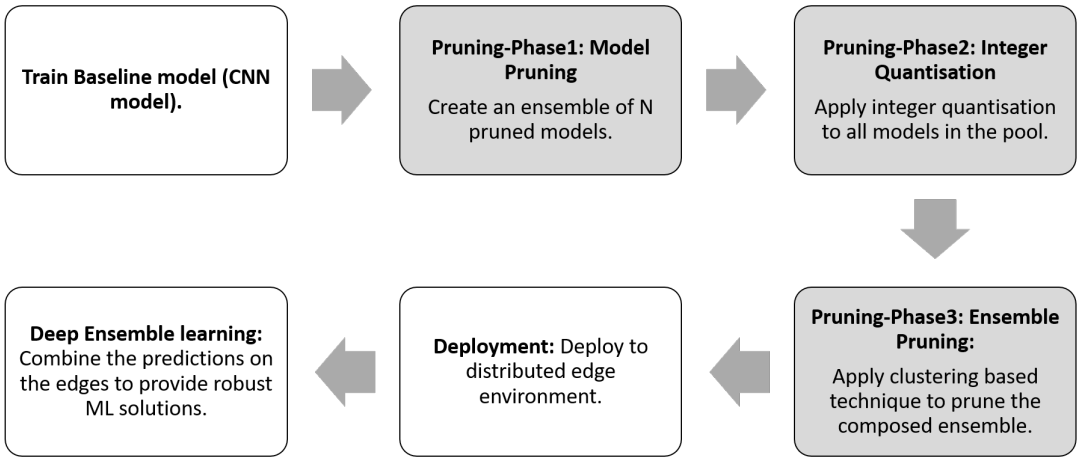

FIGURE 4.1: MicroNets: A Multi-Phase Pruning Pipeline to Deep Ensemble Learning in IoT Devices

Algorithm 7 MicroNets pseudo-code

Choose a baseline model v;
Set N as the number of models in the pool, and initialise P as a set of deep learning models.
weigh-pruning & interger-quantisation
for i $\leftarrow$ 1 to N **do**
  $P.Append(int\_quantisation(prune(v, H)))$ where $H = \{h_1, h_2, \cdots, h_n\}$ are hyperparams of the *prune* function;
end for

Clustering-based ensemble pruning
Initialise C as a two dimensional matrix;
for all models in P **do**
  $calculate(A)$
  $calculate(b)$
  $x \leftarrow [y_{b,1}y_{b,2}, \cdots, y_{b,n}]$
  C.append(x);
end for
$R \leftarrow kmeans(C, k)$ where K is the number of clusters; {the result is a set of clusters R}
Representatives Selection Process
1.Accuracy First:
Intialise A as a list of floating point numbers;
for r_i in R **do**
  $a_i \leftarrow$ accuracy(r_i);
  A.append(a_i);
end for
$b\_c \leftarrow max(desc\_order(A))$ where b_c is the best cluster with the highest accuracy, desc_order(descending order);
Deploy the top 5 models in b_c to an IoT device;
2.Diversity First:
for r_i in R **do**
  for v_j in r_i where v_j is a deep learning model **do**
    $a_J \leftarrow$ accuracy(v_j);
    A.append(a_j);
  end for
  $b\_m \leftarrow max(desc\_order(A))$
end for
Deploy the top 5 models in b_m to an IoT device;

4.3 Experimental Study

This section evaluates the performance of *MicroNets* using CIFAR10CNN and ResNetV2 models on CIFAR10 and CIFAR100 datasets, respectively. Then, we provide other results related to inference time, heat and power consumption on Raspberry Pi devices. Those measures are essential to show the effectiveness of our approach in preserving the resources of IoT devices.

4.3.1 Datasets and Models

Datasets: **CIAFR-10** consists of 70,000 images (28×28 coloured) split into 50,000 images as a training set and 10,000 images as a testing set. The images span over 10 objects' categories [0: airplane, 1: automobile, 2: bird, 3: cat, 4: deer, 5: dog, 6: frog, 7: house, 8: ship, 9: truck]. **CIFAR-100** is similar to CIFAR10 except that it has 100 classes holding 600 images each. **Models**: **CIFAR10-CNN** is a basic convolutional neural network (CNN) model with four convolutional layers with ($32 \times 3 \times 3$) filter size for the former two and ($64 \times 3 \times 3$) filter size for the rest. **ResNetV2**: is a child of Deep Residual Networks (RNNs) family that achieved groundbreaking work in the deep learning community in the last few years. ResNetV2 is the second version of ResNet and the main improvement in V2 is related to the arrangement of layers in the residual block. The model's input is a 299×299 pixels, and the output is the probability distribution for the predicted classes [He et al., 2016b].

4.3.2 Implementation Details

Model Training and Pruning

Our experimental involves two baseline models: CIFAR10-CNN and ResNetV2. The CIFAR10-CNN model is developed from the ground up and trained on the CIFAR10 dataset, achieving a test accuracy of 74% with a model size of 9,386 KB. Conversely, the ResNetV2 model is trained on the CIFAR100 dataset, reaching a test accuracy of 67% and a model size of 3,575 KB.

We employed the weight pruning technique as detailed in [Zhu and Gupta, 2017] on both baseline models, which led to the formation of two distinct sets of models, or 'pools,' with each pool comprising one hundred individually pruned models. To ensure a broad spectrum of model variations, we assigned a range of values to the pruning hyperparameters, choosing these values at random within predefined limits. The specific hyperparameters and the ranges from which their values were selected are clearly outlined in Table 5.1.

Integer Quantisation

Following the pruning phase, we applied integer quantisation to all models in both pools. This technique converts the models' weights and activations to 8-bit integers, which not only reduces the models' sizes but also optimises them for performance on dedicated integer arithmetic hardware like Raspberry Pi devices.

Ensemble Pruning

After applying quantization to the pools of pruned models, we proceeded to refine the number of models in each pool by introducing a novel ensemble pruning approach. This method involved organizing the models into clusters within their

TABLE 4.1: Hyperparameters, Weight pruning

Hyperparameter	Values_range
Epochs	[3,4,5,6,8]
Batch_size	[32,64,128]
Loss	[categorical_crossentropy, mean_squared_error, mean_absolute_error]
Optimiser	[SGD,adam,Nadam,Adadelta]
Initial_sparsity	[0.1,0.6](range)
Final_sparsity	[0.7,0.9] (range)
Frequency	[100,200,300,400]

respective pools, with the number of clusters being predetermined and denoted by the variable K. The selection process for models within these clusters was governed by two key principles:

- Accuracy first: In our accuracy first approach, we assign the highest priority to the accuracy of the models. To do this, we first calculate the average accuracy of the models within each cluster, treating this average as the representative accuracy of the cluster itself. Following this, we rank the clusters in order of their calculated average accuracy (descending order). From the cluster that emerges with the highest average accuracy, we then select the top five models that exhibit the best individual performance.
- Diversity first: In our 'Diversity first' approach, we emphasise the selection of a model from each cluster that not only showed strong performance metrics but also contributed to the overall diversity of the model ensemble.

Deployment to a Distributed Edge Environment

When the chosen pruned and quantised deep learning models from the previous step are ready to be deployed on IoT devices, each device holds two or more deep learning models and utilises deep ensemble learning to combine the models' predictions. However, the process of combining predictions from different models requires additional computational power. Thus, it is essential to bring ML inferencing power to edge devices by attaching hardware accelerators. As such, we use the Coral USB Accelerator[1] to add a portable Edge TPU coprocessor to the PI devices.

4.4 Experimental Setup

The development workstation for training/pruning deep learning models is hosted on the Google Cloud Platform with the following configuration: 8 vCPUs, 30 GB memory. 2 × NVIDIA Tesla K80 using Tensorflow 1.15 dev. We use Raspberry PI 3 Model B v1.2 attached to Google Coral to create a real edge computing environment. Google Coral is provided with a TPU coprocessor capable of performing 4 trillion operations (tera-operations) per second. Figure 4.2 represents the distributed edge environment that we use to evaluate the performance of our proposed approach. For compatibility reasons with the Edge TPU coprocessor, all models have to be

[1] https://coral.ai/docs/accelerator/datasheet/

compiled before deployment using a compiler tool [2]. The compiler tool provides a vital option called Co-Compilation[3] to boost the inferencing while running multiple models on the same IoT device. It allows various models to share the Edge TPU RAM and cache their parameter data together, eliminating the need to clear the cache each time you run a different model. In section 4.5, we investigate the effect of co-compilation on the inferencing time. As shown in Figure 4.2, two Rasp-

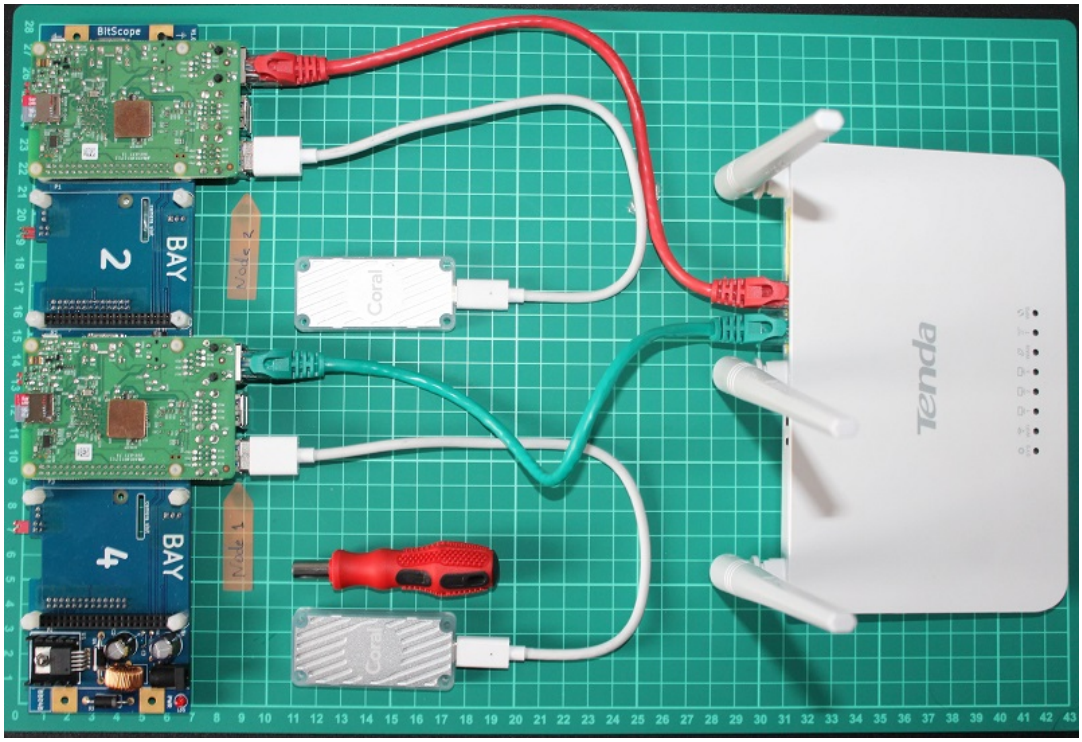

FIGURE 4.2: Two raspberry pi devices connected to: Google Coral Accelerator and a router

berry Pi nodes (running Raspbian GNU/Linux 9 (stretch) and Tensorflow-2.1.0) are connected to Tenda F3 300Mbps router, and the Coral USB Accelerator. Node1 will host two deep learning models, while Node2 will host three models. Additionally, a master node (webserver) is also connected through the router to coordinate the prediction process. First, it sends both nodes an integer as the number of required samples that need to be tested on the testing set. Next, the nodes apply deep ensemble learning and return their prediction to the server. At the server, the predictions are combined from both nodes again to produce the final result. While performing the predictions on Node2, we measure the energy consumption using YOTINO *USB Voltage and Current Detector Meter Capacity, Accuracy: ±1%'*, furthermore we monitor the temperature of Node2 using *Etekcity Lasergrip 1080 Non-contact Digital Laser IR Infrared Thermometer, Accuracy: ±2% or 2°C'*.

[2] https://coral.ai/docs/edgetpu/compiler/#system-requirements
[3] https://coral.ai/docs/edgetpu/compiler/#co-compiling-multiple-models

4.5 Results and Discussion

4.5.1 Results on CIFAR100

The ResNet model is trained on CIFAR100 data set, and then weight pruning is applied to generate a pool of 100 lightweight models. The maximum accuracy in the pruned pool is 66%, while the minimum accuracy is 1%. The model size after pruning is remarkably dropped, and the average model size in the pool is 1.293 KB which is almost 63% smaller than ResNet baseline model. A further reduction in model size is gained through integer quantisation; the average pruned model size is only 305 KB. This represents a significant compression ratio of up to 92%, which is a key success factor of the *MicroNets* towards enabling efficient deep ensemble learning on resource-limited devices. The next step in *MicroNets* is to reduce the number of models in each pool and select the best representatives based on accuracy first and diversity first strategies. Figure 4.3 compares the results obtained after adopting the proposed strategy for model selection and applying ensemble learning. As a side note here, the accuracy is tested over five hundred samples on CIFAR100 testing set, and the baseline accuracy on this subset (after quantisation) is 51%. We apply quantisation to the baseline model to make it similar to the generated models by *MicroNets* Also, from Figure 4.3, we can see that the synthesis of the models following

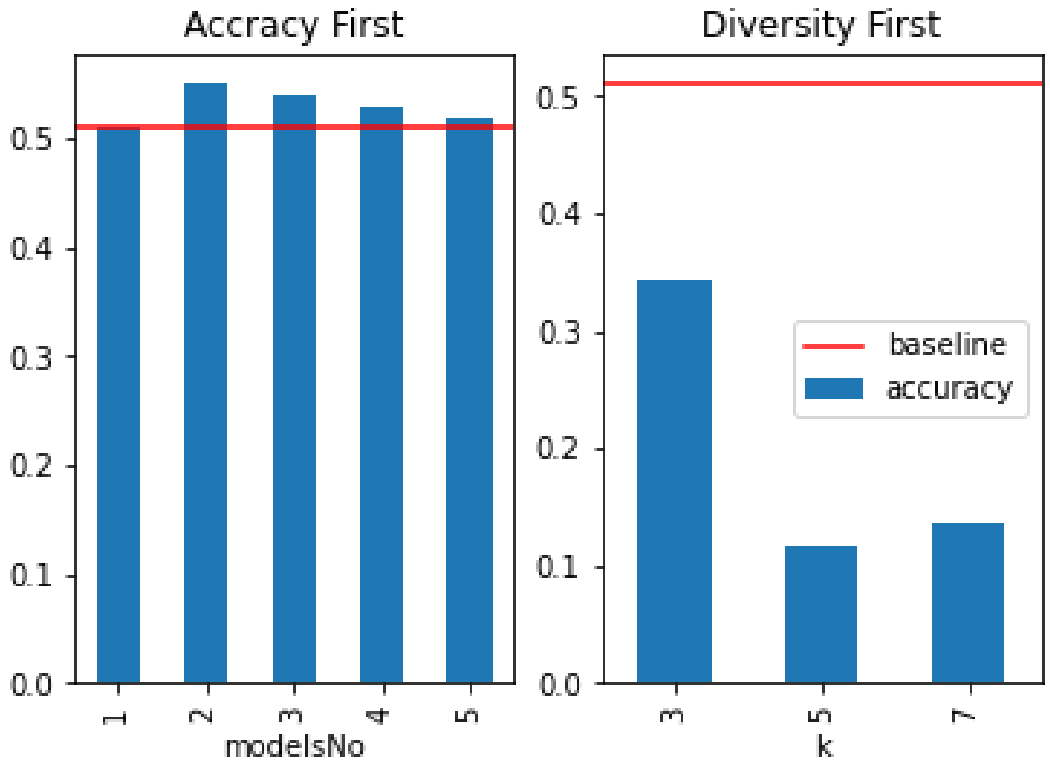

FIGURE 4.3: CIFAR100 Accuracy Comparative: "Accuracy First" versus "Diversity First"

the "Accuracy First" strategy was able to achieve higher accuracy than the baseline model. The highest accuracy is achieved when an ensemble of the two models is created (55%). On the other hand, the accuracy was remarkably dropped when "Diversity First" is adopted when $k = 3, 5, 7$. However, when $k = 3$, the accuracy reaches 34%.

Table 4.2 summarises the results on CIFAR100.

TABLE 4.2: Summary of MicroNets results on CIFAR100

Metric	**Result**
Maximum Accuracy (Pruned Pool)	66%
Model Size (Post-Pruning)	1.293 KB (63% smaller than ResNet baseline)
Model Size (Post-Pruning and Quantization)	305 KB (92% compression ratio)
Accuracy (CIFAR100 Testing Set, Baseline)	51%
Accuracy (Ensemble of "Accuracy First" Models)	55%
Accuracy ("Diversity First" with k=3)	34%

4.5.2 Results on CIFAR10

The minimum accuracy of the pruned pool on CIFAR10 dataset is 1%, while the maximum is 80% (on the training set). The average size of the models is 4,926 KB (around 48% reduction in model size compared to the baseline). After quantisation, the models became even smaller, and the size became 1,233 KB (approximately 87% reduction in size). In a similar manner to CIFAR100, Figure 4.4 presents the accuracy of *MicroNets* on five hundred images from CIFAR10 testing set, and the accuracy of the baseline model on this subset reaches 78%. What stands out in Figure 4.4 is the success of *MicroNets* in providing more accurate decisions than the baseline model in both selection strategies. In the case of "Accuracy First", all the ensembles are able to offer outperforming results, and the maximum accuracy is reached when the ensemble size is three (82%). In the "Diversity First" strategy, when $k = 3$, the accuracy of the composed ensemble is 79%. However, the accuracy was almost the same as the baseline, when the ensemble size is five $k = 5$ (77%), and when $k = 7$ (76%). Furthermore, the results are shown in 4.3 compare the accuracy of the proposed strategies against a random selection of the models.

TABLE 4.3: Accuracy Comparison: "MicroNet's Clusters" versus "Random Model Selection"

K	**Diversity First**		**Random**	
	CIFAR10	CIFAR100	CIFAR10	CIFAR100
3	**0.798**	**0.344**	0.398	**0.178**
5	0.778	0.116	0.752	0.16
7	0.76	0.136	**0.754**	0.118
NumberOfModels	**Accuracy First**		**Random**	
	CIFAR10	CIFAR100	CIFAR10	CIFAR100
1	0.814	0.518	0.234	**0.09**
2	0.81	0.55	0.208	0.072
3	**0.822**	**0.546**	**0.288**	0.068
4	0.81	0.53	0.282	0.058
5	0.81	0.522	0.286	0.068

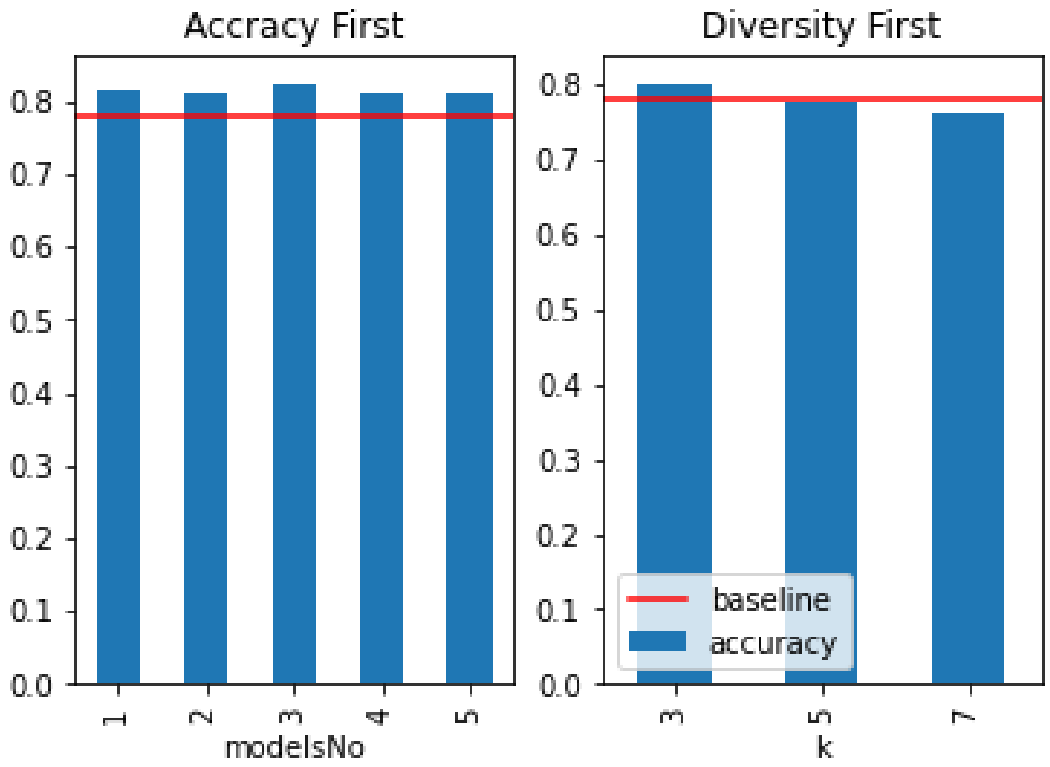

FIGURE 4.4: CIFAR10 Accuracy Comparative: "Accuracy First" versus "Diversity First"

So far, the results indicate that *MicroNets* can provide lightweight high performance models in terms of size and accuracy. Table 4.4 gives more details about size reduction (in KB) after applying the proposed pruning pipelines. It also shows the size of the ensembles in both accuracy-first and diversity-first approaches.

It is apparent from Table 4.4 that the models are effectively compressed to reach around 91% compression ratio. Additionally, the size of the ensembles (even when $k = 7$) is still smaller than that of the baseline. Moving now to provide results related to heating, inference time and energy consumption on Raspberry Pis, as a typical IoT device. As explained earlier, we compile the models with the edge_tpu compiler using two different options: regular compilation process and co-compilation process. Table 4.5 shows the results of inference time (in seconds) on a different number of CIFAR10 testing set. It also shows the external temperature of Node2 which holds three models in the rest mode (no inferencing) and the peak mode (while inferencing).

A closer inspection of Table 4.5 above shows that both compilation methods have almost similar results when the number of CIFAR10 testing samples are between [1-5]. The external temperature of Node2 has not changed (before/after inferencing) and the average time needed to perform an inference is 3.5 seconds. However, when we increase the number of testing samples passed to the models, *Co-Compilation* shows a huge difference in the required time to perform the inference. For instance, when we ask Node2 to make an inference on a hundred images, it takes only 4.56 seconds. On the other hand, *Regular-Compilation* experiences a considerable delay in performing the same task with almost 15.5 seconds. Additionally, the external temperature of Node2 has slightly increased while performing the prediction on 100 images (0.4 degrees). In short, we can conclude that if the devices need to perform inference on a large number of samples, then the models should be compiled with

TABLE 4.4: Effect of "MicroNets' on the size of the models (in KB)

		CIFAR10	CIFAR100
Size changes after pruning pipelines	Baseline	9,836	3,575
	Pruning Applied	4,926	1,293
	Quantisation Applied	1,233	305
Diversity-First	k=3	3,699	915
	k=5	6,165	1,525
	k=7	8,631	2,135
Accuracy-First	1	1,233	305
	2	2,466	610
	3	3,699	912
	4	4,932	1,220
	5	6,165	1,525

the *Co-Compilation* option as this benefits from using models parameter cashing. Regarding energy consumption, Figure 4.5 shows the changes of the "load current" while Node2 performs inferencing on various samples from the CIFAR10 testing set.

4.5.3 Energy monitoring

The energy consumption of computing devices is a critical factor to consider in constrained and resource-limited environments such as Internet of Things (IoT) devices, which typically have limited battery life or power supply. In such environments, the energy consumption of a deployed application or system needs to be monitored and optimized to prolong the device's battery life or minimise the need for frequent recharging. In this sub-section, we will discuss two measurements the Input voltage (DCV) and the Load current(DCA) measurements captured during running MicroNets on Raspberry Pi devices.

The results of the energy monitoring measurements show that the co-compilation mode had a slightly lower load current than the regular compilation mode when running a small number of models, but had a higher load current when running more significant numbers of models. For example, when running 100 models, the co-compilation mode had a load current of 0.42 DCA while the regular mode had a load current of 0.44 DCA. However, when running only one model, both modes had a load current of 0.3 DCA. It's worth noting that the input voltage remained constant at 5.03 DCV for both modes and across all numbers of models tested. These measurements are helpful in understanding the energy consumption of a Raspberry Pi device when using an edge TPU accelerator and can inform decisions on system optimisation and energy efficiency.

Table 4.6 provides information on the input voltage and load current of a Raspberry Pi device under three different conditions: (1) when no accelerator is attached, (2) when a Coral accelerator is attached, and (3) when running a Tflite server. The measurements are taken in DC volts (DCV) for input voltage and DC amperes (DCA) for load current.

Table 4.6 shows that the input voltage remains constant at 5.03 DCV across all three conditions. This indicates that the power supply for the device remains stable throughout the experiments.

TABLE 4.5: Inference (Milliseconds) and Temperature (Celsius) Results, Co/Reg Compilation

Samples	Compilation	Node1	Node2	Temp_idle	Temp_Peak
100	co	4.242	4.562	35.9	36.2
50	co	3.790	4.018	36	36.1
25	co	3.638	3.818	36	36.1
10	co	3.518	3.612	35.9	36
5	co	3.506	3.597	35.8	35.8
4	co	3.494	3.563	36.1	36.1
3	co	3.492	3.580	35.8	35.8
2	co	3.509	3.575	36.1	36.1
1	co	3.503	3.546	36	36
100	reg	11.504	15.476	36.1	36.5
50	reg	7.605	9.544	36.1	36.4
25	reg	5.473	6.421	36.1	36.2
10	reg	4.196	4.551	36.1	36.1
5	reg	3.756	3.954	36.1	36.1
4	reg	3.693	3.874	36	36
3	reg	3.631	3.769	35.9	35.9
2	reg	3.548	3.658	36	36
1	reg	3.548	3.541	36	36

TABLE 4.6: Input voltage and load current of Raspberry Pi device

	Input voltage (DCV)	Load current (DCA)
No accelerator attached	5.03	0.26
With Coral attached	5.03	0.3
Running Tflite server	5.03	0.45

However, the load current varies depending on the different conditions. When no accelerator is attached, the load current is 0.26 DCA, which increases slightly to 0.3 DCA when a Coral accelerator is attached. This suggests that the Coral accelerator consumes additional power, resulting in a slightly higher load current.

The load current increases further to 0.45 DCA when running a Tflite server, indicating a slightly higher power consumption when the device is running a server workload. This increase in power consumption is expected since running a server workload typically requires more computational resources and, therefore, more power. In order to monitor the energy consumption of a Raspberry Pi device during operation with an edge TPU accelerator, we introduced new measurements to compare the input voltage and load current for two different complication modes (on node2): co-compilation and regular compilation. Co-compilation is a technique where the code is compiled simultaneously for both the host processor and the edge TPU, while regular compilation only compiles for the host processor. We recorded both modes' input voltage and load current and compared the results. The results of the energy monitoring measurements show that the co-compilation mode requires a slightly lower load current than the regular compilation mode when running a small and large number of models. For example, when running 100 models, the co-compilation mode had a load current of 0.42 DCA while the regular mode had

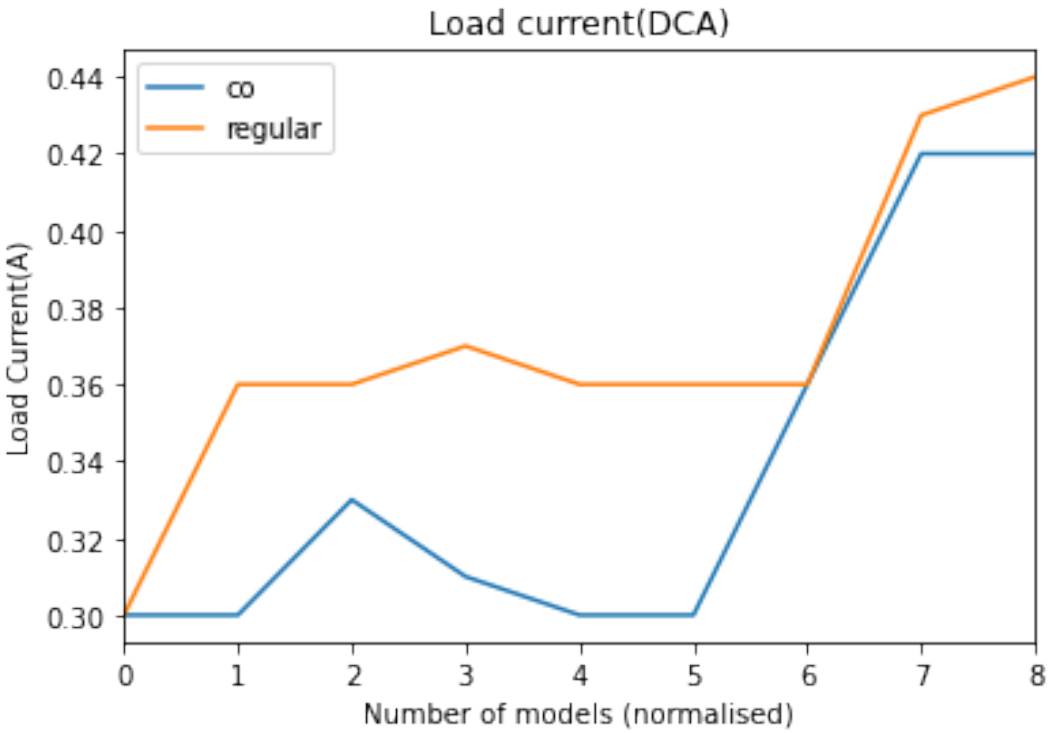

FIGURE 4.5: Energy Consumption of Micronets' Ensembles

a load current of 0.44 DCA. Similarly, when running 2 models, the co-compilation mode had a load current of 0.3 DAC, on the other hand, the regular compilation had 0.36 DAC. However, when running only one model, both modes had a load current of 0.3 DCA.

It's worth noting that the input voltage remained constant at 5.03 DCV for both modes and across all numbers of models tested. These measurements are useful for understanding the energy consumption of a Raspberry Pi device when using an edge TPU accelerator and can inform decisions on system optimization and energy efficiency. The complete results of these measurements can be found in Table 4.7 and Table 4.8.

TABLE 4.7: Energy Monitoring Results for Regular Compilation Mode

Number of models	Input voltage (DCV)	Load current (DCA)
100	5.03	0.44
50	5.03	0.43
25	5.03	0.36
10	5.03	0.36
5	5.03	0.36
4	5.03	0.37
3	5.03	0.36
2	5.03	0.36
1	5.03	0.3

From Figure 4.5, it can be seen that Node2 pulls (0.3 A) in the rest mode. While making the inference(using co-complied models, and the number of samples is less than 50), the "load current" is almost the same. The "load current" is increased

TABLE 4.8: Energy Monitoring Results for Co-compilation Mode

Number of models	Input voltage (DCV)	Load current (DCA)
100	5.03	0.42
50	5.03	0.42
25	5.03	0.36
10	5.03	0.3
5	5.03	0.3
4	5.03	0.31
3	5.03	0.33
2	5.03	0.3
1	5.03	0.3

by nearly 0.12 when Node2 tries to make predictions on fifty or a hundred images. When an inference is made with regular-complied models, the "load current" increases by a small degree even when the number of testing samples is less than fifty; it is worth mentioning that in both compilation methods, the input voltage for Node2 during the inference has not changed (5.04 V). This indicates that co-compilation does not only improve the inference time, but also reduces the required energy to run ensembles on resource-limited devices.

4.5.4 Computational Complexity

To understand the complexity of the proposed method, we first need to calculate the complexity of a deep learning model v. Assuming k is the number of layers in v, m/n are the input/output dimensions, N is the total number of parameters, and w is the depth of a neural network. Based on this, the complexity of a ResNet model as a linear programming problem (LP) for a given sample of data (size D) could be defined as: $O\Big((m\log(m))\Delta^{O(K^2)}/\epsilon^{n+m+N}D\Big)$ [Bienstock, Muñoz, and Pokutta, 2022]. Where G denotes to the computational graph of v, Δ the maximum vertex in-degree in G while $\epsilon > 0$. Similarly, the complexity on CIFAR10CNN in LP is: $O\Big((mw^{O(K^2)}/\epsilon)^{n+m+N}D\Big)$. Moving now to define the complexity of the proposed ensemble pruning technique. First, the complexity of the descending order in the worst case is $O(nlogn)$; k-means complexity is $O(n^{dk+1})$; where n is the number of entries that need to be clustered, k is the number of clusters and d is the number instances in the pruning set. Given that the number of classifiers in an ensemble is constant, the computational complexity is bounded by the complexity of the neural network, as given.

4.5.5 Discussion

This section first outlines the key hyperparameters that define the MicroNets framework, setting the stage for subsequent performance evaluations. We then present the results of MicroNets, initially examining their memory footprint to assess the practicality of deploying these ensembles on devices with constrained storage. Following this, we discuss the accuracy outcomes of the "Accuracy First" and "Diversity First" strategies, highlighting how these approaches influence ensemble performance. Finally, we consider the implications of MicroNets on the inference time and resource

usage on AIoT devices, specifically focusing on the Raspberry Pi, to gauge their efficiency in edge computing environments.

In MicroNets, the selection and tuning of hyperparameters play a pivotal role in the development and optimisation of the deep learning ensemble. The hyperparameters, such as sparsity levels (s), pruning steps (n), initial (s_i) and final sparsity (s_f), pruning frequency (Δt), and the number of clusters (k) for ensemble pruning, are critical as they directly influence the model's complexity, performance, and ultimately its deployment feasibility in resource-limited environments.

The pruning technique's hyperparameters, particularly the sparsity levels and pruning steps, dictate the degree and pace at which the model is simplified. A higher initial sparsity (s_i) may lead to a more aggressive reduction in complexity, potentially at the cost of accuracy. Conversely, a gradual increase in sparsity towards the final level (s_f) over numerous pruning steps (n) allows for a more controlled simplification, preserving the model's ability to generalise. The pruning frequency (Δt) further refines this process, enabling fine-tuning of the model's response to the pruning regimen.

The selection criteria hyperparameters, which include the accuracy and diversity-first strategies, determine the ensemble's composition. The accuracy-first strategy prioritises predictive performance, potentially leading to an ensemble skewed towards high-performing models that may, however, lack diversity. The diversity-first strategy, on the other hand, ensures a heterogeneous ensemble, enhancing the model's robustness to varied data inputs but possibly at the expense of individual model accuracy.

Moreover, the number of clusters (k) in the diversity-first strategy is a significant hyperparameter that affects the balance between diversity and accuracy. A higher number of clusters may increase diversity but can dilute the ensemble's overall predictive power if too many low-performing models are included. Conversely, fewer clusters might not capture the full spectrum of the data's characteristics, leading to a potential underfitting. The careful calibration of hyperparameters within MicroNets is reflected in the experimental outcomes, which we now turn to. Micronets employs a multi-phase pruning technique to achieve high compression ratios while still maintaining high accuracy levels through ensemble learning. The results of the experiments show that Micronets is capable of achieving compression ratios of up to 91%, leading to a reduction in the memory footprint of the ensembles created. In addition, the "Accuracy First" strategy outperforms the "Diversity First" strategy in both the CIFAR10 and CIFAR100 datasets. On the CIFAR10 dataset, the "Accuracy First" strategy generates ensembles that yield better results, with the highest accuracy achieved when the ensemble size is three, reaching 82%. In contrast, the "Diversity First" strategy produces ensembles with an accuracy of 79% when $k = 3$. When it comes to the CIFAR100 dataset, the "Accuracy First" approach produces models with higher accuracy than the baseline model. The highest accuracy is achieved by combining an ensemble of two models, with an accuracy of 55%. However, the use of the "Diversity First" strategy results in a significant drop in accuracy when using $k = 3, 5, 7$. The proposed "diversity first" approach does not appear to be very successful when applied to the CIFAR100 dataset, possibly due to the use of majority voting, which may not be effective for complex datasets such as CIFAR100 where the goal is to provide accurate prediction from a pool of 100 distinct labels. Therefore, further research is needed to explore more effective voting mechanisms for complex datasets.

The energy monitoring measurements detailed in section4.5.3 provide a nuanced

view of the power dynamics under different operational modes. Notably, the co-compilation mode demonstrates a clear advantage in energy efficiency when scaling to a larger ensemble of models, consuming less power with a load current of 0.42 DCA compared to the 0.44 DCA of the regular compilation mode for 100 models. This distinction becomes less pronounced with fewer models, where the regular mode tends to be more energy-efficient. Interestingly, when operating a single model, both modes exhibit an identical load current of 0.3 DCA, indicating a baseline power requirement for the Raspberry Pi devices equipped with an edge TPU accelerator. The input voltage remains consistently at 5.03 DCV across both modes, reinforcing the reliability of the power supply. These findings are particularly relevant in the context of resource-constrained environments, where energy efficiency translates directly into operational sustainability and cost-effectiveness. In such settings, the ability to run a larger number of models without a proportional increase in power consumption is invaluable. It implies that devices can perform more complex tasks or run for longer periods without necessitating additional power resources or frequent recharging, which is often a critical limitation in remote or mobile deployments.

Moreover, the co-compilation mode's energy savings when running extensive model ensembles suggest that it is possible to optimise system performance without linearly scaling energy consumption. This is crucial for edge computing applications where computational resources are limited, and power efficiency is paramount. By reducing the energy demand per model, the co-compilation mode enables a denser deployment of models, allowing for more sophisticated analytics and decision-making processes on edge devices.

Future work in this domain could expand on these initial insights by exploring the interplay between power consumption and other computational factors, such as the workload intensity and the processor's clock frequency. Understanding these relationships will further clarify how energy efficiency can be maximised in resource-constrained scenarios, ensuring that the computational potential of edge devices can be fully leveraged while still operating within the tight bounds of energy availability.

Additionally, in table 4.5, we find that the proposed approach operates smoothly on the Raspberry Pi device, which is resource-constrained, as the temperature and current load have only slightly changed. However, there is a slight delay in making inferences on Raspberry Pi devices, which makes it not ideal for real-time applications. Therefore, further investigation is necessary to evaluate the impact of using fine-tuned CNN architectures, such as MobileNets, which is designed specifically to run on resource-constrained environments. Using such networks could significantly reduce inference time and even model size.

Despite this limitation, Micronets demonstrates its feasibility for running deep learning ensembles on typical IoT devices, paving the way for a new generation of advanced AI applications and limitless possibilities that could benefit from the power of ensemble learning. For example, models generated by Micronets could be deployed on water distribution networks for contamination detection. The sensors in the network could collaborate to form deep learning ensembles that yield accurate decisions at the edge to contain water contamination in parts of the network rather than the entire water network. This approach could significantly improve the efficiency and accuracy of water distribution networks, leading to improved public health and environmental outcomes. Overall, Micronets represent a promising avenue for expanding the capabilities of AI applications in resource-constrained environments.

4.6 Summary

In this chapter, we presented *MicroNets* which aims to enable deep ensemble learning on AI edge endpoints, leading to improving ensemble classifiers in terms of generalisation, size, accuracy, and inference time. The experimental results on CIFAR10 and CIFAR100 were highly promising as the compression ratio was around 92%, and the composed ensembles could outperform the accuracy of baseline models. These findings suggest that *MicroNets* is a highly effective approach for running deep learning ensembles in resource-limited environments. Furthermore, we demonstrated the ability of *MicroNets* to preserve the resources of distributed Raspberry Pi devices while efficiently running ensemble learning.

However, we also found that the inference time may not be ideal for some time-critical applications that require near real-time predictions. This issue is primarily due to the fact that the architectures of these models were not optimised to run on resource-limited environments. Nonetheless, using fine-tuned CNN architectures such as MobileNets, which are explicitly designed to run on resource-constrained environments, could significantly improve inference time and even model size. Further investigations are necessary to evaluate the effect of using such networks.

Another potential area of improvement for *MicroNets* is its ensemble learning technique (max voting), which has room for improvement on the CIFAR100 dataset to enhance accuracy. While *MicroNets* demonstrated high accuracy on CIFAR10, it struggled with more complex datasets such as CIFAR100, indicating that more advanced voting mechanisms may be needed for such datasets. Further exploration is required to find more effective voting techniques that can work optimally with a more comprehensive range of datasets.

In summary, *MicroNets* represents a significant breakthrough in enabling deep ensemble learning on AI edge endpoints. The ability to run ensemble learning efficiently in resource-limited environments opens the door to new possibilities for advanced AI applications that could significantly improve public health, environmental outcomes, and other sectors. Overall, the results of this study show that *MicroNets* has the potential to revolutionise the field of deep ensemble learning and significantly impact the development of resource-efficient AI systems.

In the upcoming chapter, we will delve into the concept of *FedNets:* as a novel Federated Learning Strategy explicitly designed for Edge Devices. This cutting-edge approach utilises pruned ensembles of deep learning models, tailored to overcome the data privacy challenges inherent in AIoT devices.

By leveraging the potential of Federated Learning, FedNets empowers Edge Devices to collaborate and acquire knowledge collectively, eliminating the necessity for a centralized data repository. This methodology guarantees the privacy and security of sensitive data while facilitating the creation of robust machine learning models, even in scenarios where the edge devices exhibit varying distributions of class labels.

Moreover, the incorporation of pruned ensembles within FedNets facilitates the development of exceedingly efficient models that can seamlessly operate on resource-limited Edge Devices. Consequently, this diminishes the computational load on individual devices and paves the way for the establishment of more sustainable AIoT systems.

Overall, the adoption of FedNets represents a significant step forward in developing secure, accurate and efficient AIoT systems that can support a wide range of applications, from healthcare and finance to manufacturing and transportation.

Chapter 5

FedNets: Federated Learning on Edge Devices using Ensembles of Pruned Deep Neural Networks

In the previous chapter, we outlined a pioneering framework named *MicroNets: A multi-phase pruning pipeline to deep ensemble learning in IoT devices*, built upon the *Ensyth* approach expounded in 3. The purpose of *Micronets* was to provide a robust and resource-friendly structure suitable for resource-limited devices. Nonetheless, the practical deployment of AIoT applications imposes constraints on data privacy and security.

This chapter introduces a new paradigm, namely, *FedNets: Federated Learning on Edge Devices using Ensembles of Pruned Deep Neural Networks*, that aims to address the fourth and the fifth objectives of this thesis outlined in Chapter 1. The proposed approach describes an innovative, secure, and resource-efficient federated learning framework that operates deep learning ensembles on edge devices. To achieve this goal, *FedNets* leverages the groundwork laid out in Chapter 4, utilising graph theory to minimise communication overhead in federated learning and manage the statistical heterogeneity typical of real-world applications.

It is worth noting that the approach presented in this chapter has been published as a full paper titled "FedNets: Federated Learning on Edge Devices Using Ensembles of Pruned Deep Neural Networks" in IEEEAccess Journal, Volume11 [Alhalabi, Basurra, and Gaber, 2023].

5.1 Introduction

with the proliferation of the Internet of Things (IoT), and the launch of 5G networks, the IoT has emerged as one of the major technological advances in our lives. We can see their advances in different domains, including wearable smart health devices [Ghosh, Samanta, and Chakraborty, 2021], intelligent energy networks, smart transportation [Mohammadi and Al-Fuqaha, 2018] and smart building [Lui, Chan, and Leung, 2021], [Lui, Chan, and Leung, 2022]. These tiny connected devices generate massive amounts of data on the network edge, giving great opportunities to generate valuable insights and complete sophisticated machine learning (ML) tasks. The traditional approach to analysing IoT data is to transfer user data (clients) to a central cloud server. Then the server completes the analysis and generates the required insights [Ghosh and Grolinger, 2020]. However, moving sensitive information to a remote server can pose a significant risk to data privacy and lead to breaches of data protection laws such as GDPR (General Data Protection Regulation) [Voigt and Bussche, 2017]. Federated learning is a machine learning paradigm that enables the collaborative training of a model on data that is distributed across multiple devices or data centers without the need to transfer raw data to a central server [Yang et al., 2019]. In the standard federated learning setting, each device contributes an independent and identically distributed (IID) sample of data to the model. However,

most current FL strategies are not optimised to handle statistical heterogeneity, such as non-iid (non-independent and identically distributed) data generated by different clients [Wu, He, and Chen, 2020]. Since other clients are likely to exhibit different behaviour (distinct usage patterns), the local training samples may follow a different distribution. As a result, the local models will likely become vastly different; hence they could reduce the accuracy and lead to slow covariance [Zhao et al., 2018]. Furthermore, different studies raised some privacy concerns related to sharing weights and biases by models. Sharing the model updates during the training process make it vulnerable to penetration, potentially causing leaks of sensitive information [McMahan et al., 2018]. A number of authors have considered tackling non-iid challenges using clustering [Huang, Tiropanis, and Konstantinidis, 2022], [Li et al., 2023]. Clustering is a technique that can be used in non-IID federated learning to mitigate the impact of non-IID data distribution [Tian et al., 2022]. Clustering can group devices with similar data distributions together and allow local models to be trained on similar data; the resulting local models can then be combined to obtain a more accurate global model [Shu et al., 2023], [Morafah et al., 2022]. Personalisation in federated learning through clustering is further emphasised in recent academic works. Agarwal et al. discuss the role of personalisation in federated learning, which could be enhanced by clustering techniques to tailor models to specific user groups or devices [Agarwal, Yurochkin, and Sun, 2022]. Zhao et al. describe a two-phased federated learning approach that incorporates clustering and personalisation for more accurate predictions in applications like natural gas load forecasting [Zhao et al., 2023]. Additionally, Yan and Long explore the concept of personalisation disentanglement in federated learning, which could potentially be linked to clustering methods to separate personalised components from a shared global model [Yan and Long, 2023].

This chapter proposes a novel holistic approach to a federated learning strategy based on ensemble learning that improves accuracy and respects privacy at the edge end. This work is the first of its kind because it looks at federated learning from a different perspective. Unlike traditional federated learning strategies, our approach allows clients to collaborate by sharing complete models, not only weights. The proposed framework is an iterative process that begins by initialising a pool of pruned deep learning models (a global pool). These models are then randomly deployed to different clients to be trained on local datasets, taking into account any discrepancies in label distribution between the local and global datasets. Subsequently, the models' predictions are combined using ensemble learning to achieve a better generalisation performance on the local testing sets. The next step in the process is personalising the ensemble-based federated learning by clustering the models exhibiting similar behaviour. However, clustering DNNs on resource-constrained devices can be computationally expensive. To address this issue, we employ graph embeddings theory to reduce the complexity of the DNNs. Each ANN can be represented as a graph, with nodes representing layers and vertices representing connections between layers. We generate embeddings of all models and then cluster them. Finally, we select representatives of the resulting embedding clusters and ask the clients to share the corresponding models to be part of the global pool, thus initiating a new iteration.

Our key contributions could be summarised as follows:

- introducing a new federated learning strategy under non-iid settings; the proposed approach employs deep-ensemble learning to maximise the generalisation at the edge-end and provide better performance on different distributions of the clients' local data;

- proposing a novel ensemble pruning technique to reduce communication overheads over the network. It aims to minimise the storage footprint for ensembles by applying affinity propagation clustering. The clustering is applied to the embeddings of the models, considering that a graph could represent each artificial neural network, and
- establishing a new privacy approach to preserve clients' sensitive data in the applications that require running an ensemble of models.

The chapter is structured as follows: Section 5.2 provides a detailed description of the proposed approach, including formalization and algorithms of FedNets. In Section 5.3, we present the experimental setup, including details about the dataset, model, and server specifications. We also compare our accuracy results with the state-of-the-art federated learning strategies, and report other results related to the performance of ensembles and the privacy of clients. In Section 5.4, we conclude the chapter by highlighting the strengths and limitations of FedNets.

5.2 Method

In this section, we describe the proposed approach by providing a summary of the entire process and then explain each step in detail.

The proposed framework is an iterative process; it starts by initialising a pool of pruned deep learning models (a global pool θ_0). Then, the members of the proposed pool are randomly deployed to different clients to be trained on local datasets (assuming the label distribution of the local datasets does not match the global one). After that, the predictions of the models are combined using ensemble learning to obtain a better generalisation performance on the local testing sets. The next step in the process is to personalise the ensemble-based federated learning by clustering the models exhibiting similar behaviour. However, clustering DNNs on devices that are resource-constrained is expensive. Thus, we employ graph embeddings theory to reduce the complexity of the DNNs. Since each ANN (Artificial Neural Network) could be represented as a graph (nodes are layers, vertices are connections of layers), we generate embeddings of all models, and then we cluster them. Finally, we choose representatives of the resulting embedding clusters and then ask the clients to share the corresponding models to be part of the global pool θ_0 and a new iteration begins.

The following is a detailed description of the steps of *FedNets* and the overall system topology.

5.2.1 System Topology

As many federated learning strategies, *FetNets* follows the star network communication topology where a central cloud server is connected to a network of resource-limited devices. The server orchestrates the learning process by aggregating deep learning ensembles from all connected clients (each communication round). Figure **??** summarises the main communication topology.

5.2.2 Ensemble Generation and Pruning:

In non-iid federated learning settings, the statistical heterogeneity of clients can vary significantly [Wu, He, and Chen, 2020]. This led to a different distribution of the local data on each client [Zhu et al., 2021], hence traditional federated learning settings experience a drop in accuracy [Konečný et al., 2017]. To overcome those challenges, our

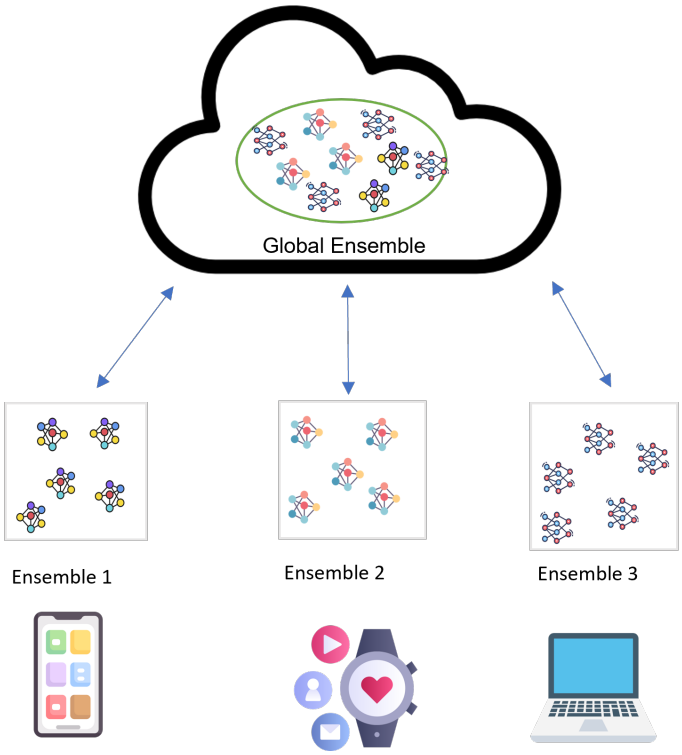

FIGURE 5.1: *FedNets* follows a star communication topology where a server connects with all the remote clients. The server orchestrates the communications in each learning round, it starts by deploying the global ensemble to the clients. Then, each client shares a few members of their ensembles with the server.

approach utilises deep ensemble learning to give more generalisation power at the edge end and boost performance. Typically, any pruning process has different hyperparameters, and it could be challenging to set the optimal values of these parameters. Thus, different optimisation techniques could be used [Ahmed et al., 2022], [Yuan et al., 2022b], [Zhao, Wang, and Mirjalili, 2022], [Yuan et al., 2022a]. However, due to the high complexity of DNNs, we aim to prune the models before running them at the edge end. To complete the pruning and the deployment of models, we follow a similar approach to the one described here [Alhalabi, Gaber, and Basura, 2021]. In this work, the authors proposed a multi-phase pruning framework that enables edge devices to run deep ensemble learning without draining the resources of devices. However, our approach has two main differences: (1)In our approach, we use Constant Sparsity Pruning [*tfmot.sparsity.keras.ConstantSparsity | TensorFlow Model Optimization* n.d.] as a preferred alternative to weight pruning. This method, prevalent in machine learning, systematically mitigates model complexity by cyclically eliminating the least consequential parameters, thereby preserving a consistent degree of sparsity throughout the progression of the training phase, and (2) integer quantisation is ignored as we do not run the experiments on real devices. At the end of this step, we have a group of N clients; each client has a group of M different models.

5.2.3 Model training on Local Datasets (NON-IIDs) Update Local Models Weights

As all federated learning strategies, *FedNets* requires the local models (ensemble members) to be trained on the datasets of each client. In the training process, the models will simply learn the good values for all the weights and the bias from the labelled examples. However, *FedNets* employs deep learning ensembles, thus the training accuracy will depend on all members. To calculate the accuracy of an ensemble: Let $En = m_1, \cdots, m_N$ be an ensemble of the deployed models m_i. The prediction of the ensemble for a training/testing example (x is the data feature, y is the data label) using max-voting is the class that receives the maximum support $\eta_{final}(W)$ from all members of the ensemble. Based on that, the output of the ensemble could be defined as:

$$\eta_{final}(W) = argmax_{j \in \{1,\cdots,C\}} \sum_{i=1}^{N} y_{i,j}$$

5.2.4 Graph Conversion

Graph clustering based on embedding is a popular technique that aims to convert graph structure and node attributes into the low dimensional feature, and split similar nodes into non-overlapping groups [Xie, Girshick, and Farhadi, 2016], [Guo and Dai, 2022]. The purpose of this step is to cluster models with similar properties using the graph topology and node features. To convert an artificial neural network to a graph, let us have a k layer neural network, each layer has a set of weights $W = w_1, w_2, \cdots, w_n$, the output of layer k is $z = W \times x + b$.

A graph $G = (V, E)$ is defined as a collection of vertices $V = \{v_1, \ldots, v_n\}$ where $v_i \in En$, edges $E = \{e_{ij}\}_{i,j=1}^{n}$ where e_i is a connection between l_i and l_{i+1} assuming that k is a layer in m_i, and $X_i = \{(w_1, b_1), \cdots, (w_n, b_n)\}$ is the corresponding node vector that holds the average weights $\sum_{i=1}^{k} avg(W_k)$ and biases $Avg(\sum_{i=1}^{k}(b))$ in each layer k.

Figure 5.2 shows how *FedNets* converts an artificial neural network into a graph.

5.2.5 Graph Embedding Generation

The representation of a node v_i according to **[bai_unsupervised_2019]** can be calculated as :

$$\mu_i^k = MLP_{W_k}^k \left((1 + \epsilon^k) \cdot \mu_i^{k-1} + \sum_{j \in N(i)} \mu_j^{k-1} \right) \tag{5.1}$$

where μ_i is the representation of node v_i, $N(i)$ is the neighbourhood of node V_i, ϵ could be learnt by a hyperparameter or GD (gradient descent) and $MLP_{W_k}^k$ refers to multilayer perceptron for the k^{th} GIN layer and weights W_k. After generating the node embeddings, we calculate the overall graph embeddings following the same approach described in [Bai et al., 2019]:

$$\boldsymbol{h}_{\mathcal{G}} = \mathrm{MLP}_{\boldsymbol{W}} \left(\|_{k=1}^{K} \mathrm{ATT}_{\boldsymbol{\Theta}(k)} \left(\boldsymbol{U}_{\mathcal{G}} \right) \right) \tag{5.2}$$

The embedding of the input graph of graph g is $U_{\mathcal{G}} \in \mathbb{R}^{N \times D}$ where the $n - th$ row, $u_n \in \mathbb{R}^D$. The output of this step is an array of the corresponding graph embeddings for each model Em_i.

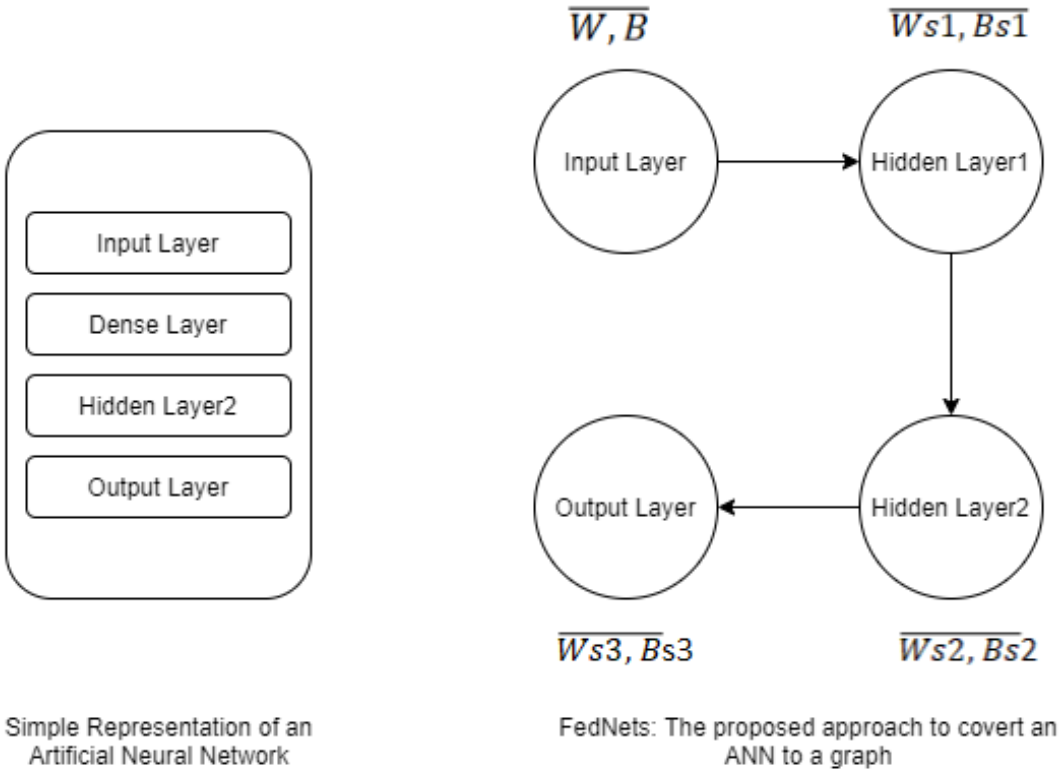

FIGURE 5.2: *FedNets* approach to convert an ANN to a graph. Each layer will be converted to a node, and the mean value of the weights/biases will be added as a feature to the node

5.2.6 Clustering of Embeddings

The clustering of embeddings is a critical step in the FedNets approach, which aims to identify and utilize the most representative models for federated learning tasks. After training, each model m_i within the initial set θ_0 is transformed into a two-dimensional array Em_i, which serves as a compact representation of the model's learned features or "embeddings". These embeddings capture the essence of the data patterns that each model has learned.

To effectively group similar models, we employ affinity clustering on the set of embeddings $\{Em_i\}$. Affinity clustering is a technique that identifies clusters based on the similarity of data points, which, in our case, are the embeddings of the models. This method does not require the number of clusters to be specified a priori, which is advantageous in federated learning scenarios where the optimal number of clusters is not known in advance.

Selection Criteria for Representative Models

Once the clusters $\{C_i\}$ are formed, we proceed to select representative models from each cluster. The selection is governed by two main criteria:

1. **Cluster Size**: A cluster C_i must have a size greater than a threshold value K. This threshold is defined as a range from 1 to 20, $K = \{1, \cdots, 20\}$. The rationale behind this criterion is to ensure that the chosen cluster has a sufficient number of models to be considered statistically significant and to avoid overfitting to particularities of the training data.

2. **Model Accuracy**: The accuracy acc_i of a model m_i on a validation set VS must exceed 55%. The validation set VS is a subset of the training data, comprising 10% of it, which is held out specifically for this validation purpose. This accuracy threshold ensures that only models with a reasonable baseline performance are considered for selection.

The equation of the representation selection is provided within the formula:

$$\begin{cases} m_i \in C_i \text{ where : } cl > K \quad \text{where} \quad K = \{1, \cdots, 20\} \\ acc_i > 55\% \text{ on} \quad \mathbf{VS} \end{cases} \tag{5.3}$$

The pseudocode of *FedNets* is shown in Algorithm 8. This pseudocode provides a step-by-step procedural guide to the implementation of the FedNets approach. It outlines the sequence of operations that are performed during the federated learning process, including the initialization of parameters, the distribution of models to clients, the local training on clients' devices, the transformation of models into embeddings, the clustering process, and the selection of representative models.

Algorithm 8 *FedNets* pseudocode

Server
Initialise global pool θ_0
for Each communication Round R, r $\leftarrow$ 1 to T **do**
  Select N Client
  for Each client $i = 1, 2, .., N$ **do**
    Download θ_r
    $Client_i$ update and receive em_i
  end for
  $GE \leftarrow AffinityPropagation(EMs)$
  $M \leftarrow RepresentativeSelection(GE)$
  Update Global Pool θ_0 with M
end for

Client Update
Replace local ensemble $\theta_i \leftarrow \theta_{i+1}$
for Local Epoch $e \leftarrow 1$ to E **do**

$$\eta_{final}(W) = argmax_{j \in \{1, \cdots, C\}} \sum_{i=1}^{N} y_{i,j}$$

end for
for Each model M in θ_i **do**
  $em \leftarrow generatedEmbeddings(M)$ return em
end for

5.2.7 Privacy Preserving

Unlike the typical federated learning approaches, the proposed method provides a privacy-preserving-by-design approach. The sharing of a subset of the local ensemble per client makes it harder to reveal the behaviour of the users of individual

clients. Formally, if the number of models making up the ensemble in a client is n, only m models are shared, where $m < n$. Thus, privacy increases when the value $\tau = \frac{m}{n}$ decreases. Note that τ can be a hyperparameter when applying the proposed method. An important factor in increasing privacy is the diversity of the ensemble. The more diverse the ensemble is the more private the proposed method is. In other words, with the same value for τ, the diverse ensemble is inherently more private.

For example, if $n = 10$ and $m = 3$, then $\tau = 0.3$. This is a setting that will result in high privacy. However, if $\tau = 0.9$, then privacy may be compromised, because most of the models that make up the ensemble are shared centrally, making it possible to reproduce the data fed to the model locally. It is worth pointing out that in a typical federated learning setting, the parameters of the model are shared centrally (i.e. $\tau = 1$), making the models shared by the clients vulnerable.

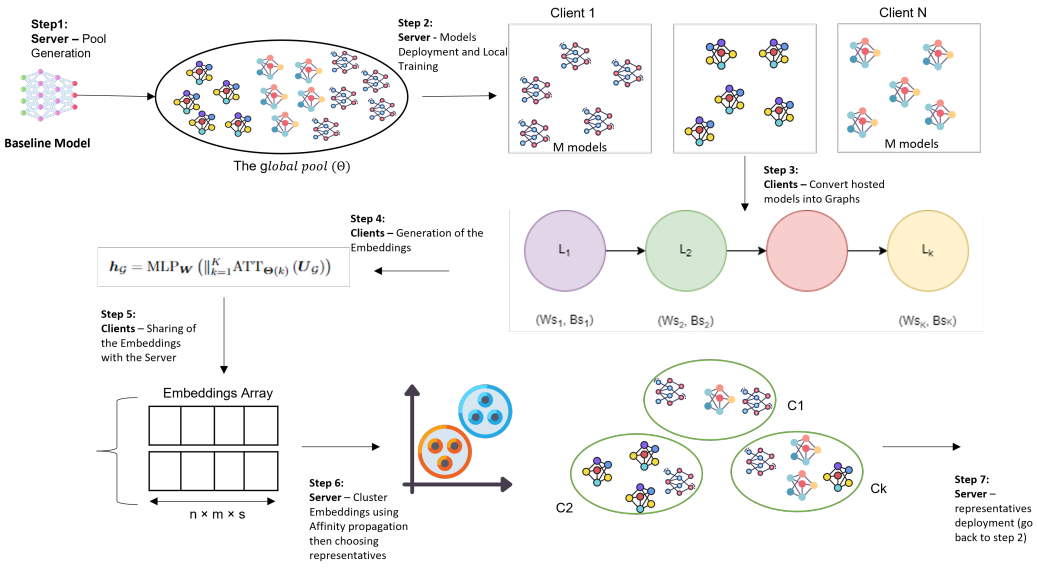

FIGURE 5.3: *FedNets Framework involves the following steps:- Step1: server deploys deep learning ensembles to the clients and clients train them locally; step2: Each client converts the local models into graphs assuming: nodes are the layers, edges are links between layers and the attributes of the nodes are mean of weights and biases of whole layers; step3: clients generate the corresponding graph embeddings and share them with the server; step4: the server to cluster the embeddings using affinity propagation and choose a representative from each cluster, then models are deployed to clients.*

5.3 Experimental study

In this section, we first explain in detail our simulation environment and setup, then evaluate the precision of *Fed-Nets* using the ResNetV2 and CIFAR100 federated dataset. Next, we compare our results with *Fed-Avg*. Finally, we provide important measures related to ensemble performance, such as inference time, required training time, and the number of models per client. Those measures are essential to give an idea about the feasibility of running deep ensembles on resource-limited devices (IoT). The code used in the experiment is publicly available on GitHub[1].

[1] https://github.com/besherh/FedNets

5.3.1 Data set and Models

Federated CIFAR100 for Simulation

This dataset is specially designed to simulate non-independent and identically distributed data samples. It is derived from the original CIFAR100 dataset, and it has 50,000 training samples and 10,000 testing samples. Unlike the original dataset, the training and testing samples are partitioned across 500 and 100 clients (respectively, and no overlapping across the clients). The training clients' IDs range from 0 to 499, while the testing clients' IDs go from 0 to 99. The data partitioning part is done using PAM (Pachinko Allocation Method) [Li and McCallum, 2006], which is an improved version of LDA (Latent Dirichlet Allocation). This approach uses a two-stage LDA process, where each client has an associated multinomial distribution over the coarse labels of CIFAR-100, and a coarse-to-fine label multinomial distribution for that coarse label over the labels under that coarse label.

ResNetV2

This model belongs to the Deep Residual Networks (RNNs) family that achieved breakthroughs in the deep learning community. ResNetV2 is the new version of ResNet, the main improvements are related to the arrangement of the layers in residual blocks. The Model accepts an input of shape 299×299 pixels; the output is the probability distribution for the predicted classes [He et al., 2016b].

5.3.2 Simulation Setup

We run our simulation on a powerful workstation with multiple GPU and CPU nodes. The following are the hardware specifications:

- CPU Nodes:5 nodes - 72 cores per node, PowerEdge R740 Server, Intel Xeon Gold 6240 2.6G.
- GPU Nodes:2 nodes - 72 cores per node, PowerEdge R740 Server, Intel Xeon Gold 6240 2.6G, NVIDIA(R) Tesla(TM) T4 16GB Passive, Single Slot, Full Height GPU (2 cards per node) - 320 Turing Tensor cores and 2560 Cuda cores per card.

All nodes have 'CentOS-8.2.2004-x86_64' operating system installed. There are also different hyperparameters that control the design of FedNets, starting from initialising the global pool and ending with the deployment of the models to the clients. Table 5.1 summarises all hyperparameters that are used in the different processes of FedNets.

As shown in Table 5.1, there are a lot of different hyperparameters that control the process flow of FedNets. Starting from the generation of the global pool and ending with representative selection. The values of the hyperparameters are selected based on a "trial and error" approach, and we only reported the values related to the presented results.

TABLE 5.1: FedNets Hyperparameters

Phase	Hyperparameter name	Values
Training and Pruning	Epochs	[2,3,4,5,6]
	Batch Size	[16,32,64]
	Loss	[Categorical Cross Entropy, Mean Squared Error, Mean Absolute Error]
	Optimiser	Adam
	Target Sparsity	[0.2,0.55]
	Frequency	[50,75,100]
Embeddings	Graph Classification Model-Layer Size	[32,64]
	Graph Classification Model- Activations	ReLU
	Graph Classification Model- Dropout	0
	Graph IDX - Size	(100, 2)
	Pair Model - optimiser	Adam((1e-2)
	Pair Model - Loss	MSE
Clustering	Validation Ratio	0.1
	Accuracy Threshold	0.3
	Cluster Length	10
	Sample Size	60

5.3.3 Results and Analysis

We start this section by introducing some details about the baseline model and the pool of models we used to deploy deep learning ensembles to clients. We next investigate the effectiveness of *FedNets* on non-iid settings. We use the federated CIFAR100 dataset as a benchmarking dataset and ResNetV2 as a baseline model. Additionally, we provide accuracy comparison with state-of-the-art federated learning algorithms including Fed-Avg and Fed-Yogi. The simulation is applied to a different number of clients (two clients, five clients and ten clients) for four federated learning rounds, where each client has an ensemble of ten pruned models. Finally, we provide time measures related to the performance of the deep learning ensembles on the proposed virtual clients.

The baseline of ResNetV2 (3,575 KB) is trained on the original CIFAR100 dataset and the accuracy of the model on the testing set is 68%. We apply constant-sparsity pruning on ResNetV2 to generate a pool of 500 models. During the pruning process, we use different values of the pruning hyperparameters to ensure diversity between the pool's members. Diversity leads to better generalisation and provides better accuracy results at the edge end as shown in [Alhalabi, Gaber, and Basurra, 2019]. The maximum accuracy in the pruned pool against a validation set (20% of the testing set) is around 66% and the minimum is 0.05%. The pruning lead to around 37% reduction of the original baseline model size(the average size of the pruned models is 1,295 KB).

Comparison With the State of the Art

Here we compare the accuracy results of *FedNets* with two of the state of the federated learning strategies (FedAvg, FedYogi) on the federated CIFAR100 dataset. In the next section, we provide the simulation results for different numbers of clients (two, five, and ten clients). As shown in Figure 5.4 and when the number of clients

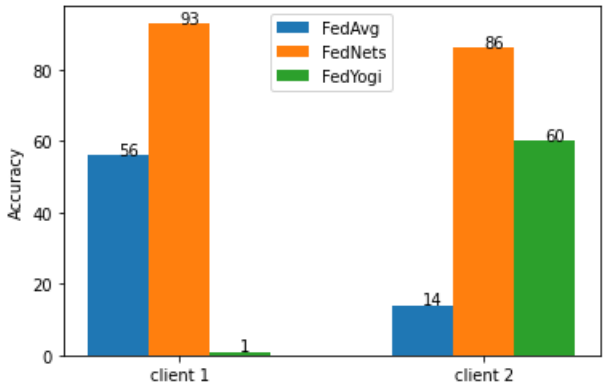

FIGURE 5.4: Accuracy comparison on two clients

is equal to two, *FedNets* accuracy is 93% and 86% in client1 and client2, respectively. On the other hand, the accuracy of FedAvg is 56% on client1 and 14% on client2. Similarly, the accuracy of FedYogi on client1 and client2 is 1% and 60%.

The results of the accuracy of *FedNets* on five different virtual clients are presented in Figure 5.5. The chart shows that *FedNet*'s accuracy is better than FedAvg and FedYogi on all clients. The minimum accuracy for *FedNets* is 90% while the best accuracy reached by FedNets is 80%.

Figure 5.6 compares the accuracy results between *FedNets*, FedAvg and FedYogi. Looking at Figure 5.6, it's apparent that *FedNets* still achieves superior performance

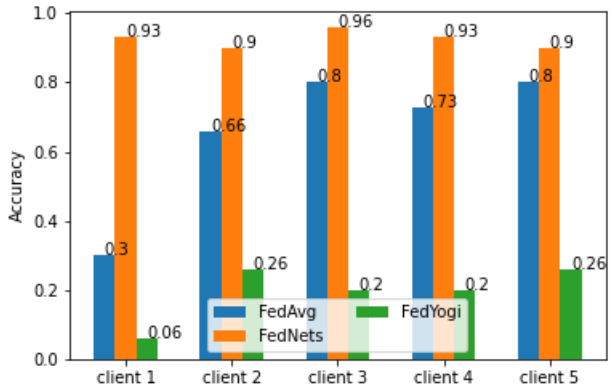

FIGURE 5.5: Accuracy comparison on five clients

.

and can beat the state-of-the-art methods on all clients. We can also see that the maximum accuracy of *FedNets* is 100% on the client3, the maximum accuracy of FedAvg is 80% on client7 and client 8, and the best accuracy of FedYogi is 53% on client10. As shown in Figures 5.4,5.5 and 5.6, there is a significant performance difference

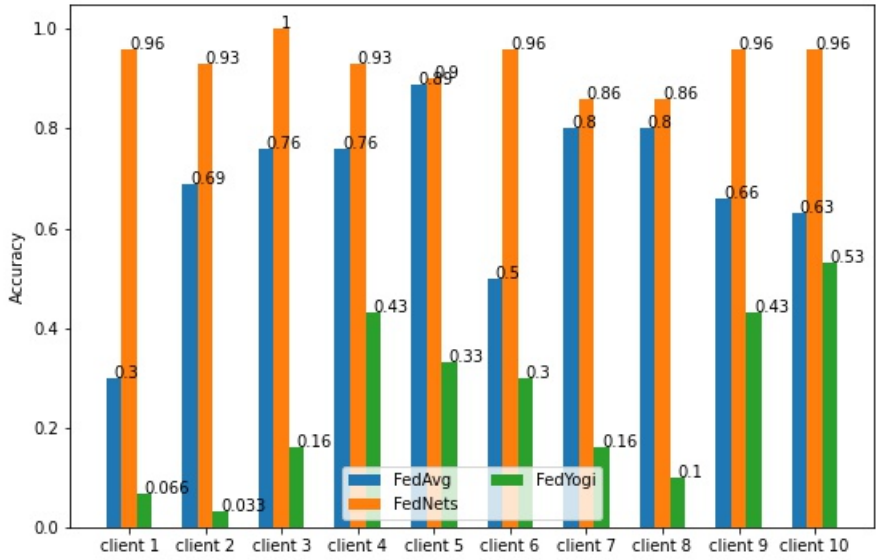

FIGURE 5.6: Accuracy comparison on ten clients

.

between *FedNets* and FedAvg/FedYogi. Our proposed approach provides better accuracy results on a different number of clients; the clients can share their knowledge by exchanging the members of the ensembles, and it seems to be better than the traditional approach of sharing the weights of the models.

Ensembles Performance

In the section above, we present the accuracy results for the federated CIFAR100 dataset. Now, we move to present the simulation results of the required time to complete one federated learning round in*FedNets*. Each round consists of three main steps: training, inferencing, and deployment. We run this simulation on CPU nodes for four federated learning rounds. The simulation results are presented in Table 5.2.

TABLE 5.2: Time required by the major steps of *FedNets* in seconds.

Measure	Round1	Round2	Round3	Round4
	2 Clients			
training locally	78.03	61	54.37	43.39
ensemble inference	25.36	19.95	19.22	14.27
deployment	64.87	62.37	62.03	60.16
	5 Clients			
training locally	236.69	228.82	256.42	222.21
ensemble inference	74.79	77.37	75.84	72.18
deployment	82.62	93.04	80.74	81.35
	10 Clients			
training locally	681.45	664.27	663.02	663.76
ensemble inference	186.63	175.94	178.53	117.06
deployment	120	121.18	116.41	116.38

On closer inspection of Table 5.2, it shows that training the deep learning ensembles on the local datasets is consuming most of the time. Training time is noticeably increased when the number of clients is ten (almost 12 minutes to complete). However, the inference time of the ensembles is acceptable in most of the applications.

It can also be seen from the data in Table 5.2 that, in general, *FedNets* requires a relatively long period of time to complete a federated learning round (especially when the number of clients is rather considerable, like ten). However, we should accept the fact that *FedNets* is an ensemble-based approach that aims to maximise the generalisation and accuracy at the edge end, so it requires more time to complete a federated learning round. Additionally, we are running the simulation on CPU nodes only. In a real-life application, resource-limited devices could be attached to cutting-edge AI accelerators that bring the power of TPUs to the edge. We believe that this led to a significant improvement in the time complexity of *FedNets* as shown in [Alhalabi, Gaber, and Basura, 2021]. Turning now to examine the size of the composed ensembles on each client after each federated learning round. This could be directly related to the effectiveness of our approach in preserving the resources of IoT devices. Table 5.3 displays the changes in the number of models per client (ensemble size) after each federated learning round.

As shown in Table 5.3, *FedNets* reduces the size of the ensemble by 50% after round 4 (assuming that each client starts with ten models, as explained earlier). The simulation on both five and ten clients shows that *FedNets* is still able to reduce the number of models per ensemble which lead to reducing the required memory/storage required by the approach.

TABLE 5.3: Changes to the number of models per client

Number of models	Round1	Round2	Round3	Round4
	2 Clients			
Client1	7	8	7	6
Client2	6	7	7	5
	5 Clients			
Client1	8	10	7	9
Client2	5	5	9	9
Client3	9	9	9	8
Client4	7	9	7	9
Client5	7	9	7	5
	10 Clients			
Client1	10	10	8	9
Client2	9	10	9	7
Client3	10	9	10	8
Client4	9	10	8	8
Client5	10	10	9	10
Client6	10	9	10	8
Client7	9	10	10	9
Client8	9	10	10	9
Client9	10	9	9	10
Client10	10	10	8	10

Preserving Privacy

As previously stated, *FedNets* prompts federated learning by allowing clients to share the members of the local ensembles; at the same time, *FedNets* respects the privacy of the clients.

Table 5.4 shows summary statistics about the number of shared models per client. It is worth mentioning that the value of τ is not controlled during the simulation. However, as explained earlier, τ could be a hyperparameter to the proposed approach which will be used to trade off accuracy with privacy by controlling the maximum number of models to be shared with the other clients. As seen in the table, when the number of models is equal to ten, *FedNets* tends to share a large number of models (average 9.6, $\tau = 0.96$). This could be minimised by defining $\tau \leq 0.5$ sharing less than half of the models per client, forcing greater privacy. However, this may come at the cost of compromising the accuracy of the federation.

Discussion of the Results

The results of FedNets as shown in Figures 5.4, 5.5, 5.6 indicate that the proposed approach is effective in providing high accuracy in non-iid settings where the distribution of class labels is vastly different among clients. This will work well for applications that cannot compromise on accuracy as the quality of the output directly impacts the reliability of the AI systems. For example, FedNets could be integrated into AI-Based Medical Diagnosis systems to offer further data privacy assurance to comply with Health Institutions' Data Protection Policies. The reasons behind the superior performance of FedNets can be attributed to several key factors:

- Ensemble of Diverse-Lightweight Models: Unlike FedAvg and FedYogi, which share model parameters over the network to update a single global model, FedNets takes a fundamentally different approach. It allows clients to have ensembles of diverse, lightweight models. This diversity enables FedNets to capture a broader range of patterns and adapt more effectively to various data distributions present in non-iid datasets like Federated CIFAR100.

- Graph Embedding Theory: FedNets leverages graph embedding theory to reduce the computational complexity of running Deep Neural Networks (DNNs) on resource-limited IoT devices. In FedNets, each DNN is treated as a graph, from which graph embeddings are generated. This innovative approach optimises the utilisation of local device resources, making it well-suited for IoT environments where computational capacity is limited.

- Dynamic Sharing of Model Components: FedNets intelligently determines which parts of the DNN should be shared with other clients based on the clustering of graph embeddings. This dynamic sharing mechanism allows clients to collaborate efficiently while minimising redundant transmissions. In contrast, FedAvg and FedYogi share entire model parameters, which can lead to excessive communication overhead.

It is possible that the results in Table 5.4 could be improved by adding the privacy factor τ into the list of hyperparameters introduced in Table 5.1. This could be useful when edge clients deal with very sensitive personal information, for example, smart wearable devices. The results obtained from Table 5.2 shows that FedNets could run on resource-limited environments effectively. However, the required time to complete one federated round could take a reasonably long time. The observed increase in time could be attributed to: a) using a baseline model that has not been fully optimised to run on resource-limited devices; and b) utilising deep ensemble learning to provide better generalisation in non-iid settings. Using edge-friendly models like the MobilesNet family [Sandler et al., 2018] could lead to shorter training and inference time. However, the fact that FedNets requires a relatively long period to complete a federated round will still be valid. This limitation renders FedNets a less desirable choice when there are no GPUs attached to the edge devices (operating solely on CPU) and when real-time applications necessitate instantaneous inference, such as autonomous vehicles.

TABLE 5.4: Statistics related to the numbers of shared models per round

Metric	Round1	Round2	Round3	Round4
	5 Clients			
Minimum	5	5	7	6
Standard Deviation	1.48	2.07	1.14	1.30
Variance	1.48	2.07	1.14	1.30
Average	7.2	8.6	8.4	8.2
	10 Clients			
Minimum	7	6	8	7
Standard Deviation	0.51	0.48	0.87	1.03
Variance	0.26	0.23	0.76	1.06
Average	9.6	9.7	9.1	8.8

5.4 Summary

In this chapter, we introduced a novel ensemble-based federated learning strategy called *FedNets*, which addresses the challenges of training deep neural networks (DNNs) in non-iid settings while preserving the privacy of client devices. Unlike other federated learning strategies, *FedNets* allows clients to run deep learning ensembles instead of having one model per client. This enables clients to explore a larger hypothesis space and achieve better generalization performance, especially in non-iid settings where the data distribution across clients is different. Moreover, instead of sharing the models' weights to update a single global model, which is prone to a privacy breach, our approach allows clients to share models (members of their deep learning ensembles) to compose a shared pool of outperforming models, then the pool is shared with all participating clients.

The experimental results on the federated CIFAR100 dataset demonstrate the effectiveness of our approach. We show that *FedNets* outperforms two state-of-the-art federated learning strategies, namely Federated Learning Averaging (FedAv) and Adaptive Federated Optimization (FedYogi), in terms of both accuracy and convergence speed. We also demonstrate that the cost of running deep learning ensembles on resource-limited edge devices is feasible, as long as appropriate architectures and hardware accelerators are used. However, we note that the inference time of the generated ensembles may be relatively high for applications that require instant inferencing, especially when the number of clients is large.

To address this limitation, we suggest the use of GPU accelerators and fine-tuned CNN architectures such as MobileNets, which have been shown to reduce inference time while maintaining high accuracy significantly. We also highlight the potential of *FedNets* for a new generation of AI applications that can leverage the power of deep ensemble learning to provide more generalization power on resource-limited devices without compromising privacy.

In conclusion, *FedNets* is an additional contribution to the field of federated learning, providing a scalable and privacy-preserving approach for training DNNs in non-iid settings. Our approach demonstrates the potential of deep learning ensembles to achieve better generalization performance while maintaining low resource consumption. We expect that *FedNets* will inspire further research in the area of ensemble-based federated learning and enable new AI applications that can benefit from the power of deep learning.

Chapter 6

Conclusion and Perspectives

6.1 Summary

The research presented in this thesis focuses on the integration of artificial intelligence (AI) and the Internet of Things (IoT) to enhance the capabilities of intelligent devices operating at the edge. The core aim of this thesis was to develop effective and privacy-preserving computer systems for deploying AI applications on the Internet of Things (IoT) devices. This aim was pursued through four key objectives.

Firstly, a critical review of the literature on IoT, pruning techniques, deep learning ensembles, federated learning, and their applications in IoT was conducted to identify the gaps in the existing literature. This served as the basis for the novel contributions that followed.

Secondly, a novel approach, the Ensyth method, was designed and developed for synthesising pruned ensembles of deep learning models. This approach aimed to make better use of IoT resources and to achieve high predictability levels, forming the core around which the other layers of our model are arranged.

The third objective led to the development of the Micronets approach, positioned in the second layer of the model. This approach involved the design and development of a multi-phase pruning pipeline incorporating weight pruning, channel pruning, and knowledge distillation. It enabled efficient deep ensemble learning on IoT devices and served as a lever for the Ensyth approach.

Lastly, the outermost layer of the model, the FetNets approach, was developed to enable edge devices to collaboratively train ensembles of pruned deep neural networks. This was aimed at tackling the statistical heterogeneity in federated learning settings while preserving data privacy, thus encapsulating the work carried out in both the Ensyth and Micronets layers.

All the proposed approaches were evaluated on benchmark datasets and their performance was compared with state-of-the-art methods in terms of accuracy, efficiency, and privacy (our fifth objective). Thus, it can be concluded that this thesis has successfully addressed its objectives and main aim. The developed model reflects a comprehensive, multi-tiered structure akin to an onion graph model, symbolising that each segment of the model is constructed upon the foundation of the preceding one.

Following this schematic representation of our thesis's core constructs and contributions, we delve into a comprehensive summary of the individual thesis chapters in the ensuing section.

Chapter 2 as outlined in Section [2], provides an exhaustive and enlightening overview of Artificial Intelligent Things (AIoT). The chapter meticulously details the various components that go into constructing AIoT applications, beginning with the topology of IoT devices and their applications, and culminating with an examination of the challenges of implementing AIoT applications in real-world scenarios.

The chapter also delves into the intricate details of the various AI algorithms that can be employed to build intelligent things. These include supervised learning, which involves training a model on labelled data to predict outcomes, unsupervised learning, which entails training a model on unlabelled data to identify patterns and structures, deep learning, a subfield of machine learning that utilizes neural networks to learn from data, and ensemble learning, which combines multiple models to improve accuracy.

Additionally, Chapter 2 comprehensively examines federated learning, a distributed learning approach that enables edge devices to collaboratively train a shared model without divulging raw data. The challenges and opportunities associated with these algorithms are discussed in detail, providing a comprehensive understanding of their capabilities and limitations.

Furthermore, the chapter identifies the primary gaps in the AIoT field and presents the author's contribution to filling these gaps. The author's contribution comprises innovative approaches that enhance the predictability and generalisation power of deep neural networks in compressed models.

Chapter 3 in this chapter discussed in Section [3], we introduce our first novel approach, named "Ensyth," which represents our initial contribution towards achieving the second and fifth objectives set forth in this thesis. By harnessing the power of deep ensemble learning, Ensyth enhances the predictability of pruned models while simultaneously improving the generalization capabilities of deep learning models operating at the edge.

Moreover, this chapter presents the analysis of the results obtained from subjecting Ensyth to rigorous evaluations across several benchmarking datasets. The results provide compelling evidence of the efficacy and versatility of our proposed approach.

Chapter 4 as discussed in Section [4], introduces our second novel approach, called "MicroNets," which addresses the third and fifth objectives of this thesis. This chapter presents a multiphase pruning pipeline that enables the deployment of resource-efficient deep ensemble learning on devices with constrained resources. Furthermore, it proposes a novel clustering-based pruning method for deep-learning ensembles, providing a mathematical formulation of the method and detailed analysis on popular benchmark datasets.

The experiments conducted to evaluate the effectiveness of MicroNets were performed on Raspberry Pi devices, and the results demonstrate its ability to preserve the resources of the IoT (PI) device effectively. The analysis includes various metrics such as accuracy, compression ratios, inference time, voltage, load current, and temperature, providing a comprehensive understanding of the effectiveness of MicroNets.

Chapter5 Chapter 5, as discussed in Section [5], represents the culmination of our efforts towards achieving the fourth and fifth objectives of the thesis. This chapter introduces a new approach called *FedNets*, which is a trailblazing concept in federated learning. Unlike conventional methods that involve sharing model weights, our approach facilitates participating edge clients to share members of their hosted ensembles with a privacy-centric design.

Notably, our proposed technique outperforms state-of-the-art federated learning approaches, including *FedAvg* and *FedYogi*. The chapter presents a comprehensive explanation of the proposed algorithm and all experimental details related to the used hardware, the number of participating clients in each federated learning round, and various metrics against benchmarking federated datasets.

The chapter provides a detailed analysis of the results obtained from subjecting our proposed approach to rigorous evaluations across various benchmarking

datasets. These results provide compelling evidence of the efficacy and versatility of FedNets, highlighting its superior performance compared to state-of-the-art federated learning methods.

6.2 Future Direction

The methodologies outlined in this dissertation unveil promising avenues for continued exploration. Balancing rigorous theoretical analysis with hands-on empirical experimentation is essential to unlock their full potential. Herein, we will delve into the distinct research directions stemming from each of our contributions and provide insights into why Federated Learning is central to our research motivation.

6.2.1 Ensyth: Delving Deeper into Ensemble Strategies

The Ensyth approach, as delineated in our studies, offers intriguing results and potentials that beckon deeper exploration. Here are key findings and future avenues:

- Performance Analysis: The Ensyth method has shown promising accuracy on the CIFAR5. However, its performance on the CIFAR10 was only slightly better than the baseline. This prompts further examination, especially on more complex datasets such as CIFAR100 and ImageNet.
- Revisiting Ensemble Techniques: The current use of the Majority Voting technique in the Ensyth approach might inadvertently favor dominant models, overshadowing insights from less frequent predictions. Given this, there's a need to critically evaluate and potentially refine this ensemble strategy, especially in tackling intricate, non-linear problems with high noise levels [Dietterich, 2000].
- Diversification through Pruned Models: A promising direction for Ensyth involves maintaining a collection of diverse, pruned models. Instead of using identical base models in traditional ensemble methods like bagging and boosting, these base models can be derived from the Ensyth process. This adaptation could promote greater model diversity. A further future direction involves investigating the relationship between diversity and the accuracy of pruned models. To this end, employing measures such as entropy [Kuncheva and Whitaker, 2003], and variance [Melville and Mooney, 2005] can offer valuable insights into the ensemble's comprehensive representation.

6.2.2 Micronets: Deep Ensemble Learning in IoT Devices

MicroNets, described as "The Multi-Phase Pruning Pipeline to Deep Ensemble Learning in IoT Devices," has showcased impressive outcomes in both accuracy and the conservation of resources on IoT devices, notably heat, load current, and voltage. A prominent challenge, however, is the observable latency during inference operations on Raspberry Pi devices. To address this, we propose a structured roadmap for future research:

- Resource-Efficient CNN Models: Investigate the adoption of lightweight CNN architectures, specifically models like MobileNets. The goal is to determine whether such architectures can enhance inference speeds on IoT devices, especially on platforms like Raspberry Pi.

- Ensemble Learning Techniques: Reevaluate the current ensemble learning method, which employs max voting, by exploring alternative advanced methods. Techniques like bagging and boosting should be considered to assess if they can alleviate the delay arising from inference tasks involving multiple models. The overarching objective is to optimize MicroNets' performance.

- Pruning Techniques Exploration:it is crucial to explore the impact of employing different pruning techniques for both the generated models and ensembles. This investigation aims to further reduce the size of the generated models and decrease the memory footprint of the ensembles when they run on IoT devices..

By undertaking these recommended experiments, we can gain valuable insights into optimising MicroNets for IoT devices and potentially overcome the delay issue experienced on Raspberry Pi devices.

6.2.3 FedNets: Embracing Federated Learning for Enhanced Edge Computing

Our research introduces an innovative approach known as FedNets: "Federated Learning on Edge Devices using Ensembles of Pruned Deep Neural Networks." This novel framework has yielded remarkable results, surpassing state-of-the-art Federated Learning (FL) algorithms, particularly when benchmarked against non-iid CIFAR100 datasets. The success of FedNets can be attributed to its intricate architecture, consisting of several key components, including the Ensemble Generation and Pruning component, the Graph Conversion and Embedding component, and the Clustering of Embeddings component.

One of the distinguishing features of FedNets is its reliance on sharing members of deep learning models, or even the models themselves, as opposed to the conventional approach of sharing only model weights. Although these models undergo pruning for size reduction, transmitting them over the network can still pose significant bandwidth and latency challenges. As a result, further experimentation is required to assess the implications of model transmission on both bandwidth usage and latency.

Expanding on this, FedNets exhibits the potential to become a cornerstone of the next generation of heterogeneous Federated Learning systems, particularly in IoT (Internet of Things) environments. The framework accommodates clients with an ensemble of models, with the knowledge embedded within each client's models distilled into graph embedding vectors. This design enables seamless integration of diverse model architectures without compromising the overall system's functionality.

In the context of heterogeneous models, where each participant device or node in a Federated Learning system employs a unique model architecture, FedNets offers numerous advantages. These include performance optimization and the implementation of advanced ensemble learning techniques like stacking, thereby enhancing the system's adaptability and efficacy.

Nevertheless, navigating a Federated Learning system equipped with heterogeneous models presents its set of challenges. Foremost among these challenges is the need to address increased communication overhead and ensure equitable learning across the diverse nodes. As we delve deeper into our research, we recognize that addressing these considerations is pivotal to unlocking the full potential of FedNets in heterogeneous Federated Learning environments, especially in the IoT landscape.

The importance of our contribution extends beyond the realm of machine learning techniques. It aligns with the evolving landscape of IoT, where a multitude of interconnected devices collect and process data. Traditional centralized data collection and model training approaches are not only impractical but also raise serious concerns regarding data privacy and security. By adopting FedNets and Federated Learning, we inherently address these concerns. Federated Learning, as exemplified by FedNets, enables localized model training on IoT devices, eliminating the need for raw data transmission to centralized servers. This not only reduces communication overhead but also safeguards sensitive information as data never leaves the device boundary. Moreover, it aligns seamlessly with data protection regulations and privacy requirements, ensuring compliance and reinforcing user trust.

Overall, we believe that these identified previous areas of investigation will significantly advance our research and provide valuable insights into the practical applications of our proposed methods.

Bibliography

Abadi, Martín et al. (May 2016). *TensorFlow: A system for large-scale machine learning*. arXiv:1605.08695 [cs]. DOI: 10.48550/arXiv.1605.08695. URL: http://arxiv.org/abs/1605.08695 (visited on 01/24/2023).

Agarwal, Mayank, Mikhail Yurochkin, and Yuekai Sun (2022). "Personalization in Federated Learning". In: Springer International Publishing. DOI: 10.1007/978-3-030-96896-0_4. URL: http://dx.doi.org/10.1007/978-3-030-96896-0_4.

Aghasi, Alireza, Afshin Abdi, and Justin Romberg (June 2018). "Fast Convex Pruning of Deep Neural Networks". In: *arXiv:1806.06457 [cs, stat]*. arXiv: 1806.06457. URL: http://arxiv.org/abs/1806.06457 (visited on 03/26/2019).

Aghasi, Alireza et al. (2017). "Net-Trim: Convex Pruning of Deep Neural Networks with Performance Guarantee". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., pp. 3177–3186. URL: http://papers.nips.cc/paper/6910-net-trim-convex-pruning-of-deep-neural-networks-with-performance-guarantee.pdf (visited on 03/26/2019).

Agrawal, Shikha and Jitendra Agrawal (Jan. 2015). "Survey on Anomaly Detection using Data Mining Techniques". en. In: *Procedia Computer Science*. Knowledge-Based and Intelligent Information & Engineering Systems 19th Annual Conference, KES-2015, Singapore, September 2015 Proceedings 60, pp. 708–713. ISSN: 1877-0509. DOI: 10.1016/j.procs.2015.08.220. URL: https://www.sciencedirect.com/science/article/pii/S1877050915023479 (visited on 02/02/2023).

Ahad, Abdul, Mohammad Tahir, and Kok-Lim Alvin Yau (2019). "5G-Based Smart Healthcare Network: Architecture, Taxonomy, Challenges and Future Research Directions". In: *IEEE Access* 7. Conference Name: IEEE Access, pp. 100747–100762. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2930628.

Ahmad, Ijaz et al. (2020). "Machine Learning Meets Communication Networks: Current Trends and Future Challenges". In: *IEEE Access* 8, pp. 223418–223460. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3041765. URL: https://ieeexplore.ieee.org/document/9274307/ (visited on 03/10/2023).

Ahmed, Mohiuddin, Raihan Seraj, and Syed Mohammed Shamsul Islam (Aug. 2020). "The k-means Algorithm: A Comprehensive Survey and Performance Evaluation". en. In: *Electronics* 9.8. Number: 8 Publisher: Multidisciplinary Digital Publishing Institute, p. 1295. ISSN: 2079-9292. DOI: 10.3390/electronics9081295. URL: https://www.mdpi.com/2079-9292/9/8/1295 (visited on 02/02/2023).

Ahmed, Shameem et al. (Aug. 2022). "Binary Simulated Normal Distribution Optimizer for feature selection: Theory and application in COVID-19 datasets". en. In: *Expert Systems with Applications* 200, p. 116834. ISSN: 09574174. DOI: 10.1016/j.eswa.2022.116834. URL: https://linkinghub.elsevier.com/retrieve/pii/S0957417422002871 (visited on 02/18/2023).

Al-Fuqaha, Ala et al. (2015). "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications". In: *IEEE Communications Surveys & Tutorials* 17.4, pp. 2347–2376. ISSN: 1553-877X, 2373-745X. DOI: 10.1109/COMST.2015.2444095. URL: https://ieeexplore.ieee.org/document/7123563/ (visited on 01/09/2023).

Al-Qaseemi, Sarah A. et al. (Dec. 2016). "IoT architecture challenges and issues: Lack of standardization". In: *2016 Future Technologies Conference (FTC)*. San Francisco, CA, USA: IEEE, pp. 731–738. ISBN: 978-1-5090-4171-8. DOI: 10.1109/FTC.2016.7821686. URL: http://ieeexplore.ieee.org/document/7821686/ (visited on 01/09/2023).

Alam, Kazi Md. Rokibul, Nazmul Siddique, and Hojjat Adeli (June 2020). "A dynamic ensemble learning algorithm for neural networks". en. In: *Neural Computing and Applications* 32.12, pp. 8675–8690. ISSN: 0941-0643, 1433-3058. DOI: 10.1007/s00521-019-04359-7. URL: http://link.springer.com/10.1007/s00521-019-04359-7 (visited on 10/26/2023).

Alhalabi, Besher, Shadi Basurra, and Mohamed Medhat Gaber (2023). "FedNets: Federated Learning on Edge Devices Using Ensembles of Pruned Deep Neural Networks". In: *IEEE Access* 11. Conference Name: IEEE Access, pp. 30726–30738. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2023.3261266.

Alhalabi, Besher, Mohamed Medhat Gaber, and Shadi Basura (Dec. 2021). "MicroNets: A multi-phase pruning pipeline to deep ensemble learning in IoT devices". en. In: *Computers & Electrical Engineering* 96, p. 107581. ISSN: 00457906. DOI: 10.1016/j.compeleceng.2021.107581. URL: https://linkinghub.elsevier.com/retrieve/pii/S0045790621005164 (visited on 12/08/2021).

Alhalabi, Besher, Mohamed Medhat Gaber, and Shadi Basurra (Oct. 2019). "EnSyth: A Pruning Approach to Synthesis of Deep Learning Ensembles". In: *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. Bari, Italy: IEEE, pp. 3466–3473. ISBN: 978-1-72814-569-3. DOI: 10.1109/SMC.2019.8913944. URL: https://ieeexplore.ieee.org/document/8913944/ (visited on 11/10/2022).

Ali, Muhammad Salek et al. (2019). "Applications of Blockchains in the Internet of Things: A Comprehensive Survey". In: *IEEE Communications Surveys & Tutorials* 21.2, pp. 1676–1717. ISSN: 1553-877X, 2373-745X. DOI: 10.1109/COMST.2018.2886932. URL: https://ieeexplore.ieee.org/document/8580364/ (visited on 01/03/2023).

Allhoff, Fritz and Adam Henschke (Sept. 2018). "The Internet of Things: Foundational ethical issues". en. In: *Internet of Things* 1-2, pp. 55–66. ISSN: 25426605. DOI: 10.1016/j.iot.2018.08.005. URL: https://linkinghub.elsevier.com/retrieve/pii/S2542660518300532 (visited on 03/05/2023).

Alom, Md Zahangir et al. (July 2018). "Effective Quantization Approaches for Recurrent Neural Networks". In: *2018 International Joint Conference on*

Neural Networks (IJCNN). Rio de Janeiro: IEEE, pp. 1–8. ISBN: 978-1-5090-6014-6. DOI: 10.1109/IJCNN.2018.8489341. URL: https://ieeexplore.ieee.org/document/8489341/ (visited on 03/10/2023).

Amari, Shun-ichi (June 1993). "Backpropagation and stochastic gradient descent method". en. In: *Neurocomputing* 5.4-5, pp. 185–196. ISSN: 09252312. DOI: 10.1016/0925-2312(93)90006-O. URL: https://linkinghub.elsevier.com/retrieve/pii/092523129390006O (visited on 01/23/2023).

Angeline, P.J., G.M. Saunders, and J.B. Pollack (Jan. 1994). "An evolutionary algorithm that constructs recurrent neural networks". In: *IEEE Transactions on Neural Networks* 5.1, pp. 54–65. ISSN: 10459227. DOI: 10.1109/72.265960. URL: http://ieeexplore.ieee.org/document/265960/ (visited on 02/22/2019).

Arivazhagan, Manoj Ghuhan et al. (Dec. 2019). "Federated Learning with Personalization Layers". In: *arXiv:1912.00818 [cs, stat]*. arXiv: 1912.00818. URL: http://arxiv.org/abs/1912.00818 (visited on 04/15/2021).

Ash, Timur (Jan. 1989). "Dynamic Node Creation in Backpropagation Networks". en. In: *Connection Science* 1.4, pp. 365–375. ISSN: 0954-0091, 1360-0494. DOI: 10.1080/09540098908915647. URL: https://www.tandfonline.com/doi/full/10.1080/09540098908915647 (visited on 10/26/2023).

Ashton, Kevin and others (2009). "That 'internet of things' thing". In: 22.7, pp. 97–114.

Badnakhe, Rahul (Feb. 2021). "How big is the IoT market?" In: URL: https://www.iotcentral.io/blog/iot-is-not-a-buzzword-but-necessity.

Badrinarayanan, Vijay, Alex Kendall, and Roberto Cipolla (Dec. 2017). "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation". en. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.12, pp. 2481–2495. ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2016.2644615. URL: https://ieeexplore.ieee.org/document/7803544/ (visited on 05/29/2023).

Bai, Yunsheng et al. (June 2019). "Unsupervised Inductive Graph-Level Representation Learning via Graph-Graph Proximity". In: *arXiv:1904.01098 [cs, stat]*. arXiv: 1904.01098. URL: http://arxiv.org/abs/1904.01098 (visited on 03/15/2022).

Bakker, Bart and Tom Heskes (Mar. 2003). "Clustering ensembles of neural network models". en. In: *Neural Networks* 16.2, pp. 261–269. ISSN: 08936080. DOI: 10.1016/S0893-6080(02)00187-9. URL: https://linkinghub.elsevier.com/retrieve/pii/S0893608002001879 (visited on 11/07/2023).

Barioli, Francesco, Shaun Fallat, and Leslie Hogben (Nov. 2004). "Computation of minimal rank and path cover number for certain graphs". en. In: *Linear Algebra and its Applications* 392, pp. 289–303. ISSN: 00243795. DOI: 10.1016/j.laa.2004.06.019. URL: https://linkinghub.elsevier.com/retrieve/pii/S0024379504002964 (visited on 06/13/2023).

Beasley, J.E and P.C Chu (Oct. 1996). "A genetic algorithm for the set covering problem". en. In: *European Journal of Operational Research* 94.2, pp. 392–404. ISSN: 03772217. DOI: 10.1016/0377-2217(95)00159-X. URL: http://linkinghub.elsevier.com/retrieve/pii/037722179500159X (visited on 02/25/2019).

Bellini, Pierfrancesco, Paolo Nesi, and Gianni Pantaleo (Feb. 2022). "IoT-Enabled Smart Cities: A Review of Concepts, Frameworks and Key Technologies".

en. In: *Applied Sciences* 12.3, p. 1607. ISSN: 2076-3417. DOI: 10.3390/app12031607. URL: https://www.mdpi.com/2076-3417/12/3/1607 (visited on 01/03/2023).

Bentéjac, Candice, Anna Csörgő, and Gonzalo Martínez-Muñoz (Mar. 2021). "A comparative analysis of gradient boosting algorithms". en. In: *Artificial Intelligence Review* 54.3, pp. 1937–1967. ISSN: 0269-2821, 1573-7462. DOI: 10.1007/s10462-020-09896-5. URL: https://link.springer.com/10.1007/s10462-020-09896-5 (visited on 01/17/2023).

Berrada, Imane Rhzioual, Fatima Zohra Barramou, and Omar Bachir Alami (Feb. 2022). "A review of Artificial Intelligence approach for credit risk assessment". In: *2022 2nd International Conference on Artificial Intelligence and Signal Processing (AISP)*. ISSN: 2640-5768, pp. 1–5. DOI: 10.1109/AISP53593.2022.9760655.

Berrar, Daniel (Jan. 2018). "Cross-Validation". In: ISBN: 978-0-12-809633-8. DOI: 10.1016/B978-0-12-809633-8.20349-X.

Berrezueta-Guzman, Jonnathan et al. (2020). "Smart-Home Environment to Support Homework Activities for Children". In: *IEEE Access* 8. Conference Name: IEEE Access, pp. 160251–160267. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3020734.

Bi, Hongliang, Jiajia Liu, and Nei Kato (July 2022). "Deep Learning-Based Privacy Preservation and Data Analytics for IoT Enabled Healthcare". In: *IEEE Transactions on Industrial Informatics* 18.7, pp. 4798–4807. ISSN: 1551-3203, 1941-0050. DOI: 10.1109/TII.2021.3117285. URL: https://ieeexplore.ieee.org/document/9565344/ (visited on 05/29/2023).

Bienstock, Daniel, Gonzalo Muñoz, and Sebastian Pokutta (Mar. 2022). *Principled Deep Neural Network Training through Linear Programming*. arXiv:1810.03218 [cs, math, stat]. DOI: 10.48550/arXiv.1810.03218. URL: http://arxiv.org/abs/1810.03218 (visited on 03/10/2023).

"Big IoT Data Analytics" (2017). "Big IoT Data Analytics: Architecture, Opportunities, and Open Research Challenges". In: *IEEE Access* 5, pp. 5247–5261. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2689040. URL: http://ieeexplore.ieee.org/document/7888916/ (visited on 03/17/2022).

Block, H. D. (Jan. 1962). "The Perceptron: A Model for Brain Functioning. I". en. In: *Reviews of Modern Physics* 34.1, pp. 123–135. ISSN: 0034-6861. DOI: 10.1103/RevModPhys.34.123. URL: https://link.aps.org/doi/10.1103/RevModPhys.34.123 (visited on 01/22/2023).

Bloem, Bastiaan R, Michael S Okun, and Christine Klein (June 2021). "Parkinson's disease". en. In: *The Lancet* 397.10291, pp. 2284–2303. ISSN: 01406736. DOI: 10.1016/S0140-6736(21)00218-X. URL: https://linkinghub.elsevier.com/retrieve/pii/S014067362100218X (visited on 02/27/2023).

Bottou, Léon (2010). "Large-Scale Machine Learning with Stochastic Gradient Descent". en. In: *Proceedings of COMPSTAT'2010*. Ed. by Yves Lechevallier and Gilbert Saporta. Heidelberg: Physica-Verlag HD, pp. 177–186. ISBN: 978-3-7908-2603-6 978-3-7908-2604-3. DOI: 10.1007/978-3-7908-2604-3_16. URL: http://link.springer.com/10.1007/978-3-7908-2604-3_16 (visited on 06/27/2022).

Boyacioglu, Melek Acar, Yakup Kara, and Ömer Kaan Baykan (Mar. 2009). "Predicting bank financial failures using neural networks, support vector machines and multivariate statistical methods: A comparative analysis in

the sample of savings deposit insurance fund (SDIF) transferred banks in Turkey". en. In: *Expert Systems with Applications* 36.2, pp. 3355–3366. ISSN: 09574174. DOI: 10.1016/j.eswa.2008.01.003. URL: https://linkinghub.elsevier.com/retrieve/pii/S095741740800078X (visited on 01/22/2023).

Briggs, Christopher, Zhong Fan, and Peter Andras (July 2020). "Federated learning with hierarchical clustering of local updates to improve training on non-IID data". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. ISSN: 2161-4407, pp. 1–9. DOI: 10.1109/IJCNN48605.2020.9207469.

Brown, Gavin et al. (2005). "Managing diversity in regression ensembles." In: *Journal of machine learning research* 6.9.

Brumley, David and Dan Boneh (Aug. 2005). "Remote timing attacks are practical". en. In: *Computer Networks* 48.5, pp. 701–716. ISSN: 13891286. DOI: 10.1016/j.comnet.2005.01.010. URL: https://linkinghub.elsevier.com/retrieve/pii/S1389128605000125 (visited on 01/10/2023).

Buciluǎ, Cristian, Rich Caruana, and Alexandru Niculescu-Mizil (2006). "Model compression". en. In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '06*. Philadelphia, PA, USA: ACM Press, p. 535. ISBN: 978-1-59593-339-3. DOI: 10.1145/1150402.1150464. URL: http://portal.acm.org/citation.cfm?doid=1150402.1150464 (visited on 02/28/2019).

Burhan, Muhammad et al. (Aug. 2018). "IoT Elements, Layered Architectures and Security Issues: A Comprehensive Survey". en. In: *Sensors* 18.9, p. 2796. ISSN: 1424-8220. DOI: 10.3390/s18092796. URL: http://www.mdpi.com/1424-8220/18/9/2796 (visited on 01/10/2023).

Bühlmann, Peter (2012). "Bagging, Boosting and Ensemble Methods". en. In: *Handbook of Computational Statistics*. Ed. by James E. Gentle, Wolfgang Karl Härdle, and Yuichi Mori. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 985–1022. ISBN: 978-3-642-21550-6 978-3-642-21551-3. DOI: 10.1007/978-3-642-21551-3_33. URL: http://link.springer.com/10.1007/978-3-642-21551-3_33 (visited on 01/16/2023).

Caldarola, Debora et al. (June 2021). "Cluster-driven Graph Federated Learning over Multiple Domains". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. Nashville, TN, USA: IEEE, pp. 2743–2752. ISBN: 978-1-66544-899-4. DOI: 10.1109/CVPRW53098.2021.00309. URL: https://ieeexplore.ieee.org/document/9522933/ (visited on 07/04/2022).

Cao, Ying et al. (June 2013). "Advance and Prospects of AdaBoost Algorithm". en. In: *Acta Automatica Sinica* 39.6, pp. 745–758. ISSN: 18741029. DOI: 10.1016/S1874-1029(13)60052-X. URL: https://linkinghub.elsevier.com/retrieve/pii/S187410291360052X (visited on 01/17/2023).

Chandola, Varun, Arindam Banerjee, and Vipin Kumar (July 2009). "Anomaly detection: A survey". In: *ACM Computing Surveys* 41.3, 15:1–15:58. ISSN: 0360-0300. DOI: 10.1145/1541880.1541882. URL: https://doi.org/10.1145/1541880.1541882 (visited on 02/02/2023).

Chang, Zhuoqing et al. (Sept. 2021). "A Survey of Recent Advances in Edge-Computing-Powered Artificial Intelligence of Things". In: *IEEE Internet of Things Journal* 8.18, pp. 13849–13875. ISSN: 2327-4662, 2372-2541. DOI:

10.1109/JIOT.2021.3088875. URL: https://ieeexplore.ieee.org/document/9453402/ (visited on 03/02/2023).

Chauhan, Jagmohan et al. (May 2018). "Breathing-Based Authentication on Resource-Constrained IoT Devices using Recurrent Neural Networks". In: *Computer* 51.5, pp. 60–67. ISSN: 0018-9162, 1558-0814. DOI: 10.1109/MC.2018.2381119. URL: https://ieeexplore.ieee.org/document/8364655/ (visited on 03/05/2023).

Chen, Mingzhe et al. (Apr. 2021). "Communication-efficient federated learning". In: *Proceedings of the National Academy of Sciences* 118.17. Publisher: Proceedings of the National Academy of Sciences, e2024789118. DOI: 10.1073/pnas.2024789118. URL: https://www.pnas.org/doi/10.1073/pnas.2024789118 (visited on 02/24/2023).

Chen, Tianqi, Ian Goodfellow, and Jonathon Shlens (Nov. 2015). "Net2Net: Accelerating Learning via Knowledge Transfer". en. In: *arXiv:1511.05641 [cs]*. arXiv: 1511.05641. URL: http://arxiv.org/abs/1511.05641 (visited on 10/09/2018).

Chen, Yiqiang et al. (July 2020). "FedHealth: A Federated Transfer Learning Framework for Wearable Healthcare". In: *IEEE Intelligent Systems* 35.4, pp. 83–93. ISSN: 1541-1672, 1941-1294. DOI: 10.1109/MIS.2020.2988604. URL: https://ieeexplore.ieee.org/document/9076082/ (visited on 04/15/2021).

Cheng, Yu et al. (Oct. 2017). "A Survey of Model Compression and Acceleration for Deep Neural Networks". en. In: *arXiv:1710.09282 [cs]*. arXiv: 1710.09282. URL: http://arxiv.org/abs/1710.09282 (visited on 09/12/2018).

Collobert, Ronan et al. (2011). "Natural Language Processing (Almost) from Scratch". en. In: *NATURAL LANGUAGE PROCESSING*, p. 45.

Conroy, Bryan et al. (Mar. 2016). "A dynamic ensemble approach to robust classification in the presence of missing data". en. In: *Machine Learning* 102.3, pp. 443–463. ISSN: 0885-6125, 1573-0565. DOI: 10.1007/s10994-015-5530-z. URL: http://link.springer.com/10.1007/s10994-015-5530-z (visited on 10/26/2023).

Conti, Mauro, Nicola Dragoni, and Viktor Lesyk (2016). "A Survey of Man In The Middle Attacks". In: *IEEE Communications Surveys & Tutorials* 18.3, pp. 2027–2051. ISSN: 1553-877X. DOI: 10.1109/COMST.2016.2548426. URL: http://ieeexplore.ieee.org/document/7442758/ (visited on 01/10/2023).

Cortes, Corinna, Mehryar Mohri, and Afshin Rostamizadeh (2012). "L2 Regularization for Learning Kernels". In: Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.1205.2653. URL: https://arxiv.org/abs/1205.2653 (visited on 10/27/2023).

Cortes, Corinna et al. (July 2016). "AdaNet: Adaptive Structural Learning of Artificial Neural Networks". In: *arXiv:1607.01097 [cs]*. arXiv: 1607.01097. URL: http://arxiv.org/abs/1607.01097 (visited on 07/02/2019).

Costarelli, Danilo and Renato Spigler (Aug. 2013). "Approximation results for neural network operators activated by sigmoidal functions". en. In: *Neural Networks* 44, pp. 101–106. ISSN: 08936080. DOI: 10.1016/j.neunet.2013.03.015. URL: https://linkinghub.elsevier.com/retrieve/pii/S0893608013001007 (visited on 01/22/2023).

Cruz, Rafael M.O. et al. (May 2015). "META-DES: A dynamic ensemble selection framework using meta-learning". en. In: *Pattern Recognition* 48.5, pp. 1925–1935. ISSN: 00313203. DOI: 10.1016/j.patcog.2014.12.003. URL: https://linkinghub.elsevier.com/retrieve/pii/S0031320314004919 (visited on 10/26/2023).

D. Whitley, C. Bogart (1990). "The Evolution of Connectivity: Pruning Neural Networks Using Genetic Algorithms". In.

Dai, Hong-Ning, Zibin Zheng, and Yan Zhang (Oct. 2019). "Blockchain for Internet of Things: A Survey". In: *IEEE Internet of Things Journal* 6.5, pp. 8076–8094. ISSN: 2327-4662, 2372-2541. DOI: 10.1109/JIOT.2019.2920987. URL: https://ieeexplore.ieee.org/document/8731639/ (visited on 01/04/2023).

Dao, Tri et al. (n.d.). "A Kernel Theory of Modern Data Augmentation". en. In: (), p. 10.

Deb, K. et al. (Apr. 2002). "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE Transactions on Evolutionary Computation* 6.2, pp. 182–197. ISSN: 1089778X. DOI: 10.1109/4235.996017. URL: http://ieeexplore.ieee.org/document/996017/ (visited on 02/25/2019).

Demir-Kavuk, Ozgur et al. (Dec. 2011). "Prediction using step-wise L1, L2 regularization and feature selection for small data sets with large number of features". en. In: *BMC Bioinformatics* 12.1, p. 412. ISSN: 1471-2105. DOI: 10.1186/1471-2105-12-412. URL: https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-12-412 (visited on 10/27/2023).

Denton, Emily et al. (Apr. 2014). "Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation". en. In: *arXiv:1404.0736 [cs]*. arXiv: 1404.0736. URL: http://arxiv.org/abs/1404.0736 (visited on 10/09/2018).

Deogirikar, Jyoti and Amarsinh Vidhate (Feb. 2017). "Security attacks in IoT: A survey". In: *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*. Palladam, Tamilnadu, India: IEEE, pp. 32–37. ISBN: 978-1-5090-3242-6 978-1-5090-3243-3. DOI: 10.1109/I-SMAC.2017.8058363. URL: http://ieeexplore.ieee.org/document/8058363/ (visited on 01/11/2023).

Deotte, Chris et al. (Oct. 2021). "GPU Accelerated Boosted Trees and Deep Neural Networks for Better Recommender Systems". en. In: *RecSysChallenge '21: Proceedings of the Recommender Systems Challenge 2021*. Amsterdam Netherlands: ACM, pp. 7–14. ISBN: 978-1-4503-8693-7. DOI: 10.1145/3487572.3487605. URL: https://dl.acm.org/doi/10.1145/3487572.3487605 (visited on 01/19/2023).

Dharwadkar, Nagaraj V. and Priyanka S. Patil (2018). "Customer retention and credit risk analysis using ANN, SVM and DNN". en. In: *International Journal of Society Systems Science* 10.4, p. 316. ISSN: 1756-2511, 1756-252X. DOI: 10.1504/IJSSS.2018.095601. URL: http://www.inderscience.com/link.php?id=95601 (visited on 04/06/2023).

Dietterich, Thomas G. (2000). "Ensemble Methods in Machine Learning". en. In: *Multiple Classifier Systems*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 1–15. ISBN: 978-3-540-45014-6. DOI: 10.1007/3-540-45014-9_1.

Dilsizian, Steven E. and Eliot L. Siegel (Dec. 2013). "Artificial Intelligence in Medicine and Cardiac Imaging: Harnessing Big Data and Advanced Computing to Provide Personalized Medical Diagnosis and Treatment". en. In: *Current Cardiology Reports* 16.1, p. 441. ISSN: 1534-3170. DOI: 10.1007/s11886-013-0441-8. URL: https://doi.org/10.1007/s11886-013-0441-8 (visited on 02/27/2023).

Ding, Shifei et al. (Mar. 2013). "Evolutionary artificial neural networks: a review". en. In: *Artificial Intelligence Review* 39.3, pp. 251–260. ISSN: 0269-2821, 1573-7462. DOI: 10.1007/s10462-011-9270-6. URL: http://link.springer.com/10.1007/s10462-011-9270-6 (visited on 02/25/2019).

Dong, Qi, Shaogang Gong, and Xiatian Zhu (June 2019). "Imbalanced Deep Learning by Minority Class Incremental Rectification". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41.6. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1367–1381. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2018.2832629.

Duan, Moming et al. (Nov. 2019). "Astraea: Self-Balancing Federated Learning for Improving Classification Accuracy of Mobile Deep Learning Applications". In: *2019 IEEE 37th International Conference on Computer Design (ICCD)*. Abu Dhabi, United Arab Emirates: IEEE, pp. 246–254. ISBN: 978-1-5386-6648-7. DOI: 10.1109/ICCD46524.2019.00038. URL: https://ieeexplore.ieee.org/document/8988732/ (visited on 06/29/2022).

Elkin, Dmitry and Valeriy Vyatkin (2020). "IoT in Traffic Management: Review of Existing Methods of Road Traffic Regulation". en. In: *Applied Informatics and Cybernetics in Intelligent Systems*. Ed. by Radek Silhavy. Vol. 1226. Series Title: Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, pp. 536–551. ISBN: 978-3-030-51973-5 978-3-030-51974-2. DOI: 10.1007/978-3-030-51974-2_50. URL: https://link.springer.com/10.1007/978-3-030-51974-2_50 (visited on 01/03/2023).

Engelbrecht, A.P. (Nov. 2001). "A new pruning heuristic based on variance analysis of sensitivity information". In: *IEEE Transactions on Neural Networks* 12.6. Conference Name: IEEE Transactions on Neural Networks, pp. 1386–1399. ISSN: 1941-0093. DOI: 10.1109/72.963775.

Evans, D. J. and K. R. Raslan (July 2005). "The tanh function method for solving some important non-linear partial differential equations". In: *International Journal of Computer Mathematics* 82.7. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/00207160412331336026, pp. 897–905. ISSN: 0020-7160. DOI: 10.1080/00207160412331336026. URL: https://doi.org/10.1080/00207160412331336026 (visited on 01/22/2023).

Ezugwu, Absalom E. et al. (Apr. 2022). "A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects". en. In: *Engineering Applications of Artificial Intelligence* 110, p. 104743. ISSN: 0952-1976. DOI: 10.1016/j.engappai.2022.104743. URL: https://www.sciencedirect.com/science/article/pii/S095219762200046X (visited on 02/02/2023).

Falter, Maarten et al. (Mar. 2019). "Accuracy of Apple Watch Measurements for Heart Rate and Energy Expenditure in Patients With Cardiovascular Disease: Cross-Sectional Study". EN. In: *JMIR mHealth and uHealth* 7.3. Company: JMIR mHealth and uHealth Distributor: JMIR mHealth and uHealth Institution: JMIR mHealth and uHealth Label: JMIR mHealth

and uHealth Publisher: JMIR Publications Inc., Toronto, Canada, e11889. DOI: 10.2196/11889. URL: https://mhealth.jmir.org/2019/3/e11889 (visited on 02/27/2023).

Farabet, Clement et al. (May 2010). "Hardware accelerated convolutional neural networks for synthetic vision systems". In: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. Paris, France: IEEE, pp. 257–260. ISBN: 978-1-4244-5308-5. DOI: 10.1109/ISCAS.2010.5537908. URL: http://ieeexplore.ieee.org/document/5537908/ (visited on 01/05/2019).

Farabet, Clement et al. (June 2011). "NeuFlow: A runtime reconfigurable dataflow processor for vision". In: *CVPR 2011 WORKSHOPS*. Colorado Springs, CO, USA: IEEE, pp. 109–116. ISBN: 978-1-4577-0529-8. DOI: 10.1109/CVPRW.2011.5981829. URL: http://ieeexplore.ieee.org/document/5981829/ (visited on 01/05/2019).

Fawcett, Tom (June 2006). "An introduction to ROC analysis". en. In: *Pattern Recognition Letters* 27.8, pp. 861–874. ISSN: 01678655. DOI: 10.1016/j.patrec.2005.10.010. URL: https://linkinghub.elsevier.com/retrieve/pii/S016786550500303X (visited on 01/31/2023).

Ferreira, Artur J. and Mário A. T. Figueiredo (2012). "Boosting Algorithms: A Review of Methods, Theory, and Applications". en. In: *Ensemble Machine Learning*. Ed. by Cha Zhang and Yunqian Ma. Boston, MA: Springer US, pp. 35–85. ISBN: 978-1-4419-9325-0 978-1-4419-9326-7. DOI: 10.1007/978-1-4419-9326-7_2. URL: http://link.springer.com/10.1007/978-1-4419-9326-7_2 (visited on 01/16/2023).

Floridi, Luciano, ed. (June 2016). *The Routledge Handbook of Philosophy of Information*. en. 0th ed. Routledge. ISBN: 978-1-317-63349-5. DOI: 10.4324/9781315757544. URL: https://www.taylorfrancis.com/books/9781317633495 (visited on 03/05/2023).

Frankle, Jonathan and Michael Carbin (Mar. 2019). "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *arXiv:1803.03635 [cs]*. arXiv: 1803.03635. URL: http://arxiv.org/abs/1803.03635 (visited on 02/16/2020).

Freund, Yoav and Robert E Schapire (Aug. 1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". en. In: *Journal of Computer and System Sciences* 55.1, pp. 119–139. ISSN: 00220000. DOI: 10.1006/jcss.1997.1504. URL: https://linkinghub.elsevier.com/retrieve/pii/S002200009791504X (visited on 01/16/2023).

Fridman, Lex et al. (Nov. 2017). "MIT Autonomous Vehicle Technology Study: Large-Scale Deep Learning Based Analysis of Driver Behavior and Interaction with Automation". In: *arXiv:1711.06976 [cs]*. arXiv: 1711.06976. URL: http://arxiv.org/abs/1711.06976 (visited on 03/22/2019).

Furlanello, Tommaso et al. (June 2018). *Born Again Neural Networks*. arXiv:1805.04770 [cs, stat]. DOI: 10.48550/arXiv.1805.04770. URL: http://arxiv.org/abs/1805.04770 (visited on 04/17/2023).

Gaber, Mohamed Medhat, Frederic Stahl, and João Bártolo Gomes (2014). *Pocket Data Mining: Big Data on Small Devices*. en. Vol. 2. Studies in Big Data. Cham: Springer International Publishing. ISBN: 978-3-319-02710-4 978-3-319-02711-1. DOI: 10.1007/978-3-319-02711-1. URL: https://link.springer.com/10.1007/978-3-319-02711-1 (visited on 03/07/2023).

Ganaie, M.A. et al. (Oct. 2022). "Ensemble deep learning: A review". en. In: *Engineering Applications of Artificial Intelligence* 115, p. 105151. ISSN: 09521976. DOI: 10.1016/j.engappai.2022.105151. URL: https://linkinghub.elsevier.com/retrieve/pii/S095219762200269X (visited on 10/26/2023).

George, D. and J. Hawkins (2005). "A hierarchical bayesian model of invariant pattern recognition in the visual cortex". In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 3. Montreal, QC, Canada: IEEE, pp. 1812–1817. ISBN: 978-0-7803-9048-5. DOI: 10.1109/IJCNN.2005.1556155. URL: http://ieeexplore.ieee.org/document/1556155/ (visited on 01/05/2019).

Ghahramani, Zoubin (2004). "Unsupervised Learning". en. In: *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures*. Ed. by Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rätsch. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 72–112. ISBN: 978-3-540-28650-9. DOI: 10.1007/978-3-540-28650-9_5. URL: https://doi.org/10.1007/978-3-540-28650-9_5 (visited on 02/02/2023).

Ghosh, Ananda and Katarina Grolinger (2020). "Edge-Cloud Computing for IoT Data Analytics: Embedding Intelligence in the Edge with Deep Learning". In: *IEEE Transactions on Industrial Informatics*, pp. 1–1. ISSN: 1551-3203, 1941-0050. DOI: 10.1109/TII.2020.3008711. URL: https://ieeexplore.ieee.org/document/9139356/ (visited on 03/17/2022).

Ghosh, Avishek et al. (June 2020). "An Efficient Framework for Clustered Federated Learning". In: *arXiv:2006.04088 [cs, stat]*. arXiv: 2006.04088. URL: http://arxiv.org/abs/2006.04088 (visited on 04/15/2021).

Ghosh, Jeet, Gopinath Samanta, and Chinmay Chakraborty (2021). "Smart Health Care for Societies: An Insight into the Implantable and Wearable Devices for Remote Health Monitoring". en. In: *Green Technological Innovation for Sustainable Smart Societies*. Ed. by Chinmay Chakraborty. Cham: Springer International Publishing, pp. 89–113. ISBN: 978-3-030-73294-3 978-3-030-73295-0. DOI: 10.1007/978-3-030-73295-0_5. URL: https://link.springer.com/10.1007/978-3-030-73295-0_5 (visited on 03/17/2022).

Ghosh, Tapabrata (Jan. 2017). "QuickNet: Maximizing Efficiency and Efficacy in Deep Architectures". In: *arXiv:1701.02291 [cs, stat]*. arXiv: 1701.02291. URL: http://arxiv.org/abs/1701.02291 (visited on 07/02/2019).

Giacinto, G., F. Roli, and G. Fumera (2000). "Design of effective multiple classifier systems by clustering of classifiers". In: *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*. Vol. 2. Barcelona, Spain: IEEE Comput. Soc, pp. 160–163. ISBN: 978-0-7695-0750-7. DOI: 10.1109/ICPR.2000.906039. URL: http://ieeexplore.ieee.org/document/906039/ (visited on 11/07/2023).

Girosi, Federico, Michael Jones, and Tomaso Poggio (Mar. 1995). "Regularization Theory and Neural Networks Architectures". en. In: *Neural Computation* 7.2, pp. 219–269. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco.1995.7.2.219. URL: http://www.mitpressjournals.org/doi/10.1162/neco.1995.7.2.219 (visited on 03/22/2019).

Gong, Yunchao et al. (Dec. 2014). "Compressing Deep Convolutional Networks using Vector Quantization". en. In: *arXiv:1412.6115 [cs]*. arXiv: 1412.6115. URL: http://arxiv.org/abs/1412.6115 (visited on 10/09/2018).

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press.

Gordon, Ariel et al. (Nov. 2017). "MorphNet: Fast & Simple Resource-Constrained Structure Learning of Deep Networks". In: *arXiv:1711.06798 [cs, stat]*. arXiv: 1711.06798. URL: http://arxiv.org/abs/1711.06798 (visited on 07/02/2019).

Guo, Lin and Qun Dai (Feb. 2022). "Graph Clustering via Variational Graph Embedding". en. In: *Pattern Recognition* 122, p. 108334. ISSN: 00313203. DOI: 10.1016/j.patcog.2021.108334. URL: https://linkinghub.elsevier.com/retrieve/pii/S0031320321005148 (visited on 11/28/2022).

Gupta, Shashank and B. B. Gupta (Jan. 2017). "Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art". en. In: *International Journal of System Assurance Engineering and Management* 8.S1, pp. 512–530. ISSN: 0975-6809, 0976-4348. DOI: 10.1007/s13198-015-0376-0. URL: http://link.springer.com/10.1007/s13198-015-0376-0 (visited on 01/12/2023).

Hagiwara, Masafumi (Apr. 1994). "A simple and effective method for removal of hidden units and weights". en. In: *Neurocomputing* 6.2, pp. 207–218. ISSN: 09252312. DOI: 10.1016/0925-2312(94)90055-8. URL: http://linkinghub.elsevier.com/retrieve/pii/0925231294900558 (visited on 01/02/2019).

Han, Song, Huizi Mao, and William J. Dally (Oct. 2015). "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding". en. In: *arXiv:1510.00149 [cs]*. arXiv: 1510.00149. URL: http://arxiv.org/abs/1510.00149 (visited on 09/26/2018).

Hansen, L.K. and P. Salamon (Oct. 1990). "Neural network ensembles". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.10, pp. 993–1001. ISSN: 01628828. DOI: 10.1109/34.58871. URL: http://ieeexplore.ieee.org/document/58871/ (visited on 10/26/2023).

Hansen, Nikolaus and Andreas Ostermeier (June 2001). "Completely Derandomized Self-Adaptation in Evolution Strategies". en. In: *Evolutionary Computation* 9.2, pp. 159–195. ISSN: 1063-6560, 1530-9304. DOI: 10.1162/106365601750190398. URL: http://www.mitpressjournals.org/doi/10.1162/106365601750190398 (visited on 02/22/2019).

Harris, Charles R. et al. (Sept. 2020). "Array programming with NumPy". en. In: *Nature* 585.7825, pp. 357–362. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-020-2649-2. URL: https://www.nature.com/articles/s41586-020-2649-2 (visited on 04/04/2023).

Hashem, Sherif (June 1997). "Optimal Linear Combinations of Neural Networks". en. In: *Neural Networks* 10.4, pp. 599–614. ISSN: 08936080. DOI: 10.1016/S0893-6080(96)00098-6. URL: https://linkinghub.elsevier.com/retrieve/pii/S0893608096000986 (visited on 10/26/2023).

Hassibi, B., D.G. Stork, and G.J. Wolff (1993). "Optimal Brain Surgeon and general network pruning". In: *IEEE International Conference on Neural Networks*. San Francisco, CA, USA: IEEE, pp. 293–299. ISBN: 978-0-7803-0999-9. DOI: 10.1109/ICNN.1993.298572. URL: http://ieeexplore.ieee.org/document/298572/ (visited on 01/02/2019).

Hastie, Trevor, Jerome Friedman, and Robert Tibshirani (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY: Springer New York. ISBN: 978-1-4899-0519-2 978-0-387-21606-5. DOI: 10.1007/978-0-387-21606-5. URL: http://link.springer.com/10.1007/978-0-387-21606-5 (visited on 01/31/2023).

He, Kaiming et al. (June 2016a). "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, pp. 770–778. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.90. URL: http://ieeexplore.ieee.org/document/7780459/ (visited on 07/08/2019).

— (July 2016b). "Identity Mappings in Deep Residual Networks". In: *arXiv:1603.05027 [cs]*. arXiv: 1603.05027. URL: http://arxiv.org/abs/1603.05027 (visited on 02/21/2020).

He, Yang et al. (July 2019). "Filter Pruning via Geometric Median for Deep Convolutional Neural Networks Acceleration". In: *arXiv:1811.00250 [cs]*. arXiv: 1811.00250. URL: http://arxiv.org/abs/1811.00250 (visited on 09/19/2020).

He, Yihui, Xiangyu Zhang, and Jian Sun (2017). "Channel Pruning for Accelerating Very Deep Neural Networks". In: pp. 1389–1397. (Visited on 03/22/2019).

Hendrycks, Dan and Thomas Dietterich (Mar. 2019). *Benchmarking Neural Network Robustness to Common Corruptions and Perturbations*. arXiv:1903.12261 [cs, stat]. DOI: 10.48550/arXiv.1903.12261. URL: http://arxiv.org/abs/1903.12261 (visited on 04/13/2023).

Him, Leong Chee, Yu Yong Poh, and Lee Wah Pheng (Nov. 2019). "IoT-based Predictive Maintenance for Smart Manufacturing Systems". In: *2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. Lanzhou, China: IEEE, pp. 1942–1944. ISBN: 978-1-72813-248-8. DOI: 10.1109/APSIPAASC47483.2019.9023106. URL: https://ieeexplore.ieee.org/document/9023106/ (visited on 01/03/2023).

Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean (Mar. 2015). "Distilling the Knowledge in a Neural Network". In: *arXiv:1503.02531 [cs, stat]*. arXiv: 1503.02531. URL: http://arxiv.org/abs/1503.02531 (visited on 01/04/2019).

Hong, Jihoon, Jungwoo Shin, and Daeho Lee (Feb. 2016). "Strategic management of next-generation connected life: Focusing on smart key and car–home connectivity". en. In: *Technological Forecasting and Social Change* 103, pp. 11–20. ISSN: 0040-1625. DOI: 10.1016/j.techfore.2015.10.006. URL: https://www.sciencedirect.com/science/article/pii/S0040162515002942 (visited on 02/28/2023).

Hong-Jie Xing and Bao-Gang Hu (Apr. 2009). "Two-Phase Construction of Multilayer Perceptrons Using Information Theory". In: *IEEE Transactions on Neural Networks* 20.4, pp. 715–721. ISSN: 1045-9227, 1941-0093. DOI: 10.1109/TNN.2008.2005604. URL: http://ieeexplore.ieee.org/document/4796255/ (visited on 01/03/2019).

Howard, Andrew et al. (May 2019). "Searching for MobileNetV3". In: *arXiv:1905.02244 [cs]*. arXiv: 1905.02244. URL: http://arxiv.org/abs/1905.02244 (visited on 07/08/2019).

Howard, Andrew G. et al. (Apr. 2017). "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *arXiv:1704.04861*

[cs]. arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861 (visited on 01/04/2019).

Hrstka, O. et al. (Aug. 2003). "A competitive comparison of different types of evolutionary algorithms". en. In: *Computers & Structures* 81.18-19, pp. 1979–1990. ISSN: 00457949. DOI: 10.1016/S0045-7949(03)00217-7. URL: http://linkinghub.elsevier.com/retrieve/pii/S0045794903002177 (visited on 02/25/2019).

Hu, Hengyuan et al. (July 2016). "Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures". In: *arXiv:1607.03250 [cs]*. arXiv: 1607.03250. URL: http://arxiv.org/abs/1607.03250 (visited on 09/19/2020).

Hu, Yiming et al. (May 2018). "A novel channel pruning method for deep neural network compression". en. In: *arXiv:1805.11394 [cs, stat]*. arXiv: 1805.11394. URL: http://arxiv.org/abs/1805.11394 (visited on 11/14/2018).

Huang, Wenxuan, Thanassis Tiropanis, and George Konstantinidis (2022). "Federated Learning-Based IoT Intrusion Detection on Non-IID Data". en. In: *Internet of Things*. Ed. by Aurora González-Vidal et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 326–337. ISBN: 978-3-031-20936-9. DOI: 10.1007/978-3-031-20936-9_26.

Huang, Xuan, Lei Wu, and Yinsong Ye (Sept. 2019). "A Review on Dimensionality Reduction Techniques". In: *International Journal of Pattern Recognition and Artificial Intelligence* 33.10. Publisher: World Scientific Publishing Co., p. 1950017. ISSN: 0218-0014. DOI: 10.1142/S0218001419500174. URL: https://www.worldscientific.com/doi/abs/10.1142/S0218001419500174 (visited on 02/02/2023).

Huanhuan Chen and Xin Yao (Dec. 2009). "Regularized Negative Correlation Learning for Neural Network Ensembles". In: *IEEE Transactions on Neural Networks* 20.12, pp. 1962–1979. ISSN: 1045-9227, 1941-0093. DOI: 10.1109/TNN.2009.2034144. URL: http://ieeexplore.ieee.org/document/5337957/ (visited on 10/26/2023).

Huynh, T.Q. and R. Setiono (2005). "Effective neural network pruning using cross-validation". In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 2. Montreal, Que., Canada: IEEE, pp. 972–977. ISBN: 978-0-7803-9048-5. DOI: 10.1109/IJCNN.2005.1555984. URL: http://ieeexplore.ieee.org/document/1555984/ (visited on 01/02/2019).

Imteaj, Ahmed et al. (2023). "Federated Learning for Resource-Constrained IoT Devices: Panoramas and State of the Art". en. In: *Federated and Transfer Learning*. Ed. by Roozbeh Razavi-Far et al. Vol. 27. Series Title: Adaptation, Learning, and Optimization. Cham: Springer International Publishing, pp. 7–27. ISBN: 978-3-031-11747-3 978-3-031-11748-0. DOI: 10.1007/978-3-031-11748-0_2. URL: https://link.springer.com/10.1007/978-3-031-11748-0_2 (visited on 02/23/2023).

Ioffe, Sergey and Christian Szegedy (Feb. 2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *arXiv:1502.03167 [cs]*. arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167 (visited on 01/04/2019).

Jabraeil Jamali, Mohammad Ali et al. (2020). "IoT Architecture". en. In: *Towards the Internet of Things*. Series Title: EAI/Springer Innovations in Communication and Computing. Cham: Springer International Publishing, pp. 9–31. ISBN: 978-3-030-18467-4 978-3-030-18468-1. DOI: 10.1007/978-3-030-18468-1_2. URL: http://link.springer.com/10.1007/978-3-030-18468-1_2 (visited on 01/09/2023).

Jaderberg, Max, Andrea Vedaldi, and Andrew Zisserman (May 2014). "Speeding up Convolutional Neural Networks with Low Rank Expansions". In: *arXiv:1405.3866 [cs]*. arXiv: 1405.3866. URL: http://arxiv.org/abs/1405.3866 (visited on 01/04/2019).

Jia, Yangqing et al. (Nov. 2014). "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *Proceedings of the 22nd ACM international conference on Multimedia*. MM '14. New York, NY, USA: Association for Computing Machinery, pp. 675–678. ISBN: 978-1-4503-3063-3. DOI: 10.1145/2647868.2654889. URL: https://doi.org/10.1145/2647868.2654889 (visited on 01/24/2023).

Jiang, Tammy, Jaimie L. Gradus, and Anthony J. Rosellini (Sept. 2020). "Supervised Machine Learning: A Brief Primer". en. In: *Behavior Therapy* 51.5, pp. 675–687. ISSN: 00057894. DOI: 10.1016/j.beth.2020.05.002. URL: https://linkinghub.elsevier.com/retrieve/pii/S0005789420300678 (visited on 01/27/2023).

Jin, Jonghoon, Aysegul Dundar, and Eugenio Culurciello (Dec. 2014). "Flattened Convolutional Neural Networks for Feedforward Acceleration". In: *arXiv:1412.5474 [cs]*. arXiv: 1412.5474. URL: http://arxiv.org/abs/1412.5474 (visited on 07/02/2019).

Johansson, E.M., F.U. Dowla, and D.M. Goodman (Jan. 1991). "BACKPROPAGATION LEARNING FOR MULTILAYER FEED-FORWARD NEURAL NETWORKS USING THE CONJUGATE GRADIENT METHOD". en. In: *International Journal of Neural Systems* 02.04, pp. 291–301. ISSN: 0129-0657, 1793-6462. DOI: 10.1142/S0129065791000261. URL: https://www.worldscientific.com/doi/abs/10.1142/S0129065791000261 (visited on 01/23/2023).

Jouppi, Norman P. et al. (June 2017). "In-datacenter performance analysis of a tensor processing unit". In: *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–12. DOI: 10.1145/3079856.3080246.

Kadiyala, Akhil and Ashok Kumar (Sept. 2018). "Applications of python to evaluate the performance of bagging methods". en. In: *Environmental Progress & Sustainable Energy* 37.5, pp. 1555–1559. ISSN: 1944-7442, 1944-7450. DOI: 10.1002/ep.13018. URL: https://onlinelibrary.wiley.com/doi/10.1002/ep.13018 (visited on 01/16/2023).

Kamalov, Firuz and Ho Hon Leung (Nov. 2020). "Deep learning regularization in imbalanced data". In: *2020 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI)*. Sharjah, United Arab Emirates: IEEE, pp. 1–5. ISBN: 978-1-72817-315-3. DOI: 10.1109/CCCI49893.2020.9256674. URL: https://ieeexplore.ieee.org/document/9256674/ (visited on 10/27/2023).

Karimireddy, Sai Praneeth et al. (Apr. 2021). "SCAFFOLD: Stochastic Controlled Averaging for Federated Learning". In: *arXiv:1910.06378 [cs, math,*

stat]. arXiv: 1910.06378. URL: http://arxiv.org/abs/1910.06378 (visited on 03/18/2022).

Kaur, Jaspinder, Sudeep Yadav, and Harjot Singh Gill (2023). "Internet of Things (IoT) for Sensor-Based Smart Farming: Challenges and Opportunities". en. In: *IoT Based Smart Applications*. Ed. by Nidhi Sindhwani et al. Series Title: EAI/Springer Innovations in Communication and Computing. Cham: Springer International Publishing, pp. 151–164. ISBN: 978-3-031-04523-3 978-3-031-04524-0. DOI: 10.1007/978-3-031-04524-0_9. URL: https://link.springer.com/10.1007/978-3-031-04524-0_9 (visited on 01/03/2023).

Kaur, Jaspinder et al. (2023). "Implementation of IoT in Various Domains". en. In: *IoT Based Smart Applications*. Ed. by Nidhi Sindhwani et al. Series Title: EAI/Springer Innovations in Communication and Computing. Cham: Springer International Publishing, pp. 165–178. ISBN: 978-3-031-04523-3 978-3-031-04524-0. DOI: 10.1007/978-3-031-04524-0_10. URL: https://link.springer.com/10.1007/978-3-031-04524-0_10 (visited on 01/03/2023).

Ketkar, Nikhil (2017). "Stochastic Gradient Descent". en. In: *Deep Learning with Python*. Berkeley, CA: Apress, pp. 113–132. ISBN: 978-1-4842-2765-7 978-1-4842-2766-4. DOI: 10.1007/978-1-4842-2766-4_8. URL: http://link.springer.com/10.1007/978-1-4842-2766-4_8 (visited on 01/23/2023).

Khalil, Ruhul Amin et al. (July 2021). "Deep Learning in the Industrial Internet of Things: Potentials, Challenges, and Emerging Applications". In: *IEEE Internet of Things Journal* 8.14. Conference Name: IEEE Internet of Things Journal, pp. 11016–11040. ISSN: 2327-4662. DOI: 10.1109/JIOT.2021.3051414.

Khan, Latif U. et al. (Oct. 2020). "Edge-Computing-Enabled Smart Cities: A Comprehensive Survey". In: *IEEE Internet of Things Journal* 7.10, pp. 10200–10232. ISSN: 2327-4662, 2372-2541. DOI: 10.1109/JIOT.2020.2987070. URL: https://ieeexplore.ieee.org/document/9063670/ (visited on 03/05/2023).

Khanna, Abhishek and Sanmeet Kaur (Sept. 2020). "Internet of Things (IoT), Applications and Challenges: A Comprehensive Review". en. In: *Wireless Personal Communications* 114.2, pp. 1687–1762. ISSN: 0929-6212, 1572-834X. DOI: 10.1007/s11277-020-07446-4. URL: https://link.springer.com/10.1007/s11277-020-07446-4 (visited on 01/03/2023).

Ko, Albert H.R., Robert Sabourin, and Alceu Souza Britto Jr. (May 2008). "From dynamic classifier selection to dynamic ensemble selection". en. In: *Pattern Recognition* 41.5, pp. 1718–1731. ISSN: 00313203. DOI: 10.1016/j.patcog.2007.10.015. URL: https://linkinghub.elsevier.com/retrieve/pii/S0031320307004499 (visited on 10/26/2023).

Konečný, Jakub et al. (Oct. 2017). "Federated Learning: Strategies for Improving Communication Efficiency". In: *arXiv:1610.05492 [cs]*. arXiv: 1610.05492. URL: http://arxiv.org/abs/1610.05492 (visited on 02/06/2020).

Krawczyk, Bartosz and Bogusław Cyganek (May 2017). "Selecting locally specialised classifiers for one-class classification ensembles". en. In: *Pattern Analysis and Applications* 20.2, pp. 427–439. ISSN: 1433-7541, 1433-755X.

DOI: 10.1007/s10044-015-0505-z. URL: http://link.springer.com/10.1007/s10044-015-0505-z (visited on 11/07/2023).

Krizhevsky, Alex (n.d.). "Learning Multiple Layers of Features from Tiny Images". en. In: (), p. 60.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (visited on 03/05/2019).

Kumbhare, Trupti A and Santosh V Chobe (2014). "An Overview of Association Rule Mining Algorithms". en. In: 5.

Kuncheva, L.I. et al. (Apr. 2003). "Limits on the majority vote accuracy in classifier fusion". In: *Pattern Analysis & Applications* 6.1, pp. 22–31. ISSN: 1433-7541, 1433-755X. DOI: 10.1007/s10044-002-0173-7. URL: http://link.springer.com/10.1007/s10044-002-0173-7 (visited on 11/09/2023).

Kuncheva, Ludmila I (2014). *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons.

Kuncheva, Ludmila I. and Christopher J. Whitaker (2003). "Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy". In: *Machine Learning* 51.2, pp. 181–207. ISSN: 08856125. DOI: 10.1023/A:1022859003006. URL: http://link.springer.com/10.1023/A:1022859003006 (visited on 01/20/2020).

lan, xu, Xiatian Zhu, and Shaogang Gong (2018). "Knowledge Distillation by On-the-Fly Native Ensemble". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., pp. 7517–7527. URL: http://papers.nips.cc/paper/7980-knowledge-distillation-by-on-the-fly-native-ensemble.pdf.

Lane, Nicholas D. et al. (Apr. 2016). "DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices". In: *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. Vienna, Austria: IEEE, pp. 1–12. ISBN: 978-1-5090-0802-5. DOI: 10.1109/IPSN.2016.7460664. URL: http://ieeexplore.ieee.org/document/7460664/ (visited on 03/05/2023).

Latifi, Shahram (May 2022). *ITNG 2022 19th International Conference on Information Technology-New Generations*. en. Google-Books-ID: 6BVuEAAAQBAJ. Springer Nature. ISBN: 978-3-030-97652-1.

Lauret, P., E. Fock, and T.A. Mara (Mar. 2006). "A Node Pruning Algorithm Based on a Fourier Amplitude Sensitivity Test Method". en. In: *IEEE Transactions on Neural Networks* 17.2, pp. 273–293. ISSN: 1045-9227. DOI: 10.1109/TNN.2006.871707. URL: http://ieeexplore.ieee.org/document/1603616/ (visited on 01/03/2019).

Lazarevic, A. and Z. Obradovic (2001). "Effective pruning of neural network classifier ensembles". In: *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*. Vol. 2. Washington, DC, USA: IEEE, pp. 796–801. ISBN: 978-0-7803-7044-9. DOI: 10.1109/IJCNN.2001.939461. URL: http://ieeexplore.ieee.org/document/939461/ (visited on 11/07/2023).

Lecun, Y. et al. (Nov. 1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11. Conference Name: Proceedings of the IEEE, pp. 2278–2324. ISSN: 1558-2256. DOI: 10.1109/5.726791.

LeCun, Yann, John S. Denker, and Sara A. Solla (1990). "Optimal Brain Damage". In: *Advances in Neural Information Processing Systems 2*. Ed. by D. S. Touretzky. Morgan-Kaufmann, pp. 598–605. URL: http://papers.nips.cc/paper/250-optimal-brain-damage.pdf.

Leong, Mei Chee et al. (Jan. 2020). "Semi-CNN Architecture for Effective Spatio-Temporal Learning in Action Recognition". en. In: *Applied Sciences* 10.2, p. 557. ISSN: 2076-3417. DOI: 10.3390/app10020557. URL: https://www.mdpi.com/2076-3417/10/2/557 (visited on 06/24/2022).

Levie, Ron et al. (Jan. 2019). "CayleyNets: Graph Convolutional Neural Networks With Complex Rational Spectral Filters". en. In: *IEEE Transactions on Signal Processing* 67.1, pp. 97–109. ISSN: 1053-587X, 1941-0476. DOI: 10.1109/TSP.2018.2879624. URL: https://ieeexplore.ieee.org/document/8521593/ (visited on 06/13/2023).

Li, Qin et al. (May 2021a). "Directed Acyclic Graph Neural Network for Human Motion Prediction". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. Xi'an, China: IEEE, pp. 3197–3204. ISBN: 978-1-72819-077-8. DOI: 10.1109/ICRA48506.2021.9561540. URL: https://ieeexplore.ieee.org/document/9561540/ (visited on 01/22/2023).

Li, Tian et al. (May 2020a). "Federated Learning: Challenges, Methods, and Future Directions". In: *IEEE Signal Processing Magazine* 37.3, pp. 50–60. ISSN: 1053-5888, 1558-0792. DOI: 10.1109/MSP.2020.2975749. URL: https://ieeexplore.ieee.org/document/9084352/ (visited on 06/28/2022).

Li, Wei and Andrew McCallum (2006). "Pachinko allocation: DAG-structured mixture models of topic correlations". en. In: *Proceedings of the 23rd international conference on Machine learning - ICML '06*. Pittsburgh, Pennsylvania: ACM Press, pp. 577–584. ISBN: 978-1-59593-383-6. DOI: 10.1145/1143844.1143917. URL: http://portal.acm.org/citation.cfm?doid=1143844.1143917 (visited on 12/12/2022).

Li, Xiang et al. (June 2020b). *On the Convergence of FedAvg on Non-IID Data*. arXiv:1907.02189 [cs, math, stat]. URL: http://arxiv.org/abs/1907.02189 (visited on 02/23/2023).

Li, Xiaoxiao et al. (May 2021b). "FedBN: Federated Learning on Non-IID Features via Local Batch Normalization". In: *arXiv:2102.07623 [cs]*. arXiv: 2102.07623. URL: http://arxiv.org/abs/2102.07623 (visited on 03/18/2022).

Li, Yiwei et al. (Jan. 2023). *Differentially Private Federated Clustering over Non-IID Data*. arXiv:2301.00955 [cs]. DOI: 10.48550/arXiv.2301.00955. URL: http://arxiv.org/abs/2301.00955 (visited on 02/22/2023).

Li, Yiying et al. (Jan. 2021c). "FedH2L: Federated Learning with Model and Statistical Heterogeneity". In: *arXiv:2101.11296 [cs]*. arXiv: 2101.11296. URL: http://arxiv.org/abs/2101.11296 (visited on 04/22/2021).

Liang, Paul Pu et al. (2020). "Think Locally, Act Globally: Federated Learning with Local and Global Representations". In: Publisher: arXiv Version Number: 3. DOI: 10.48550/ARXIV.2001.01523. URL: https://arxiv.org/abs/2001.01523 (visited on 06/30/2022).

Lin, Min, Qiang Chen, and Shuicheng Yan (Dec. 2013). "Network In Network". In: *arXiv:1312.4400 [cs]*. arXiv: 1312.4400. URL: http://arxiv.org/abs/1312.4400 (visited on 03/05/2019).

Litjens, Geert et al. (Dec. 2017). "A survey on deep learning in medical image analysis". en. In: *Medical Image Analysis* 42, pp. 60–88. ISSN: 13618415. DOI: 10.1016/j.media.2017.07.005. URL: https://linkinghub.elsevier.com/retrieve/pii/S1361841517301135 (visited on 05/22/2023).

Liu, Yuying et al. (2020). "An Anchor-Free Convolutional Neural Network for Real-Time Surgical Tool Detection in Robot-Assisted Surgery". In: *IEEE Access* 8, pp. 78193–78201. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2989807. URL: https://ieeexplore.ieee.org/document/9076706/ (visited on 02/28/2023).

Lui, Andrew Kwok-Fai, Yin-Hei Chan, and Man-Fai Leung (Dec. 2021). "Modelling of Destinations for Data-driven Pedestrian Trajectory Prediction in Public Buildings". In: *2021 IEEE International Conference on Big Data (Big Data)*. Orlando, FL, USA: IEEE, pp. 1709–1717. ISBN: 978-1-66543-902-2. DOI: 10.1109/BigData52589.2021.9671813. URL: https://ieeexplore.ieee.org/document/9671813/ (visited on 02/18/2023).

— (Apr. 2022). "Modelling of Pedestrian Movements near an Amenity in Walkways of Public Buildings". In: *2022 8th International Conference on Control, Automation and Robotics (ICCAR)*. Xiamen, China: IEEE, pp. 394–400. ISBN: 978-1-66548-116-8. DOI: 10.1109/ICCAR55106.2022.9782667. URL: https://ieeexplore.ieee.org/document/9782667/ (visited on 02/18/2023).

Luo, Jian-Hao and Jianxin Wu (June 2017). "An Entropy-based Pruning Method for CNN Compression". In: *arXiv:1706.05791 [cs]*. arXiv: 1706.05791. URL: http://arxiv.org/abs/1706.05791 (visited on 09/19/2020).

Luo, Xiao, Weilai Chi, and Minghua Deng (Nov. 2019). "Deepprune: Learning Efficient and Interpretable Convolutional Networks Through Weight Pruning for Predicting DNA-Protein Binding". In: *Frontiers in Genetics* 10, p. 1145. ISSN: 1664-8021. DOI: 10.3389/fgene.2019.01145. URL: https://www.frontiersin.org/article/10.3389/fgene.2019.01145/full (visited on 03/10/2023).

Mahdavinejad, Mohammad Saeid et al. (Aug. 2018). "Machine learning for internet of things data analysis: a survey". en. In: *Digital Communications and Networks* 4.3, pp. 161–175. ISSN: 23528648. DOI: 10.1016/j.dcan.2017.10.002. URL: https://linkinghub.elsevier.com/retrieve/pii/S235286481730247X (visited on 12/21/2019).

Maheshwari, Sumit et al. (Apr. 2019). "EdgeDrive: Supporting Advanced Driver Assistance Systems using Mobile Edge Clouds Networks". In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. Paris, France: IEEE, pp. 1–6. ISBN: 978-1-72811-878-9. DOI: 10.1109/INFCOMW.2019.8845256. URL: https://ieeexplore.ieee.org/document/8845256/ (visited on 02/28/2023).

Maldonado, F.J. and M.T. Manry (2002). "Optimal pruning of feedforward neural networks based upon the Schmidt procedure". In: *Conference Record of the Thirty-Sixth Asilomar Conference on Signals, Systems and Computers,*

2002. Vol. 2. Pacific Grove, CA, USA: IEEE, pp. 1024–1028. ISBN: 978-0-7803-7576-5. DOI: 10.1109/ACSSC.2002.1196939. URL: http://ieeexplore.ieee.org/document/1196939/ (visited on 01/22/2023).

Mansour, Yishay et al. (July 2020). "Three Approaches for Personalization with Applications to Federated Learning". In: *arXiv:2002.10619 [cs, stat]*. arXiv: 2002.10619. URL: http://arxiv.org/abs/2002.10619 (visited on 04/16/2021).

Marikyan, Davit, Savvas Papagiannidis, and Eleftherios Alamanos (Jan. 2019). "A systematic review of the smart home literature: A user perspective". en. In: *Technological Forecasting and Social Change* 138, pp. 139–154. ISSN: 0040-1625. DOI: 10.1016/j.techfore.2018.08.015. URL: https://www.sciencedirect.com/science/article/pii/S0040162517315676 (visited on 02/28/2023).

McMahan, Brendan et al. (Apr. 2017). "Communication-Efficient Learning of Deep Networks from Decentralized Data". en. In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. ISSN: 2640-3498. PMLR, pp. 1273–1282. URL: https://proceedings.mlr.press/v54/mcmahan17a.html (visited on 02/23/2023).

McMahan, H. Brendan et al. (Feb. 2018). "Learning Differentially Private Recurrent Language Models". In: *arXiv:1710.06963 [cs]*. arXiv: 1710.06963. URL: http://arxiv.org/abs/1710.06963 (visited on 03/17/2022).

Mei, Guangxu et al. (Dec. 2019). "SGNN: A Graph Neural Network Based Federated Learning Approach by Hiding Structure". In: *2019 IEEE International Conference on Big Data (Big Data)*. Los Angeles, CA, USA: IEEE, pp. 2560–2568. ISBN: 978-1-72810-858-2. DOI: 10.1109/BigData47090.2019.9005983. URL: https://ieeexplore.ieee.org/document/9005983/ (visited on 07/05/2022).

Melville, P. and R. J. Mooney (2005). "Diverse ensembles for active learning". In: *Proceedings of the 22nd international conference on Machine learning*. ACM, pp. 584–591.

Mienye, Ibomoiye Domor and Yanxia Sun (2022). "A Survey of Ensemble Learning: Concepts, Algorithms, Applications, and Prospects". In: *IEEE Access* 10, pp. 99129–99149. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3207287. URL: https://ieeexplore.ieee.org/document/9893798/ (visited on 01/18/2023).

Mills, Jed, Jia Hu, and Geyong Min (July 2020). "Communication-Efficient Federated Learning for Wireless Edge Intelligence in IoT". In: *IEEE Internet of Things Journal* 7.7. Conference Name: IEEE Internet of Things Journal, pp. 5986–5994. ISSN: 2327-4662. DOI: 10.1109/JIOT.2019.2956615.

Mittal, Sparsh (Feb. 2016). "A Survey of Techniques for Architecting and Managing Asymmetric Multicore Processors". In: *ACM Computing Surveys* 48.3, 45:1–45:38. ISSN: 0360-0300. DOI: 10.1145/2856125. URL: https://dl.acm.org/doi/10.1145/2856125 (visited on 04/18/2023).

— (Oct. 2018). "A survey of FPGA-based accelerators for convolutional neural networks". en. In: *Neural Computing and Applications*. ISSN: 0941-0643, 1433-3058. DOI: 10.1007/s00521-018-3761-1. URL: http://link.springer.com/10.1007/s00521-018-3761-1 (visited on 01/02/2019).

Mocanu, Decebal Constantin et al. (June 2018). "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network

science". en. In: *Nature Communications* 9.1. Number: 1 Publisher: Nature Publishing Group, p. 2383. ISSN: 2041-1723. DOI: 10.1038/s41467-018-04316-3. URL: https://www.nature.com/articles/s41467-018-04316-3 (visited on 04/11/2023).

Mocnej, Jozef et al. (2018). "Decentralised IoT Architecture for Efficient Resources Utilisation". en. In: *IFAC-PapersOnLine* 51.6, pp. 168–173. ISSN: 24058963. DOI: 10.1016/j.ifacol.2018.07.148. URL: https://linkinghub.elsevier.com/retrieve/pii/S2405896318308942 (visited on 01/09/2023).

Mohammadi, Mehdi and Ala Al-Fuqaha (Feb. 2018). "Enabling Cognitive Smart Cities Using Big Data and Machine Learning: Approaches and Challenges". In: *IEEE Communications Magazine* 56.2, pp. 94–101. ISSN: 0163-6804. DOI: 10.1109/MCOM.2018.1700298. URL: http://ieeexplore.ieee.org/document/8291121/ (visited on 06/16/2022).

Molchanov, Pavlo et al. (June 2019). "Importance Estimation for Neural Network Pruning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Moore, Robert C. and John DeNero (2011). "L1 and L2 Regularization for Multiclass Hinge Loss Models". In: *Symposium on Machine Learning in Speech and Natural Language Processing*. URL: http://www.ttic.edu/sigml/symposium2011/papers/Moore+DeNero_Regularization.pdf.

Morafah, Mahdi et al. (Aug. 2022). *FLIS: Clustered Federated Learning via Inference Similarity for Non-IID Data Distribution*. arXiv:2208.09754 [cs]. DOI: 10.48550/arXiv.2208.09754. URL: http://arxiv.org/abs/2208.09754 (visited on 02/20/2023).

Mrabet, Hichem et al. (June 2020). "A Survey of IoT Security Based on a Layered Architecture of Sensing and Data Analysis". en. In: *Sensors* 20.13, p. 3625. ISSN: 1424-8220. DOI: 10.3390/s20133625. URL: https://www.mdpi.com/1424-8220/20/13/3625 (visited on 01/10/2023).

Navani, Deepika, Sanjeev Jain, and Maninder Singh Nehra (Dec. 2017). "The Internet of Things (IoT): A Study of Architectural Elements". In: *2017 13th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*. Jaipur, India: IEEE, pp. 473–478. ISBN: 978-1-5386-4283-2. DOI: 10.1109/SITIS.2017.83. URL: http://ieeexplore.ieee.org/document/8334789/ (visited on 01/11/2023).

Nguyen, Dinh C. et al. (2021). "Federated Learning for Internet of Things: A Comprehensive Survey". In: *IEEE Communications Surveys & Tutorials* 23.3. arXiv:2104.07914 [eess], pp. 1622–1658. ISSN: 1553-877X, 2373-745X. DOI: 10.1109/COMST.2021.3075439. URL: http://arxiv.org/abs/2104.07914 (visited on 10/25/2022).

Nickolls, John and William J. Dally (Mar. 2010). "The GPU Computing Era". In: *IEEE Micro* 30.2. Conference Name: IEEE Micro, pp. 56–69. ISSN: 1937-4143. DOI: 10.1109/MM.2010.41.

Nielsen, Andreas Brinch and Lars Kai Hansen (June 2008). "Structure learning by pruning in independent component analysis". en. In: *Neurocomputing* 71.10-12, pp. 2281–2290. ISSN: 09252312. DOI: 10.1016/j.neucom.2007.09.016. URL: https://linkinghub.elsevier.com/retrieve/pii/S0925231208000994 (visited on 01/02/2019).

Nowlan, Steven J. and Geoffrey E. Hinton (July 1992). "Simplifying neural networks by soft weight-sharing". In: *Neural Computation* 4.4, pp. 473–

493. ISSN: 0899-7667. DOI: 10.1162/neco.1992.4.4.473. URL: http://dl.acm.org.ezproxy.bcu.ac.uk/citation.cfm?id=148167.148169 (visited on 03/22/2019).

Ordoñez-Cardenas, Ernesto and Rene de J. Romero-Troncoso (2008). "Mlp neural network and on-line backpropagation learning implementation in a low-cost fpga". en. In: *Proceedings of the 18th ACM Great Lakes symposium on VLSI - GLSVLSI '08*. Orlando, Florida, USA: ACM Press, p. 333. ISBN: 978-1-59593-999-9. DOI: 10.1145/1366110.1366188. URL: http://portal.acm.org/citation.cfm?doid=1366110.1366188 (visited on 01/05/2019).

Ouyang, Zhiyou et al. (2018). "Multi-View Stacking Ensemble for Power Consumption Anomaly Detection in the Context of Industrial Internet of Things". In: *IEEE Access* 6, pp. 9623–9631. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2805908. URL: http://ieeexplore.ieee.org/document/8291600/ (visited on 01/19/2023).

Oza, Nikunj C. and Kagan Tumer (Jan. 2008). "Classifier ensembles: Select real-world applications". en. In: *Information Fusion* 9.1, pp. 4–20. ISSN: 15662535. DOI: 10.1016/j.inffus.2007.07.002. URL: https://linkinghub.elsevier.com/retrieve/pii/S1566253507000620 (visited on 11/09/2023).

Pan, Sinno Jialin and Qiang Yang (Oct. 2010). "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10, pp. 1345–1359. ISSN: 1041-4347. DOI: 10.1109/TKDE.2009.191. URL: http://ieeexplore.ieee.org/document/5288526/ (visited on 07/01/2022).

Panagakis, Yannis et al. (May 2021). "Tensor Methods in Computer Vision and Deep Learning". In: *Proceedings of the IEEE* 109.5, pp. 863–890. ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2021.3074329. URL: https://ieeexplore.ieee.org/document/9420085/ (visited on 05/29/2023).

Panwar, Nisha et al. (May 2019). *Smart Home Survey on Security and Privacy*. arXiv:1904.05476 [cs]. URL: http://arxiv.org/abs/1904.05476 (visited on 02/28/2023).

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (May 2013). "On the difficulty of training recurrent neural networks". en. In: *Proceedings of the 30th International Conference on Machine Learning*. ISSN: 1938-7228. PMLR, pp. 1310–1318. URL: https://proceedings.mlr.press/v28/pascanu13.html (visited on 04/13/2023).

Paszke, Adam et al. (Dec. 2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. arXiv:1912.01703 [cs, stat]. DOI: 10.48550/arXiv.1912.01703. URL: http://arxiv.org/abs/1912.01703 (visited on 01/24/2023).

Peemen, Maurice et al. (Oct. 2013). "Memory-centric accelerator design for Convolutional Neural Networks". In: *2013 IEEE 31st International Conference on Computer Design (ICCD)*. Asheville, NC, USA: IEEE, pp. 13–19. ISBN: 978-1-4799-2987-0. DOI: 10.1109/ICCD.2013.6657019. URL: http://ieeexplore.ieee.org/document/6657019/ (visited on 01/05/2019).

Peng, Yaohao and Mateus Hiro Nagata (Oct. 2020). "An empirical overview of nonlinearity and overfitting in machine learning using COVID-19 data". en. In: *Chaos, Solitons & Fractals* 139, p. 110055. ISSN: 0960-0779. DOI: 10.1016/j.chaos.2020.110055. URL: https://www.sciencedirect.com/science/article/pii/S0960077920304525 (visited on 05/29/2023).

Pham, Phi-Hung et al. (Aug. 2012). "NeuFlow: Dataflow vision processing system-on-a-chip". In: *2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS)*. Boise, ID, USA: IEEE, pp. 1044–1047. ISBN: 978-1-4673-2527-1 978-1-4673-2526-4 978-1-4673-2525-7. DOI: 10.1109/MWSCAS.2012.6292202. URL: http://ieeexplore.ieee.org/document/6292202/ (visited on 01/05/2019).

Powers, David M. W. (2020). "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation". In: Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.2010.16061. URL: https://arxiv.org/abs/2010.16061 (visited on 01/31/2023).

Prabhakar, Shruthi (2017). "NETWORK SECURITY IN DIGITALIZATION: ATTACKS AND DEFENCE". en. In: 5.

Products (n.d.). en-us. URL: https://coral.ai/products/ (visited on 04/18/2023).

Puthal, Deepak et al. (May 2016). "Threats to Networking Cloud and Edge Datacenters in the Internet of Things". In: *IEEE Cloud Computing* 3.3, pp. 64–71. ISSN: 2325-6095. DOI: 10.1109/MCC.2016.63. URL: http://ieeexplore.ieee.org/document/7503493/ (visited on 01/10/2023).

Qiang, Fu, Hu Shang-xu, and Zhao Sheng-ying (May 2005). "Clustering-based selective neural network ensemble". en. In: *Journal of Zhejiang University-SCIENCE A* 6.5, pp. 387–392. ISSN: 1673-565X, 1862-1775. DOI: 10.1631/jzus.2005.A0387. URL: http://link.springer.com/10.1631/jzus.2005.A0387 (visited on 11/07/2023).

Qiu, Huaiyu et al. (2020). "Real-Time Iris Tracking Using Deep Regression Networks for Robotic Ophthalmic Surgery". In: *IEEE Access* 8. Conference Name: IEEE Access, pp. 50648–50658. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2980005.

Rashma, B. M. et al. (2021). "Handling Heterogeneity in an IoT Infrastructure". en. In: *Advances in Machine Learning and Computational Intelligence*. Ed. by Srikanta Patnaik, Xin-She Yang, and Ishwar K. Sethi. Series Title: Algorithms for Intelligent Systems. Singapore: Springer Singapore, pp. 635–643. ISBN: 9789811552427 9789811552434. DOI: 10.1007/978-981-15-5243-4_60. URL: http://link.springer.com/10.1007/978-981-15-5243-4_60 (visited on 05/29/2023).

Ray, P.P. (July 2018). "A survey on Internet of Things architectures". en. In: *Journal of King Saud University - Computer and Information Sciences* 30.3, pp. 291–319. ISSN: 13191578. DOI: 10.1016/j.jksuci.2016.10.003. URL: https://linkinghub.elsevier.com/retrieve/pii/S1319157816300799 (visited on 01/09/2023).

Reed, R. (Sept. 1993). "Pruning algorithms-a survey". In: *IEEE Transactions on Neural Networks* 4.5, pp. 740–747. ISSN: 10459227. DOI: 10.1109/72.248452. URL: http://ieeexplore.ieee.org/document/248452/ (visited on 01/02/2019).

Rice, Kenneth L., Tarek M. Taha, and Christopher N. Vutsinas (Jan. 2009). "Scaling analysis of a neocortex inspired cognitive model on the Cray XD1". en. In: *The Journal of Supercomputing* 47.1, pp. 21–43. ISSN: 0920-8542, 1573-0484. DOI: 10.1007/s11227-008-0195-z. URL: http://link.springer.com/10.1007/s11227-008-0195-z (visited on 01/05/2019).

Rigamonti, Roberto et al. (June 2013). "Learning Separable Filters". In: *2013 IEEE Conference on Computer Vision and Pattern Recognition*. Portland, OR,

USA: IEEE, pp. 2754–2761. ISBN: 978-0-7695-4989-7. DOI: 10.1109/CVPR.2013.355. URL: http://ieeexplore.ieee.org/document/6619199/ (visited on 01/23/2023).

Rodríguez, Eva, Beatriz Otero, and Ramon Canal (Jan. 2023). "A Survey of Machine and Deep Learning Methods for Privacy Protection in the Internet of Things". en. In: *Sensors* 23.3, p. 1252. ISSN: 1424-8220. DOI: 10.3390/s23031252. URL: https://www.mdpi.com/1424-8220/23/3/1252 (visited on 05/29/2023).

Rokach, Lior (2010). "A survey of Clustering Algorithms". en. In: *Data Mining and Knowledge Discovery Handbook*. Ed. by Oded Maimon and Lior Rokach. Boston, MA: Springer US, pp. 269–298. ISBN: 978-0-387-09823-4. DOI: 10.1007/978-0-387-09823-4_14. URL: https://doi.org/10.1007/978-0-387-09823-4_14 (visited on 02/02/2023).

Romero, Adriana et al. (Dec. 2014). "FitNets: Hints for Thin Deep Nets". en. In: *arXiv:1412.6550 [cs]*. arXiv: 1412.6550. URL: http://arxiv.org/abs/1412.6550 (visited on 10/09/2018).

Roodschild, Matías, Jorge Gotay Sardiñas, and Adrián Will (Dec. 2020). "A new approach for the vanishing gradient problem on sigmoid activation". en. In: *Progress in Artificial Intelligence* 9.4, pp. 351–360. ISSN: 2192-6352, 2192-6360. DOI: 10.1007/s13748-020-00218-y. URL: https://link.springer.com/10.1007/s13748-020-00218-y (visited on 01/22/2023).

Sagi, Omer and Lior Rokach (July 2018). "Ensemble learning: A survey". en. In: *WIREs Data Mining and Knowledge Discovery* 8.4. ISSN: 1942-4787, 1942-4795. DOI: 10.1002/widm.1249. URL: https://onlinelibrary.wiley.com/doi/10.1002/widm.1249 (visited on 01/16/2023).

Sammut, Claude and Geoffrey I. Webb, eds. (2010). *Encyclopedia of Machine Learning*. en. Boston, MA: Springer US. ISBN: 978-0-387-30768-8 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8. URL: http://link.springer.com/10.1007/978-0-387-30164-8 (visited on 01/31/2023).

Sandler, Mark et al. (Jan. 2018). "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *arXiv:1801.04381 [cs]*. arXiv: 1801.04381. URL: http://arxiv.org/abs/1801.04381 (visited on 07/08/2019).

Sarveniazi, Alireza (Mar. 2014). "An Actual Survey of Dimensionality Reduction". en. In: *American Journal of Computational Mathematics* 2014. Publisher: Scientific Research Publishing. ISSN: 2161-1211. DOI: 10.4236/ajcm.2014.42006. URL: http://www.scirp.org/journal/PaperInformation.aspx?PaperID=43977 (visited on 02/02/2023).

Sattler, Felix, Klaus-Robert Müller, and Wojciech Samek (Oct. 2019). "Clustered Federated Learning: Model-Agnostic Distributed Multi-Task Optimization under Privacy Constraints". In: *arXiv:1910.01991 [cs, stat]*. arXiv: 1910.01991. URL: http://arxiv.org/abs/1910.01991 (visited on 04/15/2021).

— (Aug. 2021). "Clustered Federated Learning: Model-Agnostic Distributed Multitask Optimization Under Privacy Constraints". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.8. Conference Name: IEEE Transactions on Neural Networks and Learning Systems, pp. 3710–3722. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2020.3015958.

Sau, Bharat Bhusan and Vineeth N. Balasubramanian (Oct. 2016). "Deep Model Compression: Distilling Knowledge from Noisy Teachers". In: *arXiv:1610.09650*

[cs]. arXiv: 1610.09650. URL: http://arxiv.org/abs/1610.09650 (visited on 03/05/2019).

Schaffer, J.D., D. Whitley, and L.J. Eshelman (1992). "Combinations of genetic algorithms and neural networks: a survey of the state of the art". In: *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*. Baltimore, MD, USA: IEEE Comput. Soc. Press, pp. 1–37. ISBN: 978-0-8186-2787-3. DOI: 10.1109/COGANN.1992.273950. URL: http://ieeexplore.ieee.org/document/273950/ (visited on 02/22/2019).

Sekaran, Ramesh et al. (2020). "Survival Study on Blockchain Based 6G-Enabled Mobile Edge Computation for IoT Automation". In: *IEEE Access* 8, pp. 143453–143463. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3013946. URL: https://ieeexplore.ieee.org/document/9159552/ (visited on 03/02/2023).

Sepasgozar, Samad M. E. et al. (May 2019). "Implementing citizen centric technology in developing smart cities: A model for predicting the acceptance of urban technologies". en. In: *Technological Forecasting and Social Change*. Understanding Smart Cities: Innovation ecosystems, technological advancements, and societal challenges 142, pp. 105–116. ISSN: 0040-1625. DOI: 10.1016/j.techfore.2018.09.012. URL: https://www.sciencedirect.com/science/article/pii/S004016251731870X (visited on 02/28/2023).

Setiono, Rudy (Jan. 1997). "A Penalty-Function Approach for Pruning Feedforward Neural Networks". en. In: *Neural Computation* 9.1, pp. 185–204. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco.1997.9.1.185. URL: http://www.mitpressjournals.org/doi/10.1162/neco.1997.9.1.185 (visited on 01/02/2019).

Sharma, Siddharth, Simone Sharma, and Anidhya Athaiya (May 2020). "ACTIVATION FUNCTIONS IN NEURAL NETWORKS". en. In: *International Journal of Engineering Applied Sciences and Technology* 04.12, pp. 310–316. ISSN: 24552143. DOI: 10.33564/IJEAST.2020.v04i12.054. URL: https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf (visited on 01/22/2023).

Shi, Naichen et al. (July 2021a). *Fed-ensemble: Improving Generalization through Model Ensembling in Federated Learning*. arXiv:2107.10663 [cs, stat]. URL: http://arxiv.org/abs/2107.10663 (visited on 07/04/2022).

Shi, Qiongfeng et al. (Sept. 2020). "Deep learning enabled smart mats as a scalable floor monitoring system". en. In: *Nature Communications* 11.1. Number: 1 Publisher: Nature Publishing Group, p. 4609. ISSN: 2041-1723. DOI: 10.1038/s41467-020-18471-z. URL: https://www.nature.com/articles/s41467-020-18471-z (visited on 02/28/2023).

Shi, Qiongfeng et al. (Nov. 2021b). "Artificial Intelligence of Things (AIoT) Enabled Floor Monitoring System for Smart Home Applications". In: *ACS Nano* 15.11. Publisher: American Chemical Society, pp. 18312–18326. ISSN: 1936-0851. DOI: 10.1021/acsnano.1c07579. URL: https://doi.org/10.1021/acsnano.1c07579 (visited on 02/28/2023).

Shin, MyungJae et al. (June 2020). *XOR Mixup: Privacy-Preserving Data Augmentation for One-Shot Federated Learning*. arXiv:2006.05148 [cs, eess]. URL: http://arxiv.org/abs/2006.05148 (visited on 02/20/2023).

Shu, Jiangang et al. (Feb. 2023). "Clustered Federated Multitask Learning on Non-IID Data With Enhanced Privacy". In: *IEEE Internet of Things Journal* 10.4. Conference Name: IEEE Internet of Things Journal, pp. 3453–3467. ISSN: 2327-4662. DOI: 10.1109/JIOT.2022.3228893.

Si, Jiong, Sarah L. Harris, and Evangelos Yfantis (Nov. 2018). "A Dynamic ReLU on Neural Network". In: *2018 IEEE 13th Dallas Circuits and Systems Conference (DCAS)*, pp. 1–6. DOI: 10.1109/DCAS.2018.8620116.

Siddiqui, Shoaib Ahmed et al. (2019). "TSViz: Demystification of Deep Learning Models for Time-Series Analysis". en. In: *IEEE Access* 7, pp. 67027–67040. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2912823. URL: https://ieeexplore.ieee.org/document/8695734/ (visited on 05/29/2023).

Siebel, Nils T., Jonas Botel, and Gerald Sommer (June 2009). "Efficient neural network pruning during neuro-evolution". In: *2009 International Joint Conference on Neural Networks*. Atlanta, Ga, USA: IEEE, pp. 2920–2927. ISBN: 978-1-4244-3548-7. DOI: 10.1109/IJCNN.2009.5179035. URL: http://ieeexplore.ieee.org/document/5179035/ (visited on 02/20/2019).

Siebel, Nils T. and Gerald Sommer (Oct. 2007). "Evolutionary reinforcement learning of artificial neural networks". In: *International Journal of Hybrid Intelligent Systems* 4.3. Ed. by M. Köppen and R. Weber, pp. 171–183. ISSN: 18758819, 14485869. DOI: 10.3233/HIS-2007-4304. (Visited on 02/22/2019).

SIfre, Laurent and Stéphane Mallat (Mar. 2014). "Rigid-Motion Scattering for Texture Classification". In: *arXiv:1403.1687 [cs]*. arXiv: 1403.1687. URL: http://arxiv.org/abs/1403.1687 (visited on 01/04/2019).

Simonyan, Karen and Andrew Zisserman (Apr. 2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv:1409.1556 [cs]. DOI: 10.48550/arXiv.1409.1556. URL: http://arxiv.org/abs/1409.1556 (visited on 03/10/2023).

Singh, Amanpreet, Narina Thakur, and Aakanksha Sharma (Mar. 2016). "A review of supervised machine learning algorithms". In: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 1310–1315.

Smith, Virginia et al. (Feb. 2018). "Federated Multi-Task Learning". In: *arXiv:1705.10467 [cs, stat]*. arXiv: 1705.10467. URL: http://arxiv.org/abs/1705.10467 (visited on 04/23/2021).

Sodhro, Ali Hassan et al. (Apr. 2018). "5G-Based Transmission Power Control Mechanism in Fog Computing for Internet of Things Devices". en. In: *Sustainability* 10.4. Number: 4 Publisher: Multidisciplinary Digital Publishing Institute, p. 1258. ISSN: 2071-1050. DOI: 10.3390/su10041258. URL: https://www.mdpi.com/2071-1050/10/4/1258 (visited on 03/10/2023).

Sodhro, Ali Hassan et al. (Dec. 2019). "Quality of Service Optimization in an IoT-Driven Intelligent Transportation System". In: *IEEE Wireless Communications* 26.6, pp. 10–17. ISSN: 1536-1284, 1558-0687. DOI: 10.1109/MWC.001.1900085. URL: https://ieeexplore.ieee.org/document/8938178/ (visited on 03/10/2023).

Sodhro, Ali Hassan et al. (June 2021). "Toward Convergence of AI and IoT for Energy-Efficient Communication in Smart Homes". In: *IEEE Internet of Things Journal* 8.12, pp. 9664–9671. ISSN: 2327-4662, 2372-2541. DOI: 10.1109/JIOT.2020.3023667. URL: https://ieeexplore.ieee.org/document/9195506/ (visited on 03/10/2023).

Sood, Sandeep K. and Isha Mahajan (Apr. 2018). "A Fog-Based Healthcare Framework for Chikungunya". In: *IEEE Internet of Things Journal* 5.2. Conference Name: IEEE Internet of Things Journal, pp. 794–801. ISSN: 2327-4662. DOI: 10.1109/JIOT.2017.2768407.

Southwest Jiaotong University, China et al. (Apr. 2015). "SUPERVISED MACHINE LEARNING APPROACHES: A SURVEY". en. In: *ICTACT Journal on Soft Computing* 05.03, pp. 946–952. ISSN: 09766561, 22296956. DOI: 10.21917/ijsc.2015.0133. URL: http://ictactjournals.in/ArticleDetails.aspx?id=1785 (visited on 01/27/2023).

Souza, Pedro Lopes de Lopes de, Wanderley Lopes de Lopes de Souza, and Ricardo Rodrigues Ciferri (2022). "Semantic Interoperability in the Internet of Things: A Systematic Literature Review". en. In: *ITNG 2022 19th International Conference on Information Technology-New Generations*. Ed. by Shahram Latifi. Vol. 1421. Series Title: Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, pp. 333–340. ISBN: 978-3-030-97651-4 978-3-030-97652-1. DOI: 10.1007/978-3-030-97652-1_40. URL: https://link.springer.com/10.1007/978-3-030-97652-1_40 (visited on 03/05/2023).

Sovacool, Benjamin K. and Dylan D. Furszyfer Del Rio (Mar. 2020). "Smart home technologies in Europe: A critical review of concepts, benefits, risks and policies". en. In: *Renewable and Sustainable Energy Reviews* 120, p. 109663. ISSN: 1364-0321. DOI: 10.1016/j.rser.2019.109663. URL: https://www.sciencedirect.com/science/article/pii/S1364032119308688 (visited on 02/28/2023).

Stanley, Kenneth O. and Risto Miikkulainen (June 2002). "Evolving Neural Networks through Augmenting Topologies". en. In: *Evolutionary Computation* 10.2, pp. 99–127. ISSN: 1063-6560, 1530-9304. DOI: 10.1162/106365602320169811. URL: http://www.mitpressjournals.org/doi/10.1162/106365602320169811 (visited on 02/22/2019).

Sun, Haifeng et al. (Nov. 2020). "Toward Communication-Efficient Federated Learning in the Internet of Things With Edge Computing". In: *IEEE Internet of Things Journal* 7.11. Conference Name: IEEE Internet of Things Journal, pp. 11053–11067. ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.2994596.

Szegedy, Christian et al. (Sept. 2014). "Going Deeper with Convolutions". In: *arXiv:1409.4842 [cs]*. arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842 (visited on 07/08/2019).

Szegedy, Christian et al. (Dec. 2015). "Rethinking the Inception Architecture for Computer Vision". en. In: *arXiv:1512.00567 [cs]*. arXiv: 1512.00567. URL: http://arxiv.org/abs/1512.00567 (visited on 07/29/2019).

Szegedy, Christian et al. (Feb. 2016). "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". In: *arXiv:1602.07261 [cs]*. arXiv: 1602.07261. URL: http://arxiv.org/abs/1602.07261 (visited on 01/04/2019).

Talebpour, Alireza and Hani S. Mahmassani (Oct. 2016). "Influence of connected and autonomous vehicles on traffic flow stability and throughput". en. In: *Transportation Research Part C: Emerging Technologies* 71, pp. 143–163. ISSN: 0968090X. DOI: 10.1016/j.trc.2016.07.007. URL: https://linkinghub.elsevier.com/retrieve/pii/S0968090X16301140 (visited on 02/28/2023).

Tan, Mingxing and Quoc V. Le (Nov. 2019). "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *arXiv:1905.11946 [cs, stat]*. arXiv: 1905.11946. URL: http://arxiv.org/abs/1905.11946 (visited on 05/06/2020).

Tan, Mingxing et al. (July 2018). "MnasNet: Platform-Aware Neural Architecture Search for Mobile". en. In: *arXiv:1807.11626 [cs]*. arXiv: 1807.11626. URL: http://arxiv.org/abs/1807.11626 (visited on 09/12/2018).

Tang, E. K., P. N. Suganthan, and X. Yao (Oct. 2006). "An analysis of diversity measures". en. In: *Machine Learning* 65.1, pp. 247–271. ISSN: 0885-6125, 1573-0565. DOI: 10.1007/s10994-006-9449-2. URL: http://link.springer.com/10.1007/s10994-006-9449-2 (visited on 10/26/2023).

Tang, Jie et al. (2017). "Enabling Deep Learning on IoT Devices". In: *Computer* 50.10, pp. 92–96. ISSN: 0018-9162. DOI: 10.1109/MC.2017.3641648. URL: http://ieeexplore.ieee.org/document/8057306/ (visited on 05/29/2023).

Taud, H. and J.F. Mas (2018). "Multilayer Perceptron (MLP)". en. In: *Geomatic Approaches for Modeling Land Change Scenarios*. Ed. by María Teresa Camacho Olmedo et al. Lecture Notes in Geoinformation and Cartography. Cham: Springer International Publishing, pp. 451–455. ISBN: 978-3-319-60801-3. DOI: 10.1007/978-3-319-60801-3_27. URL: https://doi.org/10.1007/978-3-319-60801-3_27 (visited on 01/22/2023).

tfmot.sparsity.keras.ConstantSparsity | TensorFlow Model Optimization (n.d.). URL: https://www.tensorflow.org/model_optimization/api_docs/python/tfmot/sparsity/keras/ConstantSparsity (visited on 06/13/2023).

Tian, Pu et al. (Oct. 2022). "WSCC: A Weight-Similarity-Based Client Clustering Approach for Non-IID Federated Learning". In: *IEEE Internet of Things Journal* 9.20. Conference Name: IEEE Internet of Things Journal, pp. 20243–20256. ISSN: 2327-4662. DOI: 10.1109/JIOT.2022.3175149.

Tsogbaatar, Enkhtur et al. (June 2021). "DeL-IoT: A deep ensemble learning approach to uncover anomalies in IoT". en. In: *Internet of Things* 14, p. 100391. ISSN: 25426605. DOI: 10.1016/j.iot.2021.100391. URL: https://linkinghub.elsevier.com/retrieve/pii/S2542660521000354 (visited on 01/18/2023).

Tsukada, Mineto, Masaaki Kondo, and Hiroki Matsutani (2020). "A Neural Network-Based On-device Learning Anomaly Detector for Edge Devices". In: *IEEE Transactions on Computers*, pp. 1–1. ISSN: 0018-9340, 1557-9956, 2326-3814. DOI: 10.1109/TC.2020.2973631. URL: https://ieeexplore.ieee.org/document/9000710/ (visited on 03/05/2023).

Tuli, Shreshth et al. (Mar. 2020). "HealthFog: An ensemble deep learning based Smart Healthcare System for Automatic Diagnosis of Heart Diseases in integrated IoT and fog computing environments". en. In: *Future Generation Computer Systems* 104, pp. 187–200. ISSN: 0167-739X. DOI: 10.1016/j.future.2019.10.043. URL: https://www.sciencedirect.com/science/article/pii/S0167739X19313391 (visited on 02/28/2023).

Tuor, Tiffany et al. (Jan. 2021). "Overcoming Noisy and Irrelevant Data in Federated Learning". In: *2020 25th International Conference on Pattern Recognition (ICPR)*. Milan, Italy: IEEE, pp. 5020–5027. ISBN: 978-1-72818-808-9. DOI: 10.1109/ICPR48806.2021.9412599. URL: https://ieeexplore.ieee.org/document/9412599/ (visited on 06/29/2022).

Upadhyay, Kartik, Ashwani Kumar Yadav, and Palak Gandhi (2019). "A Review of Internet of Things from Indian Perspective". In: *Engineering Vibration, Communication and Information Processing*. Ed. by Kanad Ray et al. Vol. 478. Series Title: Lecture Notes in Electrical Engineering. Singapore: Springer Singapore, pp. 621–632. ISBN: 9789811316418 9789811316425. DOI: 10.1007/978-981-13-1642-5_55. URL: http://link.springer.com/10.1007/978-981-13-1642-5_55 (visited on 01/03/2023).

Van Der Baan, Mirko and Christian Jutten (July 2000). "Neural networks in geophysical applications". en. In: *GEOPHYSICS* 65.4, pp. 1032–1047. ISSN: 0016-8033, 1942-2156. DOI: 10.1190/1.1444797. URL: https://library.seg.org/doi/10.1190/1.1444797 (visited on 05/29/2023).

Vanhoucke, Vincent, Andrew Senior, and Mark Z. Mao (2011). "Improving the speed of neural networks on CPUs". In: *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*.

Villanueva, Alonica et al. (July 2019). "Somnolence Detection System Utilizing Deep Neural Network". In: *2019 International Conference on Information and Communications Technology (ICOIACT)*. Yogyakarta, Indonesia: IEEE, pp. 602–607. ISBN: 978-1-72811-655-6. DOI: 10.1109/ICOIACT46704.2019.8938460. URL: https://ieeexplore.ieee.org/document/8938460/ (visited on 02/28/2023).

Vita, Fabrizio de et al. (Sept. 2020). "Quantitative Analysis of Deep Leaf: a Plant Disease Detector on the Smart Edge". In: *2020 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 49–56. DOI: 10.1109/SMARTCOMP50058.2020.00027.

Voigt, Paul and Axel von dem Bussche (2017). *The EU General Data Protection Regulation (GDPR)*. en. Cham: Springer International Publishing. ISBN: 978-3-319-57958-0 978-3-319-57959-7. DOI: 10.1007/978-3-319-57959-7. URL: http://link.springer.com/10.1007/978-3-319-57959-7 (visited on 03/17/2022).

Wan, Li et al. (June 2013). "Regularization of neural networks using dropconnect". In: JMLR.org, pp. III–1058. URL: http://dl.acm.org.ezproxy.bcu.ac.uk/citation.cfm?id=3042817.3043055 (visited on 03/22/2019).

Wan, Weishui et al. (Jan. 2009). "Enhancing the generalization ability of neural networks through controlling the hidden layers". en. In: *Applied Soft Computing* 9.1, pp. 404–414. ISSN: 15684946. DOI: 10.1016/j.asoc.2008.01.013. URL: https://linkinghub.elsevier.com/retrieve/pii/S1568494608000902 (visited on 01/02/2019).

Wang, Binghui et al. (Dec. 2020a). *GraphFL: A Federated Learning Framework for Semi-Supervised Node Classification on Graphs*. arXiv:2012.04187 [cs, stat]. URL: http://arxiv.org/abs/2012.04187 (visited on 07/04/2022).

Wang, Fangxin, Wei Gong, and Jiangchuan Liu (Apr. 2019). "On Spatial Diversity in WiFi-Based Human Activity Recognition: A Deep Learning-Based Approach". In: *IEEE Internet of Things Journal* 6.2. Conference Name: IEEE Internet of Things Journal, pp. 2035–2047. ISSN: 2327-4662. DOI: 10.1109/JIOT.2018.2871445.

Wang, Hongyi et al. (Feb. 2020b). "Federated Learning with Matched Averaging". In: *arXiv:2002.06440 [cs, stat]*. arXiv: 2002.06440. URL: http://arxiv.org/abs/2002.06440 (visited on 03/18/2022).

Wang, Jianyu et al. (2021a). "A Novel Framework for the Analysis and Design of Heterogeneous Federated Learning". In: *IEEE Transactions on Signal Processing* 69, pp. 5234–5249. ISSN: 1053-587X, 1941-0476. DOI: 10.1109/TSP.2021.3106104. URL: https://ieeexplore.ieee.org/document/9521822/ (visited on 06/29/2022).

Wang, Yansheng et al. (Aug. 2022). "Fed-LTD: Towards Cross-Platform Ride Hailing via Federated Learning to Dispatch". en. In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. Washington DC USA: ACM, pp. 4079–4089. ISBN: 978-1-4503-9385-0. DOI: 10.1145/3534678.3539047. URL: https://dl.acm.org/doi/10.1145/3534678.3539047 (visited on 12/20/2022).

Wang, Yichuan et al. (June 2021b). "Deep Learning Data Privacy Protection Based on Homomorphic Encryption in AIoT". en. In: *Mobile Information Systems* 2021. Ed. by Xiaohong Jiang, pp. 1–11. ISSN: 1875-905X, 1574-017X. DOI: 10.1155/2021/5510857. URL: https://www.hindawi.com/journals/misy/2021/5510857/ (visited on 03/02/2023).

Wang, Yifan et al. (2019). "HydraOne: An Indoor Experimental Research and Education Platform for CAVs". en. In: pp. 1–7.

Wang, Zhenyu et al. (Sept. 2020c). "Network pruning using sparse learning and genetic algorithm". en. In: *Neurocomputing* 404, pp. 247–256. ISSN: 09252312. DOI: 10.1016/j.neucom.2020.03.082. URL: https://linkinghub.elsevier.com/retrieve/pii/S0925231220304690 (visited on 10/27/2023).

Webb, Geoffrey I. (2000). "MultiBoosting: A Technique for Combining Boosting and Wagging". In: *Machine Learning* 40.2, pp. 159–196. ISSN: 08856125. DOI: 10.1023/A:1007659514849. URL: http://link.springer.com/10.1023/A:1007659514849 (visited on 11/07/2023).

Wu, Bichen et al. (Dec. 2016a). "SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving". In: *arXiv:1612.01051 [cs]*. arXiv: 1612.01051. URL: http://arxiv.org/abs/1612.01051 (visited on 01/04/2019).

Wu, Jiaxiang et al. (Dec. 2015). "Quantized Convolutional Neural Networks for Mobile Devices". In: *arXiv:1512.06473 [cs]*. arXiv: 1512.06473. URL: http://arxiv.org/abs/1512.06473 (visited on 07/02/2019).

— (June 2016b). "Quantized Convolutional Neural Networks for Mobile Devices". en. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, pp. 4820–4828. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.521. URL: http://ieeexplore.ieee.org/document/7780890/ (visited on 10/09/2018).

Wu, Qiong, Kaiwen He, and Xu Chen (2020). "Personalized Federated Learning for Intelligent IoT Applications: A Cloud-Edge Based Framework". In: *IEEE Open Journal of the Computer Society* 1, pp. 35–44. ISSN: 2644-1268. DOI: 10.1109/OJCS.2020.2993259. URL: https://ieeexplore.ieee.org/document/9090366/ (visited on 04/15/2021).

Wu, Tianze et al. (Feb. 2020). "HydraMini: An FPGA-based Affordable Research and Education Platform for Autonomous Driving". In: *2020 International Conference on Connected and Autonomous Driving (MetroCAD)*. Detroit, MI, USA: IEEE, pp. 45–52. ISBN: 978-1-72816-059-7. DOI: 10.1109/MetroCAD48866.2020.00016. URL: https://ieeexplore.ieee.org/document/9138641/ (visited on 02/28/2023).

Wu, Xiongwei, Doyen Sahoo, and Steven C. H. Hoi (July 2020). "Recent advances in deep learning for object detection". en. In: *Neurocomputing* 396, pp. 39–64. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2020.01.085. URL: https://www.sciencedirect.com/science/article/pii/S0925231220301430 (visited on 04/13/2023).

Xiao, Han, Kashif Rasul, and Roland Vollgraf (Aug. 2017). "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms". In: *arXiv:1708.07747 [cs, stat]*. arXiv: 1708.07747. URL: http://arxiv.org/abs/1708.07747 (visited on 03/24/2019).

Xie, Junyuan, Ross Girshick, and Ali Farhadi (May 2016). *Unsupervised Deep Embedding for Clustering Analysis*. arXiv:1511.06335 [cs]. URL: http://arxiv.org/abs/1511.06335 (visited on 11/28/2022).

Xie, Lingxi and Alan Yuille (Oct. 2017). "Genetic CNN". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. Venice: IEEE, pp. 1388–1397. ISBN: 978-1-5386-1032-9. DOI: 10.1109/ICCV.2017.154. URL: http://ieeexplore.ieee.org/document/8237416/ (visited on 02/25/2019).

Xiong, Zuobin et al. (Feb. 2022). "Privacy Threat and Defense for Federated Learning With Non-i.i.d. Data in AIoT". In: *IEEE Transactions on Industrial Informatics* 18.2, pp. 1310–1321. ISSN: 1551-3203, 1941-0050. DOI: 10.1109/TII.2021.3073925. URL: https://ieeexplore.ieee.org/document/9408373/ (visited on 03/02/2023).

Xu, Dongkuan and Yingjie Tian (June 2015). "A Comprehensive Survey of Clustering Algorithms". en. In: *Annals of Data Science* 2.2, pp. 165–193. ISSN: 2198-5804, 2198-5812. DOI: 10.1007/s40745-015-0040-1. URL: http://link.springer.com/10.1007/s40745-015-0040-1 (visited on 02/02/2023).

Xu, Hansong et al. (2018). "A Survey on Industrial Internet of Things: A Cyber-Physical Systems Perspective". In: *IEEE Access* 6, pp. 78238–78259. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2884906. URL: https://ieeexplore.ieee.org/document/8558534/ (visited on 02/16/2020).

Xu, Jinhua and Daniel W.C. Ho (Dec. 2006). "A new training and pruning algorithm based on node dependence and Jacobian rank deficiency". en. In: *Neurocomputing* 70.1-3, pp. 544–558. ISSN: 09252312. DOI: 10.1016/j.neucom.2005.11.005. URL: http://linkinghub.elsevier.com/retrieve/pii/S0925231206000075 (visited on 01/03/2019).

Xu, Ke et al. (Sept. 2021). "GenExp: Multi-objective pruning for deep neural network based on genetic algorithm". en. In: *Neurocomputing* 451, pp. 81–94. ISSN: 09252312. DOI: 10.1016/j.neucom.2021.04.022. URL: https://linkinghub.elsevier.com/retrieve/pii/S092523122100549X (visited on 10/27/2023).

Xu, Zirui et al. (Dec. 2019). *ELFISH: Resource-Aware Federated Learning on Heterogeneous Edge Devices*.

Yan, Peng and Guodong Long (2023). "Personalization Disentanglement for Federated Learning". In: *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*. IEEE. DOI: 10.1109/icme55011.2023.00062. URL: http://dx.doi.org/10.1109/icme55011.2023.00062.

Yan, Zhaoyi et al. (Aug. 2020). "Prune it Yourself: Automated Pruning by Multiple Level Sensitivity". In: *2020 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pp. 73–78. DOI: 10.1109/MIPR49039.2020.00022.

Yang, Qiang et al. (Mar. 2019). "Federated Machine Learning: Concept and Applications". en. In: *ACM Transactions on Intelligent Systems and Technology* 10.2, pp. 1–19. ISSN: 2157-6904, 2157-6912. DOI: 10.1145/3298981. URL: https://dl.acm.org/doi/10.1145/3298981 (visited on 03/17/2022).

Yang, Tien-Ju et al. (Sept. 2018a). "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications". In: *arXiv:1804.03230 [cs]*. arXiv: 1804.03230. URL: http://arxiv.org/abs/1804.03230 (visited on 05/04/2020).

Yang, Timothy et al. (Dec. 2018b). *Applied Federated Learning: Improving Google Keyboard Query Suggestions*. arXiv:1812.02903 [cs, stat]. URL: http://arxiv.org/abs/1812.02903 (visited on 02/23/2023).

Yang, Zhilin et al. (Mar. 2018c). *Breaking the Softmax Bottleneck: A High-Rank RNN Language Model*. arXiv:1711.03953 [cs]. DOI: 10.48550/arXiv.1711.03953. URL: http://arxiv.org/abs/1711.03953 (visited on 01/22/2023).

Yao, X. and Y. Liu (May 1997). "A new evolutionary system for evolving artificial neural networks". In: *IEEE Transactions on Neural Networks* 8.3, pp. 694–713. ISSN: 10459227. DOI: 10.1109/72.572107. URL: http://ieeexplore.ieee.org/document/572107/ (visited on 02/22/2019).

Yim, Junho et al. (July 2017). "A Gift from Knowledge Distillation: Fast Optimization, Network Minimization and Transfer Learning". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, pp. 7130–7138. ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.754. URL: http://ieeexplore.ieee.org/document/8100237/ (visited on 02/28/2019).

Yin, Xu-Cheng, Kaizhu Huang, and Hong-Wei Hao (Oct. 2015). "DE2: Dynamic ensemble of ensembles for learning nonstationary data". en. In: *Neurocomputing* 165, pp. 14–22. ISSN: 09252312. DOI: 10.1016/j.neucom.2014.06.092. URL: https://linkinghub.elsevier.com/retrieve/pii/S0925231215004270 (visited on 10/26/2023).

Yoon, Tehrim et al. (2021). "FEDMIX: APPROXIMATION OF MIXUP UNDER MEAN AUGMENTED FEDERATED LEARNING". en. In.

Yosinski, Jason et al. (2014). "How transferable are features in deep neural networks?" In: *Advances in Neural Information Processing Systems*. Vol. 27. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2014/hash/375c71349b295fbe2dcdca9206f20a06-Abstract.html (visited on 03/14/2023).

You, Shan et al. (2017). "Learning from Multiple Teacher Networks". en. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '17*. Halifax, NS, Canada: ACM Press, pp. 1285–1294. ISBN: 978-1-4503-4887-4. DOI: 10.1145/3097983.3098135. URL: http://dl.acm.org/citation.cfm?doid=3097983.3098135 (visited on 02/28/2019).

Yu, Jiyeon, Angelica de Antonio, and Elena Villalba-Mora (Feb. 2022). "Deep Learning (CNN, RNN) Applications for Smart Homes: A Systematic Review". en. In: *Computers* 11.2. Number: 2 Publisher: Multidisciplinary Digital Publishing Institute, p. 26. ISSN: 2073-431X. DOI: 10.3390/computers11020026. URL: https://www.mdpi.com/2073-431X/11/2/26 (visited on 05/22/2023).

Yu, Ruichi et al. (2018a). "NISP: Pruning Networks Using Neuron Importance Score Propagation". In: pp. 9194–9203. URL: https://openaccess.thecvf.com/content_cvpr_2018/html/Yu_NISP_Pruning_Networks_CVPR_2018_paper.html (visited on 03/10/2023).

Yu, Zhengxin et al. (Dec. 2018b). "Federated Learning Based Proactive Content Caching in Edge Computing". In: *2018 IEEE Global Communications Conference (GLOBECOM)*. Abu Dhabi, United Arab Emirates: IEEE, pp. 1–6. ISBN: 978-1-5386-4727-1. DOI: 10.1109/GLOCOM.2018.8647616. URL: https://ieeexplore.ieee.org/document/8647616/ (visited on 01/05/2023).

Yuan, Yongliang et al. (Aug. 2022a). "Alpine skiing optimization: A new bio-inspired optimization algorithm". en. In: *Advances in Engineering Software* 170, p. 103158. ISSN: 09659978. DOI: 10.1016/j.advengsoft.2022.103158. URL: https://linkinghub.elsevier.com/retrieve/pii/S0965997822000692 (visited on 02/18/2023).

Yuan, Yongliang et al. (July 2022b). "Optimization of an auto drum fashioned brake using the elite opposition-based learning and chaotic k-best gravitational search strategy based grey wolf optimizer algorithm". en. In: *Applied Soft Computing* 123, p. 108947. ISSN: 15684946. DOI: 10.1016/j.asoc.2022.108947. URL: https://linkinghub.elsevier.com/retrieve/pii/S1568494622002927 (visited on 02/18/2023).

Zeiler, Matthew D. and Rob Fergus (Nov. 2013). "Visualizing and Understanding Convolutional Networks". In: *arXiv:1311.2901 [cs]*. arXiv: 1311.2901. URL: http://arxiv.org/abs/1311.2901 (visited on 07/09/2019).

Zemouri, Ryad et al. (Dec. 2020). "A new growing pruning deep learning neural network algorithm (GP-DLNN)". en. In: *Neural Computing and Applications* 32.24, pp. 18143–18159. ISSN: 0941-0643, 1433-3058. DOI: 10.1007/s00521-019-04196-8. URL: http://link.springer.com/10.1007/s00521-019-04196-8 (visited on 10/27/2023).

Zeng, Xiaoqin and Daniel S. Yeung (Mar. 2006). "Hidden neuron pruning of multilayer perceptrons using a quantified sensitivity measure". en. In: *Neurocomputing* 69.7-9, pp. 825–837. ISSN: 09252312. DOI: 10.1016/j.neucom.2005.04.010. URL: http://linkinghub.elsevier.com/retrieve/pii/S0925231205001852 (visited on 01/03/2019).

Zhai, Shaolei et al. (2021). "Dynamic Federated Learning for GMEC With Time-Varying Wireless Link". In: *IEEE Access* 9. Conference Name: IEEE Access, pp. 10400–10412. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3050172.

Zhang, Byoung-Tak (1993). "Evolving Optimal Neural Networks Using Genetic Algorithms with Occam's Razor". en. In: p. 22.

Zhang, Heng et al. (Dec. 2021). "L1-L2 norm regularization via forward-backward splitting for fluorescence molecular tomography". en. In: *Biomedical Optics Express* 12.12, p. 7807. ISSN: 2156-7085, 2156-7085. DOI: 10.1364/BOE.435932. URL: https://opg.optica.org/abstract.cfm?URI=boe-12-12-7807 (visited on 10/27/2023).

Zhang, Huaxiang and Linlin Cao (Sept. 2014). "A spectral clustering based ensemble pruning approach". en. In: *Neurocomputing* 139, pp. 289–297. ISSN: 09252312. DOI: 10.1016/j.neucom.2014.02.030. URL: https://linkinghub.elsevier.com/retrieve/pii/S0925231214004299 (visited on 11/07/2023).

Zhang, Jing and Dacheng Tao (May 2021). "Empowering Things With Intelligence: A Survey of the Progress, Challenges, and Opportunities in Artificial Intelligence of Things". In: *IEEE Internet of Things Journal* 8.10, pp. 7789–7817. ISSN: 2327-4662, 2372-2541. DOI: 10.1109/JIOT.2020.3039359. URL: https://ieeexplore.ieee.org/document/9264235/ (visited on 01/05/2023).

Zhang, Min-Ling and Zhi-Hua Zhou (Jan. 2013). "Exploiting unlabeled data to enhance ensemble diversity". en. In: *Data Mining and Knowledge Discovery* 26.1, pp. 98–129. ISSN: 1384-5810, 1573-756X. DOI: 10.1007/s10618-011-0243-9. URL: http://link.springer.com/10.1007/s10618-011-0243-9 (visited on 10/26/2023).

Zhang, Tianyun et al. (Sept. 2018a). "A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers". In: *Proceedings of the European Conference on Computer Vision (ECCV)*.

Zhang, Xiangyu et al. (June 2018b). "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Salt Lake City, UT: IEEE, pp. 6848–6856. ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00716. URL: https://ieeexplore.ieee.org/document/8578814/ (visited on 07/08/2019).

Zhang, Ying et al. (June 2018c). "Deep Mutual Learning". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Salt Lake City, UT: IEEE, pp. 4320–4328. ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00454. URL: https://ieeexplore.ieee.org/document/8578552/ (visited on 03/06/2019).

Zhang, Yong et al. (2014). "A Weighted Voting Classifier Based on Differential Evolution". en. In: *Abstract and Applied Analysis* 2014, pp. 1–6. ISSN: 1085-3375, 1687-0409. DOI: 10.1155/2014/376950. URL: http://www.hindawi.com/journals/aaa/2014/376950/ (visited on 10/26/2023).

Zhang, Zhaozhao and Junfei Qiao (Aug. 2010). "A node pruning algorithm for feedforward neural network based on neural complexity". In: *2010 International Conference on Intelligent Control and Information Processing*. Dalian, China: IEEE, pp. 406–410. ISBN: 978-1-4244-7047-1. DOI: 10.1109/ICICIP.2010.5564272. URL: http://ieeexplore.ieee.org/document/5564272/ (visited on 01/03/2019).

Zhang, Zixuan et al. (Jan. 2020). "Smart Triboelectric Socks for Enabling Artificial Intelligence of Things (AIoT) Based Smart Home and Healthcare". In: *2020 IEEE 33rd International Conference on Micro Electro Mechanical Systems (MEMS)*. ISSN: 2160-1968, pp. 80–83. DOI: 10.1109/MEMS46641.2020.9056149.

Zhao, Shubao et al. (2023). "Two-Phased Federated Learning with Clustering and Personalization for Natural Gas Load Forecasting". In: Springer International Publishing. DOI: 10.1007/978-3-031-28996-5_10. URL: http://dx.doi.org/10.1007/978-3-031-28996-5_10.

Zhao, Weiguo, Liying Wang, and Seyedali Mirjalili (Jan. 2022). "Artificial hummingbird algorithm: A new bio-inspired optimizer with its engineering applications". en. In: *Computer Methods in Applied Mechanics and Engineering* 388, p. 114194. ISSN: 00457825. DOI: 10.1016/j.cma.2021.114194. URL: https://linkinghub.elsevier.com/retrieve/pii/S0045782521005259 (visited on 02/18/2023).

Zhao, Yanqi et al. (Oct. 2019). "Blockchain based privacy-preserving software updates with proof-of-delivery for Internet of Things". en. In: *Journal of Parallel and Distributed Computing* 132, pp. 141–149. ISSN: 07437315. DOI: 10.1016/j.jpdc.2019.06.001. URL: https://linkinghub.elsevier.com/retrieve/pii/S074373151930098X (visited on 03/02/2023).

Zhao, Yue et al. (June 2018). *Federated Learning with Non-IID Data*. Tech. rep. arXiv:1806.00582. arXiv:1806.00582 [cs, stat] type: article. arXiv. URL: http://arxiv.org/abs/1806.00582 (visited on 06/29/2022).

Zheng, Longfei et al. (May 2021). "ASFGNN: Automated separated-federated graph neural network". en. In: *Peer-to-Peer Networking and Applications* 14.3, pp. 1692–1704. ISSN: 1936-6442, 1936-6450. DOI: 10.1007/s12083-021-01074-w. URL: https://link.springer.com/10.1007/s12083-021-01074-w (visited on 07/05/2022).

Zhong, Chang-le, Zhen Zhu, and Ren-Gen Huang (Oct. 2017). "Study on the IOT Architecture and Access Technology". In: *2017 16th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES)*. Anyang: IEEE, pp. 113–116. ISBN: 978-1-5386-2162-2. DOI: 10.1109/DCABES.2017.32. URL: http://ieeexplore.ieee.org/document/8253048/ (visited on 01/10/2023).

Zhou, Bin et al. (Aug. 2016). "Smart home energy management systems: Concept, configurations, and scheduling strategies". en. In: *Renewable and Sustainable Energy Reviews* 61, pp. 30–40. ISSN: 1364-0321. DOI: 10.1016/j.rser.2016.03.047. URL: https://www.sciencedirect.com/science/article/pii/S1364032116002823 (visited on 02/28/2023).

Zhou, Zhi et al. (Aug. 2019). "Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing". In: *Proceedings of the IEEE* 107.8, pp. 1738–1762. ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2019.2918951. URL: https://ieeexplore.ieee.org/document/8736011/ (visited on 01/05/2023).

Zhou, Zhi-Hua (June 2012). *Ensemble Methods: Foundations and Algorithms*. en. 1st ed. Chapman and Hall/CRC. ISBN: 978-1-4398-3005-5. DOI: 10.1201/b12207. URL: https://www.taylorfrancis.com/books/9781439830055 (visited on 12/30/2019).

— (2021). "Ensemble Learning". en. In: *Machine Learning*. Singapore: Springer Singapore, pp. 181–210. ISBN: 9789811519666 9789811519673. DOI: 10.1007/978-981-15-1967-3_8. URL: https://link.springer.com/10.1007/978-981-15-1967-3_8 (visited on 01/16/2023).

Zhu, Fu et al. (June 2022). "An Improved MobileNet Network with Wavelet Energy and Global Average Pooling for Rotating Machinery Fault Diagnosis". In: *Sensors (Basel, Switzerland)* 22.12, p. 4427. ISSN: 1424-8220. DOI: 10.3390/s22124427. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9228785/ (visited on 04/13/2023).

Zhu, Hangyu et al. (Nov. 2021). "Federated learning on non-IID data: A survey". en. In: *Neurocomputing* 465, pp. 371–390. ISSN: 09252312. DOI: 10.1016/j.neucom.2021.07.098. URL: https://linkinghub.elsevier.com/retrieve/pii/S0925231221013254 (visited on 03/15/2022).

Zhu, Michael and Suyog Gupta (Oct. 2017). "To prune, or not to prune: exploring the efficacy of pruning for model compression". en. In: *arXiv:1710.01878 [cs, stat]*. arXiv: 1710.01878. URL: http://arxiv.org/abs/1710.01878 (visited on 11/06/2018).

Zou, Han et al. (Jan. 2018). "WinLight: A WiFi-based occupancy-driven lighting control system for smart building". en. In: *Energy and Buildings* 158, pp. 924–938. ISSN: 0378-7788. DOI: 10.1016/j.enbuild.2017.09.001. URL: https://www.sciencedirect.com/science/article/pii/S0378778817313907 (visited on 02/28/2023).