EFFICIENT NEURAL ARCHITECTURES FOR EDGE DEVICES

Dolly Sapra

EFFICIENT NEURAL ARCHITECTURES FOR EDGE DEVICES

DOLLY SAPRA

This work was financially supported by EU Horizon 2020 project Research and Innovation programme under grant agreement No. 780788 for the project ALOHA.

Cover Design: from original art by Dolly Sapra.
Thesis template: `classicthesis` by André Miede and Ivo Pletikosić.
Printed and bound by Ipskamp printing
ISBN: 978-94-6421-743-8

EFFICIENT NEURAL ARCHITECTURES FOR EDGE DEVICES

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. K.I.J. Maex
ten overstaan van een door het College voor Promoties ingestelde commissie,
in het openbaar te verdedigen in de Agnietenkapel
op woensdag 15 juni 2022, te 16.00 uur

door Dolly Sapra
geboren te Delhi

Promotiecommissie

Promotores:	prof. dr. A.D. Pimentel	Universiteit van Amsterdam
	prof. dr. ir. C.T.A.M. de Laat	Universiteit van Amsterdam
Overige leden:	prof. dr. ir. C.I. Sánchez Gutiérrez	Universiteit van Amsterdam
	prof. dr. J.A. Good	Universiteit van Amsterdam
	dr. P. Grosso	Universiteit van Amsterdam
	prof. dr. D. Marculescu	The University of Texas at Austin
	prof. dr. P. Lago	Vrije Universiteit van Amsterdam
	prof. dr. M. Verhelst	KU Leuven

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

To my lovely daughter Amaira.
Thank you for understanding that mummy was sometimes busy with her work.

CONTENTS

ACRONYMS

AI	Artificial Intelligence
ATME	Accuracy, Throughput, Memory and Energy : characteristics of a neural network when executed on a target hardware
CNN	Convolutional Neural Network
CPU	Central Processing Unit
EPT	Evolutionary Piecemeal Training
FC	Fully Connected
GA	Genetic Algorithm
GPU	Graphical Processing Unit
GSOF	Gradually Saturating Objective Function
GSOP	Gradually Saturating Objective Problem
HAR	Human Activity Recognition
IMU	Inertial Measurement Unit
IoT	Internet of Things
KCN	Knowledge Centric Networking
MAC	Multiply Accumulate (arithmatic) operation
M2M	Machine to Machine
NAS	Neural Architecture Search
ONNX	Open Neural Network eXchange framework
PR-AUC	Area under curve for precision recall graph
ReLU	Rectified Linear Unit

RGB	Red Green Blue (channel of a color-image)
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SBRS	Scenario Based Run-time Switching
MoC	(SBRS) Model of Computation
TP	(SBRS) Transition Protocol
UAV	Unmanned Aerial Vehicle

1

INTRODUCTION

In recent times, the brain-inspired neural networks are at the forefront of Artificial Intelligence (AI) based research and development. Neural networks often demonstrate excellent proficiency in performing complex and challenging problems such as image classification, speech recognition and natural language processing [1, 2]. When a neural network contains many layers, it is known as a deep neural network or a deep learning algorithm. They are able to learn patterns and correlations from available data to predict behavior of the unseen information, and are extremely useful in situations where the relationship between input and output is complex, non-linear and dynamic in nature.

Fast growing research and engineering endeavors are being pursued by both academia and industry, to realize various deep learning based solutions. As reported in *AI index report 2021* [3], global corporate investment in AI in 2020 was ≈67.9 billion USD, which is more than five times higher than in 2015, clearly indicating the importance of AI based solutions in various industries. Concurrently, the number of research publications in the area of deep learning have grown exponentially in the last decade, as illustrated in Figure 1.1 for Arxiv publications, stipulating the expanding research initiatives in the domain of neural networks.

In the last few years, several different flavors of deep learning models have been investigated for a wide ranging set of applications, for example, Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNN). CNNs, such as Alexnet [4] and Resnet [5], are a special case of deep learning algorithms, where matrix multiplications include convolutional filter operations designed for image and video analysis. There are other types of deep neural networks designed for a variety of tasks; an overview of different types of deep learning models can be found in [6]. This thesis focuses on Convolutional Neural Networks (CNNs), and more details on CNNs are provided in Chapter 2.

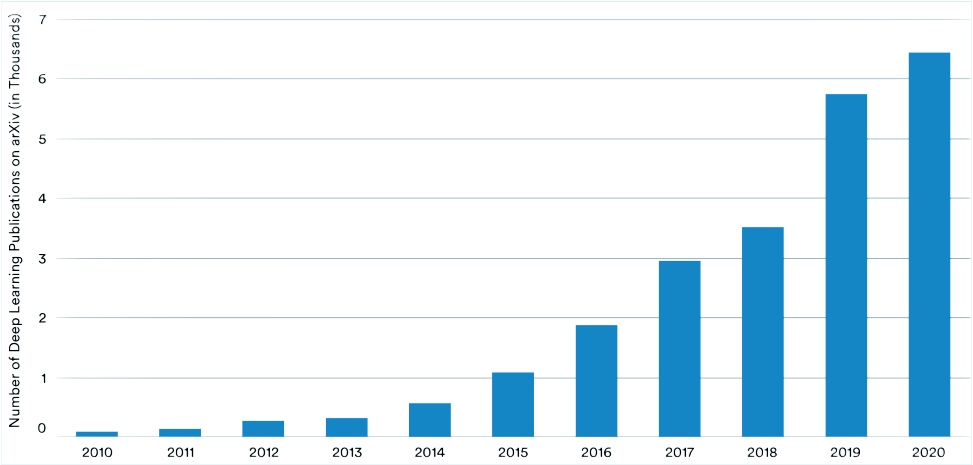


Figure 1.1: Number of deep learning publications on Arxiv, 2010-20 [3]

Furthermore, there are many design choices to construct a CNN, sometimes called *hyperparameters*. In addition to the structure related parameters, the term *hyperparameters* is also used to capture other details pertaining to the training of the CNN from available data. In this thesis, to avoid confusion, we use the terms *neural architecture*, *model architecture* or simply *architecture* to refer to the structure of the neural network.

A distinct advantage neural networks have over some other models is their ability to consume and analyse large amounts of data, with little to no pre-processing required. This thesis only looks into supervised learning [7], where the deep learning models learn from available data and the only pre-requisite is the presence of output labels for every intended input.

A neural network trains on data in the format of input-output pairs, and stores the information (about the data) in the form of weights or coefficients of mathematical operations. The sophisticated training process is performed via a back-propagation algorithm [8], briefly described in Chapter 2. A trained neural network can be deployed to predict the output labels on previously unseen data. This prediction process is also referred to as *inference*.

The model architecture and coefficients together form the core and essence of any neural network. Essentially, a neural network is a numerical model, based on several matrix multiplications, which are connected in a pattern specified through its architecture. The numerical model in turn implies that some computations can be done in parallel (or distributed) manner, depending on available hardware resources and scheduling techniques.

To summarize, typically a neural network can be characterized by:

- the ability to learn a complex and non-linear function, given sufficient data;
- encapsulation of information from input data into the model coefficients, through training;
- little dependence on data pre-processing;
- computationally intensive;
- high degree of the task- and data-level parallelism.

1.1 DEEP LEARNING AT THE EDGE

In the age of the Internet of Things (IoT), where we have a complex network of connected devices, sensors and computing units, there is a large amount of data churning up every minute. As predicted by [9], the number of devices connected to IP networks will be more than three times the global population by 2022. Simultaneously, as these IoT devices may generate a lot of data, global Machine to Machine (M2M) IP traffic will grow more than seven-fold in 5 years, from 3.7EB per month in 2017 to more than 25EB by 2022. These enormous amounts of data generated cannot be filtered and analyzed by humans, therefore, the need for artificial intelligence models such as neural networks to be utilized to convert raw data, from these devices, into meaningful knowledge. This conversion is majorly performed on a central server or knowledge creator node with high computation capabilities. However, it is becoming highly desirable to have data processing closer to the source.

Usually an embedded system, which is a small micro-processor based computer hardware with limited capabilities, is deployed closer to the data sensors or consumers, to perform a dedicated task. It is also referred to as an *edge device*, which may act independently or as a special part of a larger, typically cloud-based, system. In this thesis, an edge, an edge device and an embedded system are interchangeably used, unless stated specifically.

Autonomous data processing at edge devices may increase privacy and security of the system in general, since there is less data to send to the cloud/rest of the IoT network. This also leads to faster response times and low dependence on the internet for decision making. The system may have high availability with increased reliability, since power outages and other

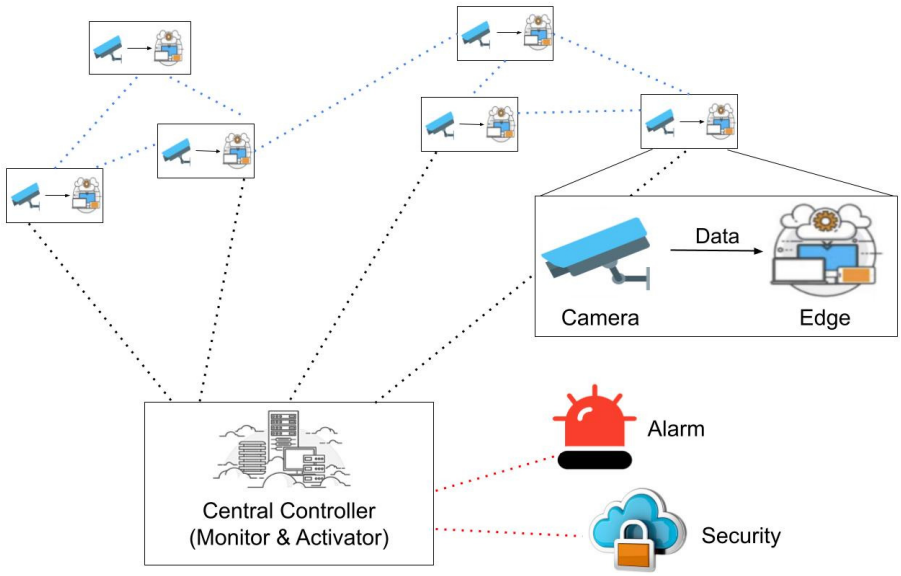


Figure 1.2: Video surveillance : an IoT network

network disruptions have less impact on the operation. Moreover, there can be a significant environmental impact along with cost savings, since less energy is spent on transporting, managing and securing data.

Figure 1.2 illustrates an example of an IoT network, a video surveillance system, with connected cameras and intrusion detection system. A camera is a data generator with some computing resources i.e., an edge device, to perform simple processing. It is advantageous to have a CNN running on each camera/edge node to provide an intelligent surveillance application. All nodes can be connected to a powerful (central) server to perform compute intensive tasks, which in turn may also operate as an activator for reactions, based on surveillance decisions.

As much as the deployment of neural networks on the edge is desirable, it is not without its challenges. Oftentimes, the edge devices are cheap, able to fit in small spaces, and run on an internal battery. Generally, edge devices have limited resources available locally to them. If they have a GPU, it has fewer capabilities than a typical high-end server GPU, which is required for a fast CNN execution. Additionally, the memory can be limited on the edge owing to small surface area and if they operate on batteries, they may also have strict energy consumption restrictions.

Since training of a neural network, using a typical back-propagation algorithm (as will be briefly described in Chapter 2), is an extremely resource

intensive algorithm, it is generally accepted that the training is best done on a high-end hardware or a server machine in the cloud. Some re-training or small modifications to the CNN are possible on the edge, though these operations are limited due to the underlying resource constraints. Therefore, an edge device is often utilized to perform inference with an already trained neural network and only in a few cases, may perform small updates to the deep learning model. Even so, it is noteworthy that not all neural networks are suitable to be deployed at the edge for inference either, owing to the resource constraints.

1.2 NEURAL ARCHITECTURE SEARCH

In a traditional system design process, the neural architecture is manually designed, adhering to the resource constraints of the target hardware to a feasible extent. However manual design is time consuming, and heavily dependent on human expertise in both the task domain and neural architecture optimization. Moreover, the awareness of target hardware's characteristics and constraints at design time can be limited, leading to unnecessary iterations between architecture design and its inefficient mapping to the designated hardware.

The research in automation of neural network design, called Neural Architecture Search (NAS), has led to many novel methodologies being proposed over the last few years. These methodologies diligently search for an efficient neural network architecture for the specific task, while demanding low human expertise and interference. The NAS algorithms that have been proposed over the last few years, cover a wide range of domains and search techniques. They vary in their definition of the search space, search strategy, or performance estimation technique. Popular NAS methodologies use evolutionary algorithms, Reinforcement Learning (RL) and one-shot search techniques. We will discuss these more in detail in Chapter 3 and Chapter 5. Most of these techniques consider the search as an optimization problem, where the objective is mainly to achieve a high accuracy.

The attentiveness towards *only* high accuracy is reasonable when resource consumption by a neural network is not a limiting factor for their smooth operation. This thesis, as evident from its title, is focused on deep learning for embedded systems, specifically on efficient neural architectures for resource constrained edge devices. Thus, the first question this thesis attempts to answer is centered on searching for neural architectures that are suitable to be deployed on an edge device.

For edge devices, it is critical that additional non-functional objectives are satisfied during the search for an efficient neural network. This is to ensure that a deep learning model can smoothly operate within the available resources. These additional objectives can be generic such as low arithmetic operation count or they can be specific for the target hardware such as low energy cost, low memory usage, low latency etc. Therefore, the NAS methodologies for an embedded system imperatively need to include multiple objectives as their search criteria.

In a multi-objective scenario, both RL and One-Shot NAS methodologies are limited in their flexibility to incorporate additional objectives. The functioning of a RL algorithm is based on a reward function, which directs the architecture search by rewarding good architectures. In multi-objective RL approaches, an extension to the reward function needs to be suitably designed, to include all the trade-offs between various objectives. For one-shot models, it is also not obvious how they can be extended for multi-objective optimization since it is based on a search for sub-networks, which are of roughly the same size. There are some works proposed [10, 11] to overcome these issues, although, it is still not clear how more objectives can be added to search in this indirect manner. Evolutionary algorithms are more flexible in this regard, with years of research on multi-objective optimization, and the ease to extend the algorithm to include more objectives.

However, the evolutionary algorithms need numerous GPUs to prepare, run and converge the search process, which can consume tens to thousands of days. This is because many of the existing evolutionary NAS approaches rely heavily on resource-intensive and time-consuming training algorithms to evaluate the accuracy of a neural network, which is required to direct the search. Usually, full training of every CNN architecture, is considered to be an isolated and separate task in these algorithms. This problem leads us to our first research question:

RQ1: *How can we design an efficient NAS algorithm that reduces the search time, and has the capability to optimize for multiple objectives?*

To be able to retain the flexibility of evolutionary algorithms for multi-objective NAS and yet converge within reasonable time, we examined these algorithms from a different viewpoint. In our work, we look into the possibility of searching for efficient architectures during a modified and ex-

tended training process, in contrast to the conventional training methodology as a distinct function. In Chapter 3, we propose a novel evolutionary based NAS algorithm, called Evolutionary Piecemeal Training (EPT), which searches for an efficient neural architecture for a given task and converges in a few GPU hours.

Our work leverages a population-based computing technique which allows a group of Convolutional Neural Networks (CNNs), a subset of neural networks with mainly convolutional layers, to train in parallel. During this training, evolutionary operators are applied to some random neural networks at regular intervals which leads to architecture modification and hence exploration of the search space. A new architecture derived like this is always already partially trained as it was modified from another architecture undergoing training. In subsequent iterations, the derived architectures continue to train. Towards the end of this algorithm, the best candidates are selected from the population, which can then be post processed or trained further, as needed.

The algorithm is discussed in more detail in Chapter 4, which states the training and subsequent accuracy evaluation as a "Gradually Saturating Objective Function" and explains the modifications carried out to the standard algorithm, in order to allow for a better and more efficient search methodology.

Both Chapter 3 and Chapter 4 only consider accuracy as an objective, though all the neural networks generated and modified during search are bound by minimum and maximum values for each architecture parameter. These constraints are in place to make sure that architectures do not become too large and ensures to limit the resource consumption of the final neural network on the target hardware. This is one of the most important factors to consider for tasks intended to be deployed on embedded systems.

The methodology is further extended in Chapter 5 to include multiple objectives in the EPT methodology. To prove the efficacy of the extension to original methodology, a first consideration is the reduction of the number of parameters of the neural network as an additional search goal. The accuracy maximization and parameter minimization can be conflicting objectives for an efficient neural network. Smaller CNNs tend to have lower accuracy and high accuracy is generally obtained by larger neural networks. However, too many parameters tend to cause over-fitting, which may lead to poor generalization, and therefore, highlight the importance of a constrained search process not only for hardware resource usage, but also to avoid over-fitting.

With multiple objective based searches, selection of the best candidates is concluded through pareto optimization, where any objective cannot be improved without worsening some of the other objectives. The set of candidates selected in such a fashion are collectively called a Pareto Front. The Pareto Front obtained upon convergence sets forth the various possible CNNs to deploy on the target edge device. It allows the designer to be aware of the architecture choices available in terms of which CNNs provide a trade-off between the size of CNN versus the accuracy. One of these CNNs can be strategically deployed depending upon the desired functional goals and available resources on the device.

To further augment and enhance our methodology for platform aware characteristics, we looked at the most common demands on a CNN-based application deployed on an edge, which are:

1. **High accuracy:** The CNN should be able to properly perform a task, for which it is designed;
2. **High throughput:** Typically, the applications on the edge, require CNNs to provide real-time response;
3. **Low memory cost:** Most edge devices have a limited amount of memory available;
4. **Low energy cost:** The energy of battery-powered edge devices, like e.g. drones, is strictly limited.

The CNN execution characteristics - Accuracy, Throughput, Memory cost, and Energy cost are hereinafter referred to as ATME characteristics. The accuracy, typically measured in percent, characterizes the fraction of correct predictions generated by a CNN from the total number of predictions generated by the CNN. The throughput, typically measured in frames per second (fps), characterizes the speed with which the CNN is able to process input data and produce output data. The memory cost, typically measured in Megabytes (MB), specifies the total amount of memory required to execute a CNN. The energy cost, measured in Joules, specifies the amount of energy consumed by a CNN to process one input frame.

1.3 ADAPTIVITY IN CNN-BASED APPLICATIONS FOR EDGE

In real-world applications, the priorities of a CNN-based application, in terms of its design objectives, are often influenced by the application environment, and can change during the application run-time. The question

then arises if the neural architectures searched by a NAS methodology are still efficient, when faced with a dynamic environment. Hence, the second part of the thesis focuses on incorporating adaptivity (or some aspects of it) into a CNN-based application, which is deployed on an edge device.

Considering the example shown earlier for a video surveillance application (Figure 1.2), the cameras can be mounted on an unmanned aerial drone to monitor the traffic conditions. The CNN-based application running on the drone can have different priorities, dependent on the situation on the roads and the level of the device's battery. When the traffic is heavy, the application priority would be to have high throughput and high accuracy to quickly process its input data, which normally would result in high energy cost. On the other hand, during a traffic jam, when the high throughput is not essential, or in the event when the battery of the drone is running low, the application would perform better by prioritizing energy efficiency over high throughput.

The characteristics of a system that do not directly pertain to its core functionality, but are important to ensure that the system operates smoothly, are called extra-functional characteristics of the system. The example above shows that CNN-based applications need a mechanism that can adapt their extra-functional characteristics to the changes in the environment during run-time. Additionally, such a mechanism should provide a high level of responsiveness, e.g., if a drone battery is running low, the CNN-based application on the drone, should switch to an energy-efficient mode as soon as possible. This leads to our second research question:

RQ2: *How can we ensure that a CNN-based application is able to efficiently adapt its extra-functional characteristics synchronously with the changes in its environment at run-time?*

To answer this question, in Chapter 6, we propose a novel Scenario-Based Run-time Switching (SBRS) methodology for CNN-based applications executed at the edge. A CNN is associated with each of the application's scenarios, which is specifically designed to conform to certain application's needs in terms of the ATME characteristics. During the application execution, the environment can trigger the application to switch between the scenarios, thereby adapting the characteristics of the application to the environmental variations.

The scenarios are derived prior to deployment, from the Pareto Front obtained by executing the multi-objective EPT methodology with ATME characteristics as its four objectives. Thus, the adaptivity in SBRS methodology is contrived on the foundations of multi-objective NAS, while taking advantage of unique characteristics of the diverse neural architectures on the final Pareto Front.

It is important to note that the versatility required from an CNN-based application on the edge goes beyond immediate environmental changes. Over the course of an application's active lifetime, it is aspired that the application continues to be resilient and adaptable to changes. Considering the fact that CNNs are not only a means to add intelligence to a device, they are also an important knowledge modality for the endless data being produced. Moreover, we already see the dynamic behaviour of a neural network during the operation of the EPT algorithm. Neural architectures are slightly modified during each iteration to explore a huge search space. Even though the motivation behind continuous neural architecture modification is different than adaptivity, it is easy to notice that neural networks can be seen as dynamic entities. This idea motivates the last research chapter of our thesis, where we examine adaptivity of neural network over a longer period of time, as opposed to just one operation (in the previous research question). This awareness steers us towards the third research question:

RQ3: *Is it possible for neural networks to be treated as a dynamic entity during its active lifetime? If so, how can we ensure that a CNN, deployed at the edge, can be regularly updated and maintained?*

To answer this question, we look towards the *Knowledge Centric Networking* (KCN) paradigm, where the communication in an IoT network changes from a data-centric communication to knowledge-centric communication. In a knowledge-centric communication, different AI models form the basis of a communication between different devices, as opposed to the raw-data (or semi-processed data).

Revisiting the example of the video surveillance system presented in Figure 1.2, which in essence is a distributed intelligent network, with each camera having its own deep learning model. Depending on when the camera is added to the system, it is possible that they have slightly different models deployed on them. Nevertheless, it is still expected that they are

highly correlated owing to a common task they all perform. Assuming that this system is a KCN based IoT network, this high correlation of CNNs deployed on different devices can be exploited by the distributed system to reuse and combine each other's knowledge to create a better application. This can be achieved by coordinating efficient communication, exchange and update of the various neural networks deployed, while being adaptable to accommodate new data generated and insights learnt.

Standardizing the knowledge update process, in the context of KCN motivates the proposal of a framework in Chapter 7. The proposed framework focuses on CNNs as dynamic models and various approaches to update them. It is noteworthy that the techniques discussed in Chapter 7 are not limited to a KCN based IoT network. They can be applied in any situation that warrants an adaptive deep learning based application.

In brief, the framework facilitates creation of a new CNNs and modification of existing ones, and allows combination of multiple CNNs to compose a new CNN. In addition, it is able to isolate layers of the CNN, which can be individually transferred while supporting packaging and compression for distribution. Some of these techniques are motivated from the NAS work in the previously proposed EPT, where architecture modifications are performed through genetic operators to carry out the exploration. There are multitude of ways in which deep learning models can be modified, both weights and architecture of the neural network can be updated, e.g. add/prune layers, add residual connections, change layer activation. Importantly, all of these tasks are unrelated to the training of CNNs or knowledge creation directly, even so, these tasks are needed to keep the network's knowledge contemporary and maintain communication brevity with frequent updates.

1.4 THESIS OVERVIEW

Figure 1.3 visualises how this thesis is organised. Chapter 2 provides background information for topics discussed throughout the rest of the thesis. This chapter explores various neural architectures in the context of CNNs and their evaluation for extra-functional characteristics.

Following the background chapter, this thesis is organized into two major parts. The first part, consisting of three chapters, is focused on Neural Architecture Search. Chapter 3 introduces a novel NAS algorithm called Evolutionary Piecemeal Training (EPT), which is based on a genetic algorithm. EPT defines a search space of valid CNNs and trains a population of

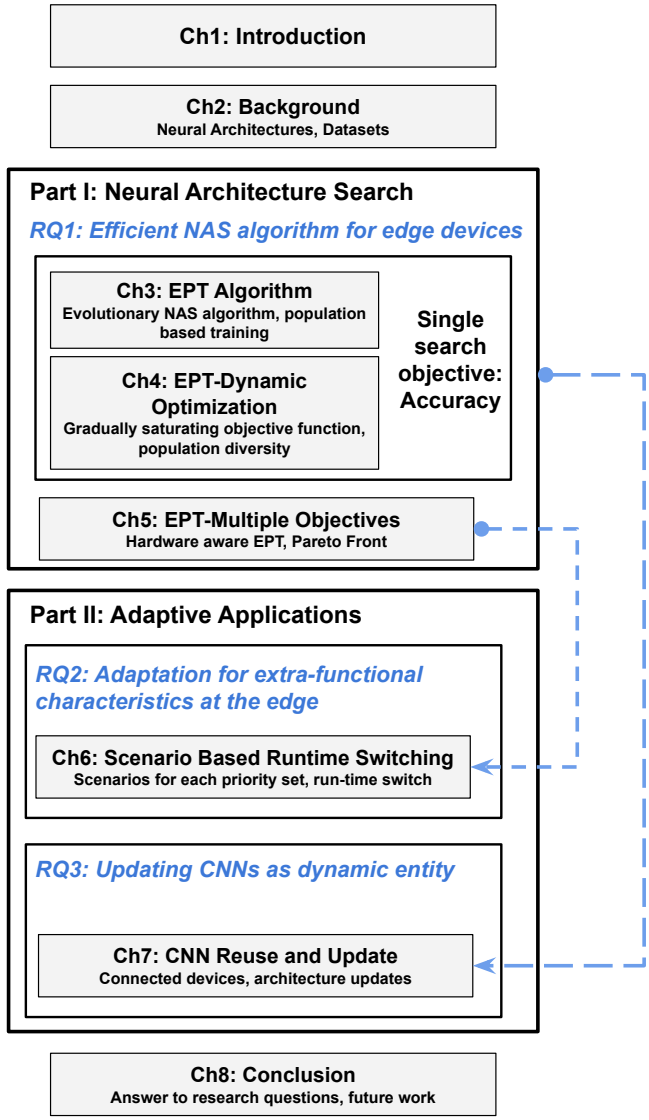


Figure 1.3: Thesis organisation, including the chapters, keywords and research questions

them in parallel while searching for an efficient neural architecture through evolutionary operators. Chapter 4 dives deeper into the workings of the EPT algorithm. It explains the nature of dynamic optimization required and a population diversity based mechanism deployed to derive a suitable and efficient neural architecture.

So far, both these chapters consider only accuracy as the objective for the search. Chapter 5 extends the original EPT algorithm to include multiple objectives to provide a Pareto Front of suitable neural architectures. The flexibility of the EPT algorithm is demonstrated through two sets of separate experiments, each with a different number of multiple objectives.

The second part of the thesis contains two chapters and is focused on adaptivity in CNN based applications deployed on edge devices. The work presented in both of these chapters is motivated from the first part of the thesis. Chapter 6 presents a mechanism, called Scenario Based Run-time Switching, for a CNN-based application to allow dynamic adaptation towards its extra-functional behaviour, with respect to the changes in the application environment at run-time. This chapter depends heavily on the hardware-aware EPT algorithm presented in Chapter 5, to derive the scenarios suitable for different operational modes.

While Chapter 6 focuses on adaptivity during operation time, Chapter 7 examines adaptivity during a life cycle of a CNN-based application by considering neural networks as a dynamic entity. In Chapter 7, we discuss various possible scenarios where the application needs to be fine-tuned, when the application is already deployed at the edge and suggests approaches to update the CNNs as required. The concept of treating a CNN as a dynamic entity originates from the EPT algorithm, where neural architectures are constantly being modified in each iteration. Although the reason for the modification is to traverse a large search space of neural architectures, the rationale of treating a neural network as a dynamic model is inherently present in the algorithm. The work presented in Chapter 7 expands on the ideas taken from the first part of the thesis and other existing literature to formulate them in the context of Knowledge Centric Networking for adaptive environments.

These five research chapters contain our core contributions, and the thesis culminates with a concluding chapter (Chapter 8), where we reflect on the research questions and present some ideas on future work.

1.4.1 *Origins*

Listed below are the author's contributions and papers on which each of the research chapters is based. The next section enumerates the full list of the author's publications.

PART I

ch.3: *"Constrained evolutionary piecemeal training to design convolutional neural networks"* [P1]

"Designing convolutional neural networks with constrained evolutionary piecemeal training" [P2]

ch.4 *"An evolutionary optimization algorithm for gradually saturating objective functions"* [P3]

ch.5 *"Designing convolutional neural networks with constrained evolutionary piecemeal training"* [P2]

"Scenario Based Run-time Switching for Adaptive CNN-based Applications at the Edge" [P4]

PART II

ch.6: *"Scenario Based Run-time Switching for Adaptive CNN-based Applications at the Edge"* [P4]

ch.7: *"Deep learning model reuse and composition in knowledge centric networking"* [P5]

For papers [P1–P3, P5], the author of this thesis is the principal author and performed all of the data analysis, software development, experimental set-up, validation and writing of the original draft. In the paper [P4], the author of this thesis was the main machinist in scenario definition and derivation for the target hardware. The author performed data analysis, software development, experiments and validation for scenario derivation and is responsible for the writing of related parts in the original draft.

Papers [P6–P8] were part of the papers published though the project *ALOHA*, which funded the author's PhD. The content of these papers is part of the thesis in some manner. Some parts from the paper [P9] have been used in Chapter 2 to provide some background for neural architectures. The rest of the papers are not directly related to the thesis, but were published during the PhD.

1.5 AUTHOR PUBLICATIONS

[P1] Dolly Sapra and Andy D Pimentel. "Constrained evolutionary piecemeal training to design convolutional neural networks." In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. (Best Paper Award). 2020.

[P2] Dolly Sapra and Andy D Pimentel. "Designing convolutional neural networks with constrained evolutionary piecemeal training." In: *Applied Intelligence* (2021).

[P3] Dolly Sapra and Andy D Pimentel. "An evolutionary optimization algorithm for gradually saturating objective functions." In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2020.

[P4] Svetlana Minakova, Dolly Sapra, Todor Stefanov, and Andy D Pimentel. "Scenario Based Run-time Switching for Adaptive CNN-based Applications at the Edge." In: *ACM Transactions on Embedded Computing Systems (TECS)* (2021).

[P5] Dolly Sapra and Andy D Pimentel. "Deep Learning Model Reuse and Composition in Knowledge Centric Networking." In: *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. 2020.

[P6] Paolo Meloni, Daniela Loi, Paola Busia, Gianfranco Deriu, Andy D Pimentel, Dolly Sapra, Todor Stefanov, Svetlana Minakova, Francesco Conti, et al. "Optimization and deployment of CNNs at the edge: the ALOHA experience." In: *Proceedings of the 16th ACM International Conference on Computing Frontiers*. 2019.

[P7] Paolo Meloni, Daniela Loi, Gianfranco Deriu, Andy D Pimentel, Dolly Sapra, Bernhard Moser, Natalia Shepeleva, Francesco Conti, Luca Benini, et al. "ALOHA: an architectural-aware framework for deep learning at the edge." In: *Proceedings of the Workshop on INTelligent Embedded Systems Architectures and Applications*. 2018.

[P8] Paolo Meloni, Daniela Loi, Gianfranco Deriu, Andy D Pimentel, Dolly Sapra, Maura Pintort, Battista Biggio, Oscar Ripolles, David Solans, et al. "Architecture-aware design and implementation of CNN algorithms for embedded inference: the ALOHA project." In: *2018 30th International Conference on Microelectronics (ICM)*. 2018.

[P9] Ilja van Ipenburg, Dolly Sapra, and Andy D Pimentel. "Exploring Cell-based Neural Architectures for Embedded Systems." In: *2nd International Workshop on IoT, Edge, and Mobile for Embedded Machine Learning*. 2021.

[P10] Uraz Odyurt, Dolly Sapra, and Andy D Pimentel. "The Choice of AI Matters: Alternative Machine Learning Approaches for CPS Anomalies." In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. 2021.

[P11] Dolly Sapra and Sebastian Altmeyer. "Work-in-progress: design-space exploration of multi-core processors for safety-critical real-time systems." In: *2017 IEEE Real-Time Systems Symposium (RTSS)*. 2017.

1.6 SOURCE CODE

- The EPT implementation and SBRS scenario derivation script is available at `https://github.com/dollysapra/EPT`
- The framework implementation as described in Chapter 7 is available at `https://github.com/dollysapra/ONNXAlter`

2

BACKGROUND

In this chapter, we discuss the core ideas and terminologies which form the foundation of this thesis. We delve into deep learning and challenges faced for its deployment at the edge. Additionally, details are provided on the AI tasks and datasets that have been used for experiments in the later chapters.

2.1 CONVOLUTIONAL NEURAL NETWORK

Convolutional Neural Networks (CNNs) are the most popular type of neural network [12], and are mainly utilized for vision based tasks due to their ability to recognize patterns in images. A CNN consists of many hidden layers between input and output through which the data must transit. The majority of these hidden layers in a CNN are convolutional layers. Figure 2.1 illustrates the working of a convolutional operation with a small mathematical matrix, called kernel. A kernel performs a matrix multiplication with the input data to construct the output. The kernel is typically much smaller than the input, so it repeatedly performs the matrix multiplication operation until the whole input is traversed.

The number of steps moved per operation by the kernel is called its stride. The input can also be padded with zeroes or repeating edge values, in order to avoid shrinking of the layer output. A typical convolutional layer consists of many kernels, whose quantity is called the number of units in a layer, or the channel width of the layer. Intuitively each kernel in a layer is trying to detect a *feature* in the input layer. By having multiple convolutional layers, it can be considered that earlier kernels are trying to search for basic features in a small area of the input. Similarly, kernels in later layers are seen as aggregating the information from previous layers, to search for more complicated features of the input.

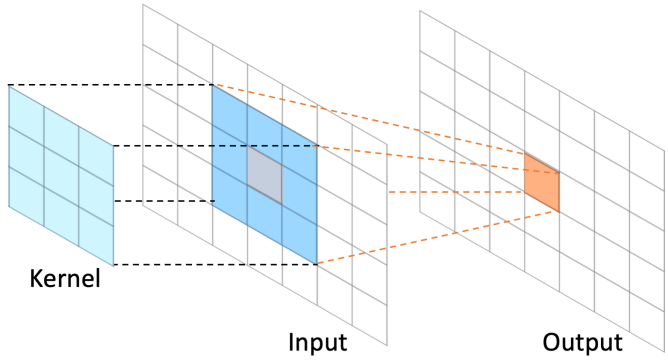


Figure 2.1: An example of convolution operation on input with a small kernel

Since there are a large number of kernels in a typical CNN, there are many parameters in the form of mathematical matrices, which are together called model weights or coefficients. These coefficients are estimated, or learned, during the training process of the deep learning model using the training data available. The training of the neural network is performed through the back-propagation algorithm [8], with the help of gradient descent.

The back-propagation algorithm works on the basis of an error function, which is computed using the difference in correct output and the predicted output by the neural network. Since this thesis only looks into supervised learning [7], it is assumed that a correct output label is available for every intended input during training. To get the predicted output for available input data, a normal matrix multiplication operation through all the layers is performed first, in forward direction. Next, the error is computed and the method further calculates the gradient of the error function, with respect to the neural network's weights. The estimation of the gradient then proceeds backwards through the neural network, i.e. the gradient of the last layer weights is calculated first and the gradient of the first layer weights is computed last. This iterative backward and forward flow of information, to compute error and its gradients, allows for an efficient and fast estimation of coefficients for each layer of the CNN.

With sufficient data available, the training of the neural network allows the conversion of information from the input data to the coefficients of the model. This results in many computations being performed during the training, which is typically performed on high end GPUs with access to large amounts of memory (>10GB). Once a neural network is sufficiently trained, it can be deployed as a service to perform inference on the unseen

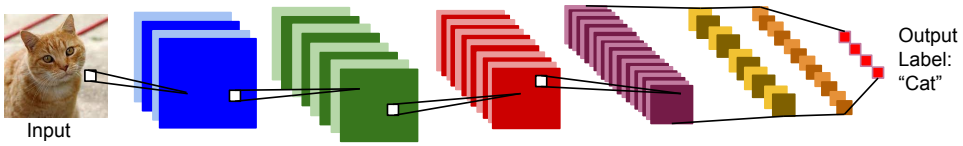


Figure 2.2: Deep neural network example for image classification

data. Inference refers to the execution of a trained model to perform its intended task, such as predicting a label of an image in image classification task or recognizing an activity in Human Activity Recognition (HAR) task.

In this sense, the life-cycle of a neural network has two distinct phases: *training* and *inference*. In Chapter 7 we argue that a neural network can be treated as a dynamic entity and can have one more phase: *update*, which involves a small update to the neural network, such as, changes to its architecture or re-training the model.

2.2 NEURAL ARCHITECTURES

As discussed in Chapter 1, the neural architecture captures the manner in which all the layers are arranged to construct the whole neural network. The simplest architecture can be composed by linearly connecting the layers in a chain structure, where any layer is only connected to two other layers. It receives data from one layer and passes its output to the next. In this thesis, such a CNN is also referred to as "Plain CNN", an example of which is illustrated in Figure 2.2. In this illustration, the input data, in the form of an image, passes through a sequence of linearly connected layers. The final layer of this neural network is a predictor or a classifier for the expected output.

Although the Plain CNNs are considered to have the simplest architecture, there are numerous design choices to consider in building one such model. Typically there are multiple layers in a CNN, with each layer having specific parameters depending on its operation type. For instance, every convolutional layer will have a few parameters to specify, a few of which are number of kernel units, kernel size, stride and padding. For example, one of the experiments in Chapter 3, for the CIFAR-10 dataset (the next section explains the datasets used in this thesis), consists of 10^8 possible configurations or design choices to build a plain CNN.

However, very deep neural networks can be hard to train, because of the so-called vanishing gradient problem. During training with stochastic

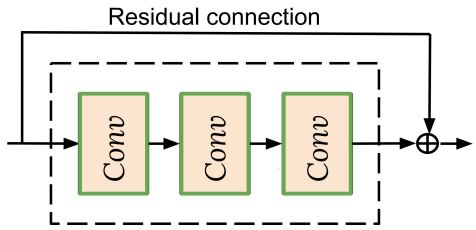


Figure 2.3: A residual connection with a block of convolutional layers

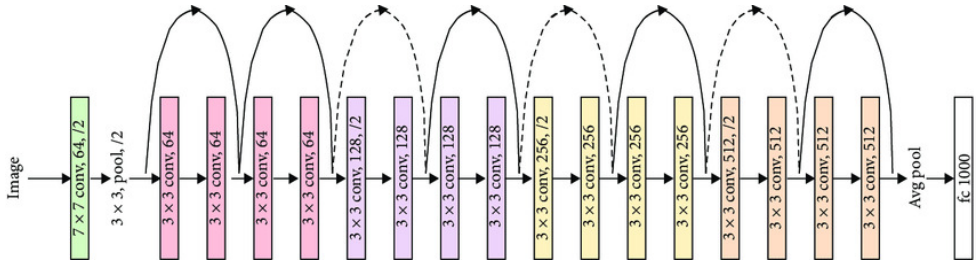
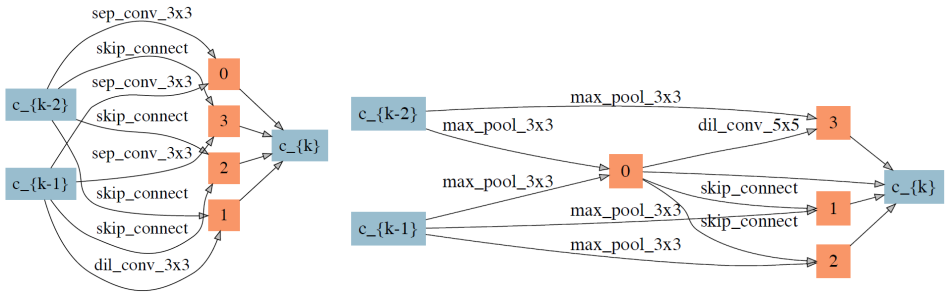


Figure 2.4: Standard Resnet-18 architecture, with many residual connections [5]

gradient descent [13], as the gradient is back-propagated to earlier layers, continual multiplication may make the gradient rather small or even zero. Hence, with a deeper model, the performance gets saturated or degrades quickly. One of the ways to solve this problem is to introduce shortcut connections between some layers, which allow the gradient to flow easily through earlier layers. Figure 2.3 illustrates the shortcut connection, bypassing a group of convolutional layers. This shortcut connection is also called *residual connection*, skip connection or identity connection.

The family of neural networks with many such residual connections was proposed in [5] and the architecture was named *Resnet architecture*. Depending on the number of layers, a number is usually added to a standard Resnet architecture. For example, Figure 2.4 illustrated the standard Resnet-18 architecture with 18 layers and multiple residual connections. It is noteworthy that these are standard Resnet architectures as proposed by the original work. It is possible to have Resnet-style neural networks with multiple residual connections bypassing any number of layers in between.

As discussed in Chapter 1, there is an important and increasingly popular subgroup of NAS approaches called one-shot methodology, consisting of algorithms focused on cell-based neural architectures. These algorithms search for a small sub-network called a *cell*, which are then linearly connected to form the complete neural network. The composition of the final



(a) Normal cell (b) Reduction cell

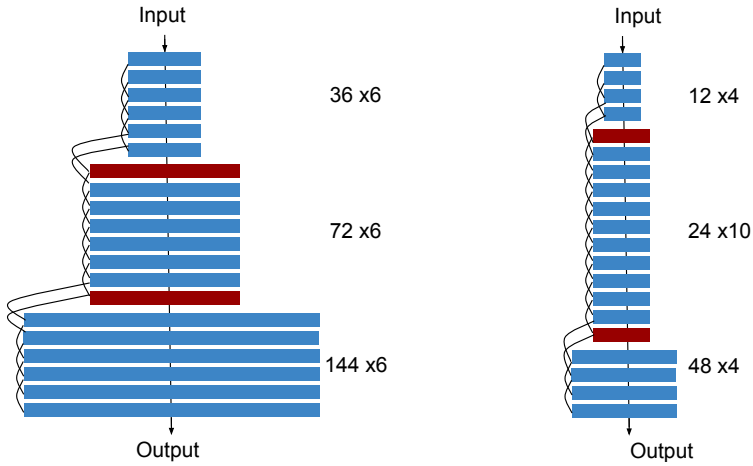
Figure 2.5: Cells found by one of the NAS algorithm: SNAS (mild constraint) [15]

neural network, established through the width of the cells and the depth of the connections, is manually designed while being influenced by the available GPU memory.

The cell-based NAS typically discovers two types of cells, namely, a normal cell and a reduction cell [14]. Figure 2.5 illustrates an example of cells as found by the SNAS methodology [15]. The cell itself is designed to take the output of the two previous cells and consists of an acyclic graph of various nodes. In the figure, blue nodes are input/output nodes of the cell and orange nodes are the intermediate nodes. Each edge between an input node and an intermediate node is either a convolutional operation or a skip connection. The output of all the intermediate nodes is then concatenated to produce the final cell output.

The normal cell is designed to maintain the feature map size of the input, whereas the reduction cell reduces the feature map size. Feature map size refers to the intermediate data that is transferred from one convolutional layer to the next layer in a CNN. The complete neural architecture is generated by forming a linear connection of the normal cells, interrupted by a few reduction cells at regular intervals. The neural architectures created by repeating the same cell possess an inherent flexibility to be able to form neural networks of different sizes. Individual cells can have variable channel width, i.e. can be wide or narrow (depending on the number of kernels it has).The variable frequency of cell repetition in the neural network may further add to their flexible nature.

As discussed in [16], cell-based NAS has some advantages over other methodologies. Firstly, the search space of the NAS algorithm is reduced as the algorithm only searches for a small sub-network, a *cell*, which is a small part of a complete neural architecture. Secondly, these cells can be transferred to and re-used in different datasets and domains.



(a) SNAS original architecture (2.9M parameters, 97.02% accuracy)

(b) SNAS^c Architecture for an edge device (0.3M parameters, 96.14% accuracy)

Figure 2.6: Cell-based architectures for CIFAR-10 with three blocks of normal cells partitioned by two reduction cells (in red). FxN to the right of each block represents the number of channels per cell in the block $\times$ number of cells per block.

We did an exploratory work, to understand the suitability of cell based architectures for edge devices [17]. Unsurprisingly, it was discovered that the original cell-based architecture designed by SNAS [15] for a high-end GPU is easily adaptable to consider resource limitations and therefore suitable for an edge device as well. It was observed that the number of operations per cell have to be drastically reduced, or in other words, channel width has to be minimized, to make a cell-based architecture be deployed on an edge device. Figure 2.6 illustrates the difference in channel widths and placement of reduction cells between original SNAS architecture and another derived architecture for an edge device. This new architecture sees a drop of $\approx 1\%$ in the accuracy, but has far less parameters. This translates to lower memory requirements, fewer arithmetic operations and hence, lower energy costs. It is important to note that the new architecture is formed from the same SNAS cell structure, where the channel width per cell has been reduced.

However, while a cell-based neural network may be extremely efficient, these are not always easy to understand, train and implement. Most medium complexity tasks and domains such as human activity recognition [18], earth sciences [19, 20] and astronomical studies [21] deploy plain convolutional networks as they are considered sufficient as well as easy to under-

stand by scientists with a non-AI background. Therefore, a large part of our work in the domain of NAS is focused on plain CNNs. In Chapter 5, we further extend this work to include ResNet-style CNNs in the design space of search algorithm.

2.3 DATASETS

In later chapters, the CIFAR-10 [22] and the Pascal VOC [23] datasets for image classification and the PAMAP2 [24] dataset for Human Activity Recognition (HAR) are used for different experiments. PAMAP2 has data from body-worn sensors and predicts the activity performed by the wearer, while Pascal VOC and CIFAR-10 are multi-label image classification datasets with 20 classes and 10 classes, respectively.

CIFAR-10 [22] is a labelled set of 60,000 images, bifurcated into training and test sets in the ratio of 5 : 1. The images are of size $32 \times 32 \times 3$ and are divided into 10 classes. We reserved 5,000 images from the training set, to use them for validation during the search process. The test set was eventually used to evaluate the final accuracy of the neural network, which is what we report in this thesis. Standard data augmentation techniques were deployed [25, 26], which include small amounts of translation, crop, rotation and horizontal flip. Data augmentation techniques create different data from the available dataset, by altering them slightly in various ways. This ensures that a newly created data-point still retains the essence of the original class, and at the same time allow the deep learning model to train from new inputs. These techniques are useful to improve the performance and outcomes of the neural network in question.

Pascal VOC [23] has a set of around 17,000 labeled images belonging to 20 categories. It is sometimes referred to as just VOC in this thesis. The image categories are some commonly found objects such as 'cat', 'dog', 'car' etc. The images vary in their dimensions and were downsampled to a fixed resolution of 384×384 with 3 color channels. 20% of the images were reserved for validation and testing and the rest were used during the training process. It is important to note that VOC is an imbalanced dataset, where some of the categories are significantly underrepresented in the dataset. For such types of datasets, accuracy may provide an inaccurate picture, since a high accuracy (or low error) is achievable by a no-skill model that only predicts the majority class. In the next section, we discuss other metrics that can be used to evaluate such imbalanced datasets. There are many threshold based metrics and rank based metrics which are considered more suitable

to evaluate models for such imbalanced datasets, we discussed some of these metrics in the next section.

The markedly different PAMAP2 dataset [24] compiles recordings from body-worn sensors. The input is organised as time-series data from a total of 40 channels from three Inertial Measurement Units (IMU) along with a heart rate monitor. The person wearing these sensors performed one of the twelve different activities in everyday life. We ignored some of the optional activities provided in the dataset. For a fair comparison with other papers, the validation and the test sets were the same as in the other papers working with this dataset ([27, 28]), i.e. the recordings from participants 5 and 6 were used as validation and test sets respectively. To prepare the data, firstly, the recordings from all IMUs were downsampled to 30 Hz and secondly, the data was segmented through a sliding window approach, with a window size of 3s (100 samples) and step size of 660ms (22 samples). For appropriate data augmentation, we moved the sliding window using different step sizes while the window size was kept the same at 3s.

Both PAMAP2 and Pascal VOC are imbalanced datasets, though the imbalance in VOC is much higher. In PAMAP2, the ratio between minority and majority class representation in the dataset is $\approx 1:4$, whereas the same in VOC is $\approx 1:20$. Moreover, one particular class label: *person*, is present in the dataset with a very high frequency.

2.4 NEURAL NETWORK EVALUATION

In order to find a good neural network, we need to define the metric which evaluates a neural network and determines which neural network can be considered *good*. The most popularly used metric to evaluate a neural network is classification accuracy, which is computed as the number of correctly predicted input frames to the total number of the CNN input frames. For an imbalanced dataset, which means that some of the classes are overrepresented in the data set, classification accuracy is not always the best choice to evaluate the CNN. Instead, F-1 score, precision, recall, Area under curve for precision recall (PR-AUC) are some of the metrics used for neural networks for imbalanced datasets [29].

One of the metrics we use for such datasets in this thesis is the weighted F1-score ($F1_w$) and mean F1-score ($F1_m$). These scores are computed using precision and recall for each class, and weigh the classification of each class based on the ratio of class representation in the dataset. Precision is the ratio of true positives to all positive predictions for a class. Whereas recall is the

fraction of true positives predicted by the model from among all positive labels for a class. Using precision and recall values per class, the F1 scores are then computed as:

$$F1_w = \sum_i 2 \times \frac{n_i}{N} \times \frac{precision_i \times recall_i}{precision_i + recall_i}$$

$$F1_m = \frac{2}{N} \times \sum_i \frac{precision_i \times recall_i}{precision_i + recall_i}$$

Here, n_i is the number of samples per class and N represents the number of data points in the whole dataset. In comparison to the classification accuracy, especially for imbalanced datasets, the F1-scores provide a better judgement about the performance of a neural network. An F1-scores in essence, is based on threshold based class assignments, which means a threshold value (> 0.5 in our work) is used to decide whether an output from the classifier can be treated as an assigned label (to a class) or not. The classic accuracy evaluation is a threshold based metric as well.

For PAMAP2 based evaluations, where the imbalance between classes is not too high, accuracy is still usable during our NAS algorithm, which is presented in the next chapter. F1-scores are only evaluated in the end, in order to compare with some other state-of-the-art methodologies presented in the HAR domain with PAMAP2 dataset.

Another metric for imbalanced datasets, *precision recall - area under curve* (PR-AUC), is calculated as the average of precision scores calculated for each recall threshold. A PR curve is plotted as shown in Figure 2.7, where precision is on the y-axis and recall is on the x-axis, and area under the curve is considered the evaluation metric. It is desired that both precision and recall are high, and the optimal PR curve is when the plot reaches the upper right-hand corner where precision and recall both are 1. However, a trade-off exists between precision and recall [30].

The intuition behind the suitability of PR-AUC for imbalanced datasets is that, since PR-AUC focuses on the fraction of true positives among positive predictions, it tells how relevant the correctly labeled data points are. This is important for under-represented classes in the dataset. Also, the PR-AUC value represents the performance of a neural network across many thresholds, rather than a single value. Hence, PR-AUC is able to reveal the differences in the performance of a neural network that may go unnoticed when using accuracy or F-1 score for highly imbalanced datasets [31].

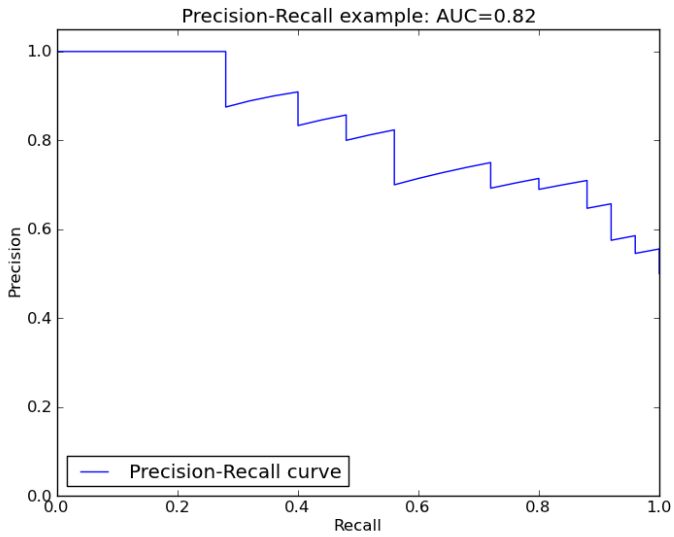


Figure 2.7: An example of area under curve for precision-recall curve

For Pascal VOC, the accuracy was used during initial experiments, which always reported more than 97% accuracy, as the over-represented classes were often correctly predicted. Hence, in this situation, the F1-score is more useful to perform comparisons between partially trained models (during the NAS, see Chapter 3). However, once a CNN is fully trained, the PR-AUC is more insightful for highly imbalanced datasets [30]. The PR curve represents various different levels of thresholds and has many F1-score values for various points on its curve. Therefore, for Pascal VOC, PR-AUC is the metric used in our experiments in Chapter 5 to evaluate the fully trained CNNs, although F1-score is used to compare the performance of a partially trained CNN during the NAS.

2.5 INTEROPERABILITY

In any environment with multiple products and systems, it is desired that all interfaces are well understood and are capable of comprehending each other. There are many powerful deep learning languages, toolsets and frameworks and it is possible to have an environment where all of these are present in at least one of the devices. Further optimizing deep learning models for specific hardware is difficult and since each hardware target (cloud/edge, CPU/GPU, etc.) has different capabilities and characteristics,

the problem becomes extremely hard and complex. Models from a variety of frameworks need to run on a variety of platforms. It is very time consuming to optimize all the different combinations of frameworks and hardware. A solution to train in any framework but being able to communicate anywhere on the cloud or edge is needed. Keeping this in mind, all our works use the Open Neural Network eXchange framework (ONNX) [32]. ONNX is well suited for this task as it encapsulates architecture and coefficients in a single modality and is widely seen as a solution to the interoperability problem concerning different deep learning frameworks. Most of them already allow exporting or converting the model to the ONNX format, such as PyTorch (and Caffe2, which got merged with Pytorch) [33], Keras [34], Tensorflow [35], Apache MXNet [36], Microsoft Cognitive Toolkit [37]. Converted models to ONNX can run on a variety of platforms and devices directly using ONNX Runtime [38].

Part I

NEURAL ARCHITECTURE SEARCH

The first part of the thesis is focused on the search for efficient neural network architectures, which are suitable to be deployed on resource-constrained edge devices. We explain the novel genetic algorithm based approach, called Evolutionary Piecemeal Training, which forms the basis of this thesis and further exploration towards adaptive applications in the next part of this thesis.

Chapter 3, Constrained Evolutionary Piecemeal Training, gives a brief introduction to the algorithm. Which searches for an efficient neural architecture within a constrained search space, to ensure that the best model can fit into the target edge device.

Chapter 4, Evolutionary Piecemeal Training as Dynamic Optimization, further explains the working of the algorithm, which treats the search for neural architectures as dynamic optimization.

Chapter 5, Multi-Objective Evolutionary Piecemeal Training, explains the extension of the proposed algorithm to include more than one objective, such as the target hardware specific objectives.

3

CONSTRAINED EVOLUTIONARY PIECEMEAL TRAINING

This chapter explains the basic concepts of the proposed NAS methodology, called Evolutionary Piecemeal Training (EPT), with constraints placed to limit the architecture size of the neural networks in the search space. This algorithm treats the search for neural architectures as an optimization problem, where the objective of the optimization is to maximize the test accuracy of the resulting CNN. The constraints are put on minimum and maximum bounds on architecture parameters of the neural networks. This enforces that all the neural networks under consideration are restrained from becoming too large, and thus may be deployed on an edge device with limited resources. To validate the algorithm, the search begins with random untrained models, and achieves fully trained models with a competent architecture, on CIFAR-10 and PAMAP2 datasets.

This Chapter is based on:

- **D. Sapra** and A. D. Pimentel *"Constrained evolutionary piecemeal training to design convolutional neural networks"* [39], in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, © Springer. **Best Paper Award**

and its extended journal paper:

- **D. Sapra** and A. D. Pimentel *"Designing convolutional neural networks with constrained evolutionary piecemeal training"* [40], in *Applied Intelligence*, © Springer.

3.1 INTRODUCTION

Popular NAS methodologies use evolutionary algorithms [41, 42] and reinforcement learning [43, 44], however, they need numerous GPUs consecutively, to prepare, run and converge the search process, and can consume tens to thousands of GPU-days. Many of these approaches rely heavily on resource-intensive training to direct the search algorithm, which is usually considered to be an isolated and separate task, to estimate the model performance. NAS is examined from a different viewpoint in our work, by looking at the possibility of searching for efficient architectures *during* a modified and extended training process. This contradicts the conventional training methodology as a distinct function, required for accuracy evaluation.

Moreover, recent research in NAS approaches is mainly focused on datasets in the image classification domain, specifically CIFAR-10 [22] and Imagenet [45]. This has contrived novel and intricate search spaces typically matched to the vision based tasks. These innovative search spaces are usually derived from previous hand-crafted architectures, for example, residual connections [5], cell based designs such as inception [46], dense net [47] or generated in a graph-like fashion [48]. Even though these innovative search spaces exhibit ingenuity as well as efficiency, they are complex to understand, design and train. Many tasks and domains that are of medium complexity such as human activity recognition [18], earth sciences [19, 20] and astronomical studies [21] utilize plain Convolutional Neural Networks (CNNs) in their research. They are considered sufficient and are well understood by scientists and researchers who do not come from a background in Artificial Intelligence (AI).

Therefore, a tool, which is convenient to be used by non-AI experts from various domains, and one which finds an efficient CNN in a timely manner, is vital to simplify the design process of neural networks and subsequently democratizing AI. For example, a neural network for breast cancer classification has been proposed in [49], which was manually designed and further optimized. For a non-AI expert, the absence of relevant tools and limited knowledge about the neural network architecture design can be a hindrance in effectively utilizing AI models in their respective domains. In this direction, the chief contribution of this work is a novel algorithm for NAS, called Evolutionary Piecemeal Training.

Our algorithm explores the constrained design space of CNNs for the selected task and attempts to discover an efficient architecture while con-

verging in a restrained amount of GPU hours. The CNN models that are created and altered during the search are constrained by a minimum and maximum value for each of the architecture parameters. These bounds have been set up to ensure that the size of all CNN architectures is regulated, and at the same time, it limits their potential to outgrow the availability of hardware resources. This is one of the crucial considerations for models intended to be implemented on embedded systems, for instance wearables in the Human Activity Recognition (HAR) domain. High computational and memory demands by large neural networks may result in inefficient utilization of the limited resources on the embedded device.

The proposed NAS technique explained in this chapter is based on a population based computation method which allows a group of neural networks to train simultaneously. During the training process, random CNNs from the population are chosen and evolutionary operators applied to them. These evolutionary operators are designed in such a way that they lead to small architecture modifications and hence guide the exploration of the larger search space. Every new architecture derived through modification is invariably partially trained, since the parent architecture was already undergoing the training. In every subsequent iteration, CNNs continue to train while some of them are subject to architecture modifications. When the algorithm converges, the best individual models (with high accuracy) are chosen from the population. These selected CNNs can be post-processed and trained for more epochs.

In particular, we use a genetic algorithm in our methodology, which was chosen after considering various factors from amongst different meta-heuristic algorithms, such as, Simulated Annealing and Particle Swarm Optimization [50]. The most important factor was the ability of the algorithm to simultaneously allow the training to continue, while searching for an appropriate neural architecture. To this end, the genetic operators (mutation and recombination), can be defined in such a manner, that there is minimum disruption to the training process, while the large design space of architectures is explored. Additionally, these algorithms are well studied in the multiple-objective search domain. At the onset of this research, the potential to extend to multi-objective search was taken into consideration. Moreover, other NAS methodologies, which are based on evolutionary algorithms, prominently use the genetic algorithm in their research [41, 51].

In this chapter, the experiments and results are presented with two distinct datasets. The first one is the PAMAP2 [24] dataset, for the HAR domain, where the data is measured from body worn sensors on a person's

body to anticipate the activity being performed by the human wearer. The second dataset is the CIFAR-10 dataset, which is popularly used in the domain of image classification. The versatility of the proposed approach is demonstrated through the use of two markedly different datasets in terms of both input data type and format.

3.2 RELATED WORK

Various research works have been published recently demonstrating proficiency of NAS techniques. They can be partitioned mainly into three categories, namely Reinforcement Learning (RL) based, evolutionary algorithms and one-shot architecture search. Both reinforcement learning and evolutionary based algorithms mandate the complete training of the neural network at each search iteration for performance evaluation. In RL based methods [43, 44, 52], the validation performance, or accuracy, of the trained model guides the reward towards the RL agent. When the agent is continuously rewarded for finding better architectures, the search is slowly steered towards neural networks with higher performance. Any RL approach resolutely demands a suitable agent, which frequently happens to be a complicated model or perhaps another neural network. The construction of the agent and its optimization involve substantial effort towards designing and subsequent fine tuning.

Evolutionary methodologies [42, 51, 53–55] utilize genetic algorithms to discover the efficient neural architecture in a large search space. Evolutionary algorithms have also been successfully deployed for CNN optimizations, such as, for compression [56], pruning [57] and hyper-parameter optimization [58]. Evolutionary NAS algorithms work with a population of possible CNN candidates, where each one of them is trained and evaluated at every iteration. With subsequent iterations, the models in the population get selected, rejected and modified depending on their accuracy and other control variables of the algorithm. The aim is to improve the population's average performance with time. Eventually when the algorithm converges, it has discovered an architecture for a high performing neural network. Our work also utilizes an evolutionary algorithm for architecture modification, where the key difference is in the manner training and architecture modifications are interwoven in the algorithm to conduct joint search for both weights and architecture.

Unfortunately, most of these approaches demand intensive computational resources to train hundreds or thousands of neural network architectures.

For example, the RL method in [44] trained more than 10,000 models, involving over a thousand GPU days, while another adept evolutionary search [59] required 56 GPU days to finally converge. Other works have utilized proxy tasks, for example, hyper-networks [60], predictors [52, 61] and controllers [43] to fasten up the search process. However, they still continue to demand abundant planning and time to be implemented before the actual search commences. In direct contrast, our algorithm does not require helpers and proxy tasks and still converges in a reasonable time.

One-Shot NAS methodologies are based on the concept of a trained super-network, consisting of all the possible sub-networks within. The entire super-network may have to fit in the GPU memory during the NAS execution, which results in a highly restricted architecture size, and it typically results in a discovery of a small sub-network, called a cell with a limited number of operations. The cell that is discovered through one-shot search is sufficiently repeated and connected in an appropriate manner, to eventually form a neural network that will perform the intended task. See Chapter 2 for more explanation about cell-based neural architectures.

DropPath [62] is an example of a one-shot search approach, where a path is dropped out with a fixed probability, and by randomly removing different paths, a new sub-network is formed. The pre-trained super-network is then used to evaluate and eventually discover the best sub-network architecture. DARTS [63] additionally proposed an architecture parameter for every path and by employing the standard gradient descent to train the weight and architecture parameters together. Other approaches attempt to be more efficient by utilizing other proxy tasks, for instance [64] proposed a memory-efficient algorithm to update fewer paths while searching. Aside from posing a meta-architecture design challenge, the models based on replication of single cells, may not be suitable to various domains.

3.3 METHODOLOGY

In this section, we go into the details of our proposed methodology, Evolutionary Piecemeal Training, describe its key concepts and the complete algorithm. Piecemeal training makes a reference to the training of a CNN with a small 'data-piece' randomly taken from the whole dataset, the size of which, referred to as δ, can vary from 5% to 20% of the dataset. The conventional training of a neural network is regularly interrupted through an evolutionary operator, at intervals determined by δ. The operator modifies some parameters in the model architecture, and subsequently permits the

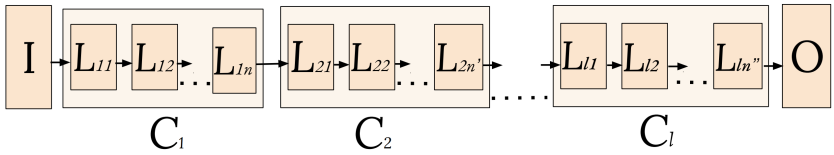


Figure 3.1: A general cluster based architecture with l clusters, where each cluster defines its layer type along with the constraints it enforces on member layers.

continuation of the training process. Numerous CNNs begin this training in parallel, creating a population that is subject to architecture modifications after each iteration of piecemeal training. The models that do not perform as well as other models are removed from the population. Conceptually, in the context of neural network training, this can be envisioned as the early termination of candidates showing no promise in their ability to reach a high accuracy.

All possible neural network architectures with their configurations and constraints constitute the search space for our work. More specifically, we focus on linearly connected plain CNNs, where layers are only connected to their consecutive layer, and do not have complex connectivity through residual connections or branches. We anticipate that the search methodology can be extended to more complicated search spaces. However, this particular research is focused on plain CNNs, which are used by many non-AI experts in their respective domains and may be considered sufficient for the given task [18, 19, 21].

Figure 3.1 illustrates a general cluster based architecture, where similar consecutive layers are grouped into clusters. Additionally, every cluster in the architecture places constraints on its number of layers, on the number of units per layer, and on other layer specific parameters such as the kernel size in a convolutional layer.

For our experiments, all clusters and their boundary definitions are construed before the start of the search algorithm. All possible permutations of layers and their hyper-parameters together represent the whole search space. This architecture search space is usually not a trivial space to navigate. For example, the search spaces for experiments in Section 3.4.1 have 10^8 (CIFAR-10) and 10^5 (PAMAP2) possible architectures. This search space definition is encoded in the form of a genotype to ensure the availability of a factory to generate new neural networks, and also to warrant that the evolutionary operators adhere to the cluster constraints.

3.3.1 *Population based training*

We employ a population based training process where an initial population of neural networks is randomly created from the defined gene pool. In each iteration, all candidates of the population are piecemeal-trained and then evaluated using the validation set. Depending upon the available resources, all candidates can be trained in any combination of parallel and sequential manner. The size of the population is kept constant throughout the algorithm, though the candidates of the population keep changing as they are altered through the evolutionary operators applied in each iteration. The number of candidates in the population needs to be large enough to maintain enough diversity of CNN architectures in the population, while still satisfying the constraints applied to it.

3.3.2 *Evolutionary operators*

The evolutionary algorithm needs to sufficiently explore the huge search space to ensure that a good model with high performance can be discovered in a reasonable time. To this cause, at each evolutionary step, architectures of some of the models in the population are modified through one of the evolutionary operators, i.e. the recombination or mutation operator. While mutation does small changes to one layer at a time, recombination exchanges some layers in one model to another to create significantly different models. The mutation operator explores the search space closer to the existing population, and in contrast, the recombination operator explores a wider design space by generating diverse architectures. Next chapter, Chapter 4 provides more details about how these two evolutionary operators are designed to maintain diversity in the population during the search process. The number of evolutionary operators executed in each iteration is controlled through a pre-defined mutation rate (P_m) and recombination rate (P_r).

Algorithm 1: Mutate

Inputs: T_{parent}, ρ_m

1 $L_m \leftarrow \mathtt{randomLayer}(T_{parent})$
2 $L_m \leftarrow \mathtt{ChangeParameterOf}(L_m, \rho_m)$
3 $T_{child} \leftarrow \mathtt{Merge}(T_{parent}, L_m)$
4 **return** T_{child}

Mutation operates on a CNN and randomly selects one layer to change one of its hyper-parameters, such as the number of kernel units in the layer or the kernel size. We employ the Net2Wider operator from [65] to broaden the layer by increasing the number of kernel units. On the other hand, to shrink the layer, we use a pruning process [66] to reduce the number of units, by removing the least significant kernels in terms of their activation weights. Kernels are radially zero-padded or cropped from the outer edge, when their size changes because of mutation. The mutate operator is described in Algorithm 1, which accepts a topology T_{parent} as an input and returns the mutated topology T_{child}.

Furthermore, the mutation operator is devised to be function-preserving [65] in nature to make sure that mutation does not disrupt the ongoing training of the neural networks. Any change to the architecture will invariably cause an additional loss in the training process. The functions in mutation operator were particularly chosen since these are either totally, or at the minimum, partially function-preserving, implying that the loss drawn from these operators is as minimum as possible and recovers quickly during later piecemeal-training iterations.

In direct contrast to mutation, the recombination operates on two neural networks and swaps all their layers in a cluster. The swap is carried out for only one randomly selected cluster position. Figure 3.2 shows an example of the recombination operator, which swaps different numbers of layers from the clusters C_2 in two different neural networks. Since all models have exactly the same number of clusters, it follows that the layers that are exchanged are approximately in the same stage of neural computation, and hence the new models need minimum repair to the architecture to remain valid. Algorithm 2 shows the steps for the recombination operator, which accepts two parent topologies and returns children topologies after the recombination operator has been applied.

It is important to note that the recombination is not a function preserving operator. However, they are required in the algorithm to introduce and maintain diversity [67] by introducing significantly varied models into the

Algorithm 2: Recombination

Inputs: $T_{parent1}, T_{parent2}$

1 $k \leftarrow randomClusterPosition(T_{parent1}.Num_{cluster})$

2 $T_{child1}, T_{child2} \leftarrow SwapClusterAt(k, T_{parent1}, T_{parent2})$

3 **return** T_{child1}, T_{child2}

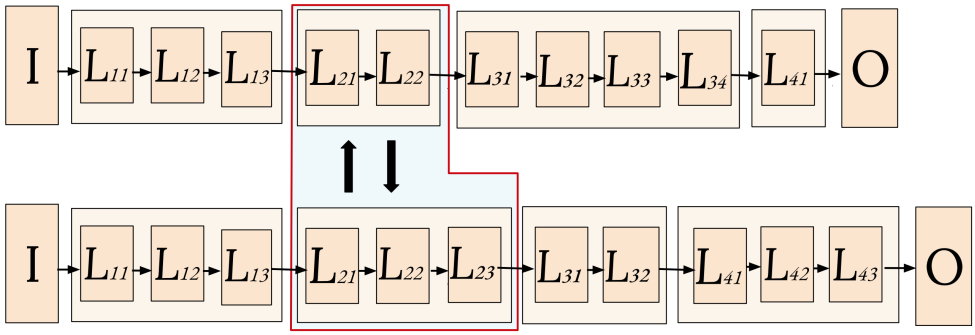


Figure 3.2: Recombination operator applied to two neural networks, where the cluster C_2 gets swapped. The cluster has a different number of layers in each model.

population. This is achievable due to the fact that the total number of layers being swapped is not the same. To diminish the adverse effect of loss incurred by the recombination operator, a cooling-down approach is applied to the recombination rate. During the early iterations, when the training loss has not yet started to converge and is at a high value, more swaps are allowed as compared to later iterations, when the training loss is low.

Together, these evolutionary operators are responsible for traversing the large design space of neural network architectures in an efficient manner. Additionally, these operators are responsible for making sure that the cluster constraints are always adhered to. The mutation operator never allows a layer to expand or narrow beyond the cluster defined boundaries, and it also ensures that other layer hyper-parameters conform to the cluster specifications as well. The recombination operator swaps clusters which are already within their bounds, thus maintaining the constraints.

3.3.3 *Selection and Replacement*

One of the most important features of the population based evolutionary approach is that every subsequent population attempts to be better than the one in the previous iteration. This is achieved through selection and replacement policies geared towards retaining the better performing candidates at every step. A remove-worst strategy is employed to select the next generation of the population. However, the rejection rate is kept relatively low, around 2-5% of the total population. To keep the population size constant, individuals are selected and put back in the population using a non-elitist random selection policy. This means that every neural network

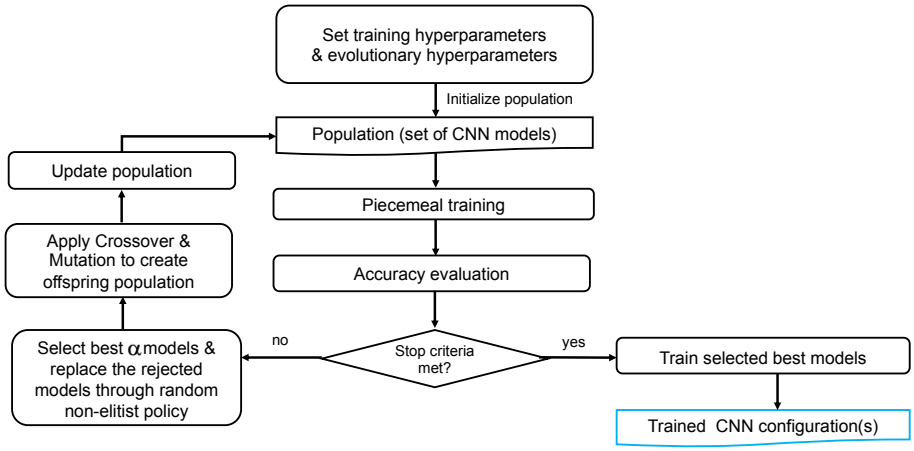


Figure 3.3: Workflow for the Evolutionary Piecemeal Training.

in the population has an equal chance of being selected to replace the worst performing model.

3.3.4 *Workflow and Algorithm*

We assemble all the concepts described and consolidate them to present a brief overview of the workflow (Figure 3.3), together with the complete algorithm (Algorithm 3). The algorithm begins with the initial set up of the configuration parameters for both training and evolutionary operators. The evolutionary inputs are, N_g: the number of iterations, N_p: the population size, P_r: Recombination rate, P_m: Mutation rate and α: the selection policy for each iteration. Additionally, the training parameters τ_{params} including optimizer choice, learning rate, batch size etc., and δ to determine the size of the subset of data to be used for piecemeal training are provided. All the evolutionary and training parameters are empirically selected after conducting a small number of initial experiments, in order to fine-tune the algorithm. The selection of evolutionary parameters is mainly guided by available computing resources, along with the complexity of the task. Whereas the training parameters are determined through a small grid search by training a few architectures from the population prior to commencement of the whole algorithm. These parameters stay constant throughout the algorithm, unless specifically stated.

A genotype represents the search space, and contains all the information about clusters and their respective constraints. The population is generated

using this genotype. The function $InitializePopulation()$ creates the population $\wp_o$, of N_p neural networks, and their initialization can be random or through training for a few epochs.

Algorithm 3: Evolutionary Piecemeal Training

Evolutionary Inputs: N_g, N_p, P_r, P_m, α
Training Inputs : τ_{params}, δ

$\wp_o \leftarrow InitializePopulation(N_p)$
for $i \leftarrow o$ .... N_g **do**
  $\wp_i \leftarrow PiecemealTrain(\wp_i, \tau_{params}, \delta)$
  $E_v \leftarrow EvaluateAccuracy(\wp_i)$
  $\wp_{best} \leftarrow BestSelection(\alpha, \wp_i, E_v)$
  $\wp_r \leftarrow random((1-\alpha) * \wp_i)$
  update $\wp_i \leftarrow \wp_{best} + \wp_r$
  $\wp_{rc} \leftarrow RecombinePopulation(\wp_i, P_r)$
  $\wp_{mu} \leftarrow MutatePopulation(\wp_i, P_m)$
  $\wp_{remaining} \leftarrow UnchangedPopulation(\wp_i)$
  update $\wp_i \leftarrow \wp_{mu} + \wp_{rc} + \wp_{remaining}$
end
$E_v \leftarrow EvaluateAccuracy(\wp_{N_g})$
return *BestCandidates*(E_v)

Once the start set-up is complete, the iterative core of the algorithm is initiated and this iterative algorithm runs for N_g generations. The population at every i^{th} iteration is called $\wp_i$. First, the function $PiecemealTrain()$ trains all individuals in the population $\wp_i$, with the random subset of the data. Next, $EvaluateAccuracy()$ evaluates the accuracy of every model in the population. Based on the accuracy values, $BestSelection()$ selects the α best individuals found so far ($\wp_{best}$), from the whole population. To keep rejection rate low, α is chosen to be a high ratio of $> 0.95 * N_p$, which is important to keep the focus on removing the poor performing architectures gradually from the population. This approach discourages the promotion of a model that is able to learn fast but is unable to finally reach a high accuracy. Afterwards, to keep the population size constant, $1 - \alpha$ neural networks ($\wp_r$), are randomly selected from survivors. The population $\wp_i$ is updated by replacing the population with $\wp_{best}$ and $\wp_r$. Random selection makes sure that subsequent generations do not get crowded with only one parent architecture, which got higher accuracy by chance due to the stochastic nature of training. The evolutionary operators, $RecombinePopulation()$

and MutatePopulation() select individuals ($\wp_{rc}$ and $\wp_{mu}$), with probability of P_m and P_r respectively, from the population to alter some of the neural network architectures. The actual alteration in performed through Algorithm 1 and Algorithm 2. The population is then updated with modified neural networks from $\wp_{rc}$ and $\wp_{mu}$, while the part of the population not undergoing any modification ($\wp_{remaining}$) remains unchanged for the next iteration of the algorithm.

After the iterations conclude, the algorithm evaluates all the remaining models in the final population and returns the best neural networks determined. These best models are post-processed and modified, if needed, and further trained to achieve final CNN configurations. Other hyper-parameter optimization techniques [68] can be utilized to find the optimal training parameters, in order to train the CNNs at this stage.

3.4 EXPERIMENTS

In this section, we present the experimental setup, in addition to the algorithm's evaluations, using the datasets: CIFAR-10 and PAMAP2. We describe the search spaces for both the datasets, with the constraints and the results achieved. We have utilized the Java based Jenetics library [69] for evolution based operators and computation, while the Python based Caffe2 [33] library was used for the training and accuracy evaluation. The neural networks were represented in the ONNX [32] format, which combines architecture and weights in one file, and facilitates the storage and transfer of the CNNs across different modules. All our experiments were performed on a single GeForce RTX 2080Ti GPU.

3.4.1 *Search Space*

We outline the search space specifications for CIFAR-10 and PAMAP2 in Table 3.1 and Table 3.2 respectively. In the CIFAR-10 experiments, the number of kernel units per layer were multiples of 16, which brings the total number of models in the search space to the order of 10^8. For PAMAP2, the number of kernels per layer are multiples of 8 and the total design points are to the order of 10^5.

Table 3.1: CIFAR-10 Architecture Search Space

Cluster Type	**Layers**		**Units/Layer**		**Kernel-size**		**Stride**
	β^{min}	β^{max}	η^{low}	η^{up}	K^{min}	K^{max}	S_t
C_1:Convolution	2	5	48	96	3x3	7x7	1
C_2:MaxPool	1	1	1	1	2x2	2x2	2
C_3:Convolution	2	7	80	320	3x3	7x7	1
C_4:MaxPool	1	1	1	1	2x2	2x2	2
C_5:Convolution	2	7	256	640	3x3	7x7	1
C_6:MaxPool	1	1	1	1	2x2	2x2	2
C_7:FullyConnected	2	3	128	1024	-	-	-

Table 3.2: PAMAP2 Architecture Search Space

Cluster Type	**Layers**		**Units/Layer**		**Kernel-size**		**Stride**
	β^{min}	β^{max}	η^{low}	η^{up}	K^{min}	K^{max}	S_t
C_1:Convolution	2	4	64	128	3x1	7x1	1
C_2:MaxPool	1	1	1	1	2x1	2x1	2
C_3:Convolution	2	5	96	256	3x1	7x1	1
C_4:GlobalMaxPool	1	1	1	1	2x1	2x1	2
C_5:FullyConnected	1	3	128	512	-	-	-

3.4.2 *Training Setup*

The CIFAR-10 dataset was trained for 80 generations, while the population size was kept at 80. The data size, δ, was set to 4,000 images ($\approx$ 8% of training data) to be used by the piecemeal-training. Every convolution and fully connected layer was appended by ReLu activations. Training was performed with the Adam optimizer [70], while the batch size was set at 80. Initial learning rate was set to $5e^{-4}$, with a step learning rate decay policy where the learning rate was reduced by $1e^{-4}$ at the interval of 20 iterations. The evolutionary selection probabilities P_m and P_r were both set to 0.3 at

the beginning. P_m stayed constant, whereas P_r was reduced to reach 0.01 at the last iteration.

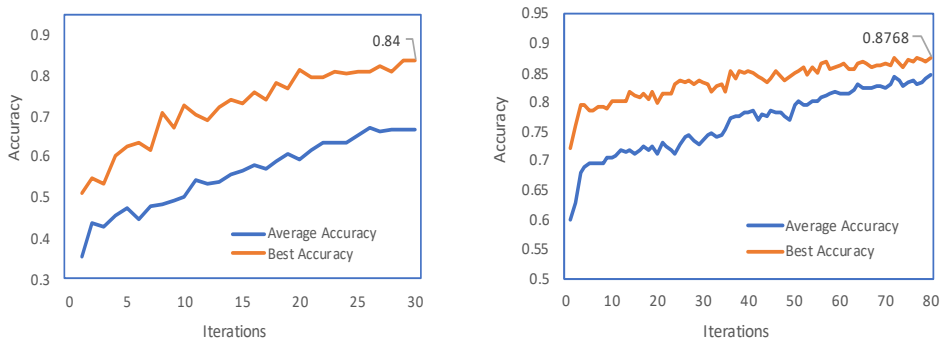
For the second dataset, PAMAP2, training was done during 30 generations with a population size of 50. The data size, δ, for piecemeal-training was set to 20,000 samples ($\approx$ 10% of training data). ReLu activations follow every convolution and fully connected layer. The training was performed with the Adam optimizer, the batch size of 100, and a learning rate of $1e^{-4}$. Evolutionary selection probabilities P_m and P_r were both initialized as 0.3. Similar to the CIFAR-10 experiment, P_m was kept constant, whereas P_r was reduced to 0.01 towards the end.

The CIFAR-10 models are substantially more memory consuming than the PAMAP2 models, which limits the amount of parallelism for training on a single GPU with 11 GB memory. For CIFAR-10, 4 parallel training threads could execute, while 7 simultaneous threads for PAMAP2 could run. The limit on the level of parallel executions was governed by the GPU memory available. Additionally, no Batch Normalization was used in order to fasten up the search, since it consumes more memory and therefore reduces the parallelism. Once the search finished, the best model found was altered, to have a batch normalization layer following every convolutional layer and was trained for 100 epochs more.

3.4.3 *Results*

The training curves for experiments on PAMAP2 and CIFAR-10 are depicted in Figure 3.4, where accuracy maximization is the only objective in consideration. These graphs depict the accuracy of the piecemeal training process, and do not include the post-processing and final training of the best found architectures. As the iterations continue, it can be observed that the average accuracy of the whole population generally increases, despite architecture modifications interrupting the training. The best accuracy of any individual in the population is similarly increasing gradually with each iteration. The best model in one iteration may be different from the best one in the next iteration. The best model(s) discovered at completion was trained further for more epochs to achieve the accuracy that is reported in Table 3.3 and Table 3.4.

The first experiment using the CIFAR-10 dataset consumed 2-GPU days and reached the best prediction accuracy of 92.5% on the test set. Table 3.3 compares our results with other evolutionary based NAS approaches. We understand that when we compare the accuracy of 92.5% to other pub-



(a) Training curve for PAMAP2

(b) Training curve for CIFAR-10

Figure 3.4: Training curves. Average accuracy refers to the average performance of the whole population at the given search iteration. Best accuracy refers to best performance of any individual model in the population.

lished works, it ranks slightly lower than the other efforts, however, the key difference is that the architecture space is defined for plain CNNs. There is a marked omission of architectural enhancements such as residual connections and cells in our architecture search space. In addition, advanced data augmentation like mixup [71] or cutout [72] were not deployed either. Other approaches commonly use a hybrid search space, which may include different cell modules or architecture blocks along with arbitrary residual connections.

Despite the lower accuracy, we emphasize the shorter convergence time, of only 2 GPU-days, when compared with other evolutionary NAS methodologies. As summarized in Figure 3.5 (a), the best CNN found by the search algorithm had 13 convolutional layers with addition of 2 fully connected layers.

The PAMAP2 dataset is an unbalanced dataset, i.e. some of the classes are over-represented in the data set. We compute both F1-scores as well as accuracy, in order to compare the result with other published works.

Our algorithm was able to achieve impressive results on the PAMAP2 dataset. The search took 10 GPU-hours, while the best neural network discovered after complete training was able to reach a prediction accuracy of 94.36%. Table 3.3 compares our algorithm's results against other published works. In direct comparison, the grid search [28] on neural networks for

Table 3.3: CIFAR-10 Accuracy Comparisons with Evolutionary Approaches

Model	**Search Space**	**GPU-days**	**Accuracy(%)**
CoDeepNeat [41]	hybrid	-	92.7
GeneticCNN [54]	hybrid	17	92.9
EANN-Net [53]	hybrid	-	92.95
AmoebaNet [42]	cell	3150	**96.6**
NSGANet [55]	hybrid	8	96.15
Evolution [51]	hybrid	1000+	94.6
EPT (ours)	plain CNN	**2**	92.5

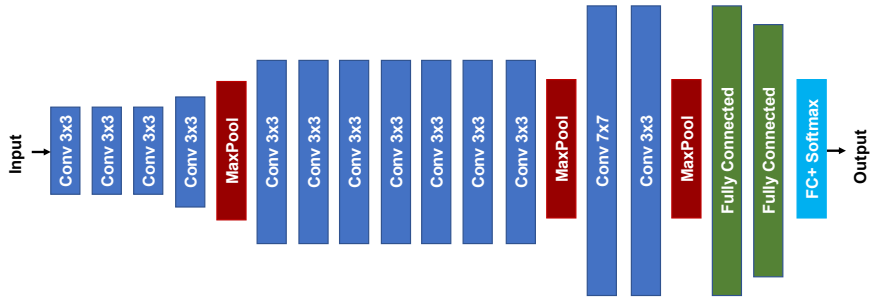
Table 3.4: PAMAP2 Accuracy Comparisons

Model	**Accuracy(%)**	**$F1_w$(%)**	**$F1_m$(%)**
Hand Designed [73]	93.13	93.21	-
Grid Search (CNN) [28]	-	-	93.7
D^2C [27]	-	-	92.71
D^2CL [27]	-	-	93.2
EPT (ours)	**94.36**	**94.17**	**94.36**

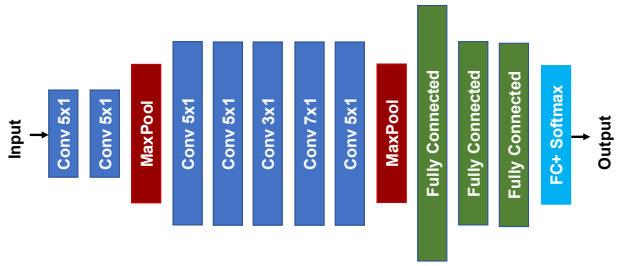
PAMAP2 was able to reach their best at 93.7% and another hand-crafted model [73] achieved 93.21%. These results clearly demonstrate that our methodology is more effective than the naive algorithms that involve simple approaches such as random search or grid search. The best neural network found had 7 convolutional layers and 3 fully connected layers as shown in Figure 3.5 (b).

3.5 SUMMARY

In this chapter, a novel approach called Evolutionary Piecemeal Training was presented, which traverses the search space of plain CNNs to find an efficient architecture from a constrained search space for a given task. The algorithm was validated on two different datasets, demonstrating the versatility of our method. We showed that for moderate complexity tasks such as the PAMAP2 dataset, our approach is better and more efficient



(a) CIFAR-10 Model



(b) PAMAP2 Model

Figure 3.5: Best neural networks found for (a) CIFAR-10 dataset and (b) PAMAP2 dataset. Every Convolutional layer is followed by a Batch Normalization and a ReLu activation layer.

than random or grid search methodologies. The next chapter focuses on describing the intricate characteristics of the methodology and how this algorithm is treated as a dynamic optimization problem.

4

EVOLUTIONARY PIECEMEAL TRAINING AS DYNAMIC OPTIMIZATION

As explained in the previous chapter, the search for a suitable neural architecture is treated as an optimization problem. This chapter further analyses the dynamic nature of the optimization in the EPT algorithm. This chapter defines a Gradually Saturating Objective Function (GSOF) in a dynamic optimization, its pertinence to the EPT methodology and the challenges hence faced. An adaptive population-diversity based approach is proposed, and its efficiency validated, to solve GSOF and improve the standard evolutionary algorithm. It is important to note that this chapter does not present an improved EPT algorithm, rather it discusses the challenges faced and solutions proposed during the design of EPT algorithm.

This Chapter is based on:

- **D. Sapra** and A. D. Pimentel *"An evolutionary optimization algorithm for gradually saturating objective functions"* [67], in *Proceedings of the Genetic and Evolutionary Computation Conference*, © ACM.

4.1 INTRODUCTION

In traditional optimization problems, all environment variables and constraints are previously known and remain static throughout the optimization task. In real life optimization problems, however, an environment may change due to several factors, such as fault occurrence, slow degradation, planned updates and modifications over a long period of time [74, 75].

These are called dynamic optimization problems (DOPs) wherein objective functions, constraints or the number of parameters change over time [76]. For static optimization problems the optimization model is predetermined and designed for a specific non-moving objective. If an environment changes infrequently after long time frames, the dynamic optimization problem can be treated as a sequence of static optimization tasks. However, in continuously changing dynamic optimizations, the model might require continuous adaptations along with the changing parameters and/or moving optimum.

Evolutionary Dynamic Optimization (EDO) [77] in literature is focused on recurrent or abrupt changes in the environment. There are various methodologies to detect sudden changes in the landscape [78, 79], memory based approaches to handle recurring behavior (e.g. [80, 81]), and prediction strategies to predict the moving optimum or the population suitable in the new environment [82–84].

In a gradually changing environment, these techniques are too complicated and somewhat of an overkill. Approaches based on maintaining diversity are more suitable in such scenarios. High diversity in the population restricts the convergence of the optimization algorithm to a small search space, consequently preventing it from getting stuck in a local optimum. This allows the algorithm to monitor diverse parts of the search space so the optimization can be efficient while the environment changes slightly with each iteration.

Our focus in the EPT algorithm is on a dynamic environment with gradually changing objective functions which have a tendency to saturate, hence turning the dynamic optimization into pseudo-static optimization after saturation. As explained in previous chapter, the objective of the search is to find a neural network topology that is efficient with high accuracy. A population of neural networks is trained in parallel on a dataset and their architecture is modified during the training using genetic operators. Figure 4.1 illustrates an example of a neural network performance during the training process, also referred to as training curves. The training curve represents the iterative performance of a neural network during the training and closely resembles an increasing saturating function such as functions from the power law or the sigmoidal family [85].

As is evident from Figure 4.1, the prediction accuracy during the training process is a continuous and slow moving target. The maximum achievable accuracy increases with each training epoch and eventually starts to saturate when the training is nearly complete. The high diversity in the popu-

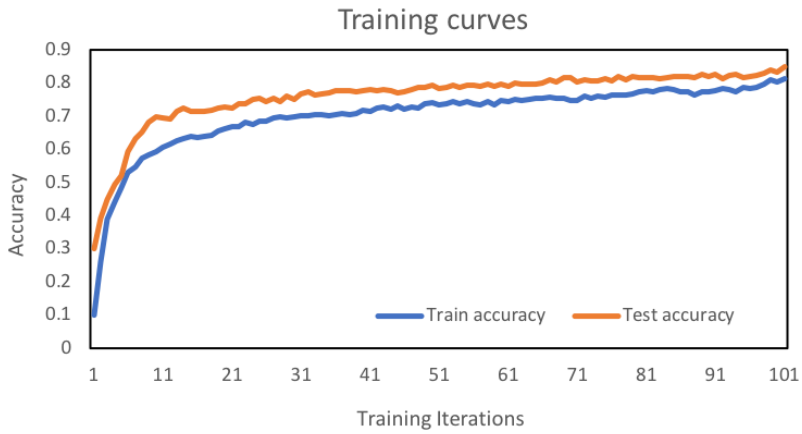


Figure 4.1: Example of training curves. Train and test accuracy are evaluated on the train and test data set respectively.

lation in saturated stages is not desired, since it might counteract the need for the algorithm to converge to good points now that the optimization problem is pseudo-static.

For Gradually Saturating Objective Functions (GSOF) in an evolutionary algorithm, we propose an adaptive diversity control approach to solve this dynamic optimization problem. We define two levels of diverseness within the population and modify the algorithm with disruptive recombinations and non-disruptive mutations while keeping the diverseness levels in mind. By introducing controlled diversity into the population through these genetic operators we are able to guide the optimization to achieve better results as proved by the experimental results.

4.2 RELATED WORK

Dynamic optimization problems (DOPs) are characterized by a variety of mechanisms that can cause a change in the problem environment during the optimization process. Some of the attributes that outline a dynamic behavior are frequency, severity and predictability of the change. In evolutionary solutions for DOPs, these environmental changes are handled in various ways such as memory based approaches, multi-population based techniques, prediction based methods and diversity based approaches. Some hybrid approaches such as memetic algorithms [86], which combine differ-

ent aspects of these approaches, have also been proposed over the years. An appropriate approach is chosen depending on the type of dynamism present in the optimization problem's environment.

For periodical changes, memory based approaches are suitable where some candidates from the population are stored for later use. It reduces the computation complexity by making good candidates readily available in recurring situations. Memory can be implicitly encoded in genotype [80] or explicitly stored externally [81, 87]. For sudden and irregular changes, the main concern is to detect when the change occurs and to adapt the population to be suitable for the new optimum as quickly as possible. A change can be indicated by population statistics [88] or external sensors [89].

Multi-population approaches divide the population into multiple sub-populations, and each one tracks the optimum in different promising search areas [90–93]. Sub-populations are generally independent of one another and each one might employ its own search technique or track different optimum in multi-objective optimization problems. Sub-populations usually remain disparate throughout the process, but some algorithms combine them after some iterations to combine the search space explored individually by each sub-population [94].

For gradually changing targets, which is the focus of this chapter, the techniques for maintaining diversity are more relevant. Diverse individuals keep the search space broad and prevents the algorithm from prematurely converging. This allows a wider exploration and lets the algorithm move its focus in the search area with the moving optimum. The benefits of diversity in evolutionary algorithms has been surveyed and analyzed in [95]. A classification of diversity maintaining, controlling and learning mechanisms is discussed extensively in [96].

Hyper-mutation [97] and random immigrants [98] are two well-known techniques and are widely used for introducing diversity in the evolutionary algorithms. Hyper-mutation increases mutation rate for a period of time when a change is detected and random immigrants introduces randomly generated individuals into the population with each generation. Variable local search [99] is similar to hyper-mutation, it increases mutation strength upon detecting a change, instead of changing the mutation rate. Fitness sharing [100] penalizes similar individuals to encourage diversity in the population. In dynamic problems where high diversity is critical, the problem is converted to a multi-objective optimization problem with diversity

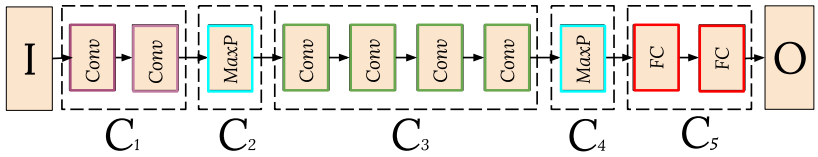


Figure 4.2: Plain CNN architecture where similar consecutive layers are grouped into clusters. Conv is convolutional, MaxP is max-pool and FC is fully-connected layer respectively.

as an extra objective to be maintained throughout the optimization process [101].

Our approach is closer to hyper-mutation and variable local search approaches, where a disruptive genetic operator is used to introduce diversity in the population. However, that is where the similarity ends. There is little need to detect the changes in gradually moving functions, moreover high population diversity is not a requisite near the saturation points. Our work differs from most diversity maintenance techniques in the way diversity level is explicitly guided in an adaptive manner based on the rate of change of the moving optimum.

4.3 PROBLEM DEFINITION

In this section, the Evolutionary Piecemeal Training algorithm is formally defined as an optimization problem. We train a population of neural networks, and the objective is to find a good topology, such that the accuracy of the neural network is maximal for the given dataset, upon completion of the training.

4.3.1 *Neural Network*

Formally, a neural network $\widetilde{nn}$, as presented in previous chapters, consists of its architecture T and weights $\omega \in \mathbb{R}$.

$$\widetilde{nn} = \{T, \omega\} \tag{1}$$

Here, T is a sequence of layers, which can further be grouped into clusters of consecutively similar layers. Figure 4.2 illustrates the concept of cluster formation for a simple CNN. All subsequent layers that are of the same type are grouped in the same cluster, for instance, the first two convolutional layers are in the cluster C_1 and cluster C_5 contains only fully-connected

layers. A general cluster based architecture T with l clusters, and with I and O as input and output layers respectively, can now be defined as:

$$T = \{I, C_1, C_2 ... C_l, O\}, \tag{2}$$

A cluster C_k of type C_k^{type} consisting of n layers can be represented by:

$$C_k = \{L_{k1}, L_{k2} ... L_{kn}\}: \ \beta_k^{min} \leqslant n \leqslant \beta_k^{max}$$

$$\text{where, } L_{ki} = \{L | \, L \in [C_k^{type}, \eta_{ki}, p_{ki}]\}: \ \eta_k^{low} \leqslant \eta_{ki} \leqslant \eta_k^{up}$$

$$\ni \{\beta_k^{min}, \beta_k^{max}, \eta_k^{low}, \eta_k^{up} \in \mathbb{N}^+\} \text{ and } p_{ki} \in \pi_k \tag{3}$$

Where $(\beta_k^{min}, \beta_k^{max})$ are the bounds on the number of layers possible in the cluster, and the number of neurons in a layer are bounded by $(\eta_k^{low}, \eta_k^{up})$. Moreover, the hyper-parameters that are dependant on the layer type, p_{ki}, such as kernel size and stride, are defined by π_k in the cluster. These constraints are specific to each cluster and are independent from bounds of the other clusters. Every layer in a cluster is of the same type (e.g., convolution, pooling, fully connected), and its hyper-parameters conform to the constraints placed by its parent cluster.

4.3.2 *Population based training*

While a neural network is being trained, its weights are constantly changing, and in that sense both the weights and the neural network may be considered as functions of time (i.e., iterations): $\omega(t)$ and $\widetilde{nn}(t)$. $\widetilde{nn}(0)$ is then the initial neural network at the beginning of its training with randomly initialized weights $\omega(0)$. The architecture of this neural network remains unchanged during the training. Hence,

$$\widetilde{nn}(t) = \{T, \omega(t)\} \tag{4}$$

A neural network $\widetilde{nn} \in \widetilde{NN}$, where $\widetilde{NN}$ is the set of all possible neural networks with an architecture $T \in \widetilde{T}$, where $\widetilde{T}$ is the set of all architectures defined in the search space along with its constraints. The population of neural networks may also be seen as a function of time. The population

$\widetilde{P}(t)$ of CNNs at any given time can then be defined as a set of neural networks at that point,

$$\widetilde{P}(t) = \{\widetilde{nn_1}(t), \widetilde{nn_2}(t), \widetilde{nn_3}(t).....\widetilde{nn_s}(t)\}: \ s \in \mathbb{N}^+ \tag{5}$$

The population size, s, is constant throughout the duration of the algorithm. If a neural network is dropped from the population because it is not performing as well as the other models, then it has to be replaced by another neural network. Moreover, the population size is required to be large enough so that enough diversity is maintained among the CNN models in the population.

The algorithm runs for τ_{max} iterations, the value of which is dependent on the nature and complexity of the task. τ_{max} can be defined at the beginning of the search or can be updated during the iterations based on the rate of change of the evaluation metric, such as prediction accuracy. We use validation accuracy on test data, $Acc(\widetilde{nn})$, as the main performance metric of a neural network, which is dependent on topology as well as its training time or the number of iterations of the evolutionary algorithm. Accuracy during training is then defined as:

$$\forall\, 1 \leqslant t \leqslant \tau_{max},\ Acc(\widetilde{nn}(t)) = Acc(\widetilde{nn}(t-1)) + \frac{\partial(Acc(\widetilde{nn}))}{\partial t} \tag{6}$$

In GSOF, $\frac{\partial(Acc(\widetilde{nn}))}{\partial t}$ is small and in the saturation phase it is almost zero. Additionally, at any time t, the population is defined as,

$$\forall\, 1 \leqslant t \leqslant \tau_{max},\ \widetilde{P}(t) = f_{ept}(\widetilde{P}(t-1)) \tag{7}$$

The function $f_{ept}()$ is applied to the population during every iteration. In absence of the evolutionary architecture modifications, $f_{ept}()$ consists of only the training function $f_{train}()$, which trains a CNN with a small subset of training data (i.e., a piece-meal training step). It is possible to train all neural networks in an iteration, in any order of sequential and parallel executions, to best utilize the computational resources available.

4.3.3 *Selection and Replacement*

To summarize, if $f_{evo}()$ represents the evolutionary operator (both mutation and recombination) function and $f_{select}()$ represents the population

selection function then, $f_{ept}()$ from Equation (7), for population $\widetilde{P}(t)$ at any iteration t, can be defined as

$$f_{ept}(\widetilde{P}(t)) = f_{train} \circ f_{select} \circ f_{evo}(\widetilde{P}(t)) \tag{8}$$

The function $f_{ept}()$, as composition of the other mentioned functions, defines the transition function of the population from one iteration to the next. $f_{train}()$ trains every CNN in the population, $f_{select}()$ evaluates the population and selects (or rejects) suitable candidates, some of which then go through evolutionary operators in $f_{evo}()$.

As the iterations continue, the population keeps gradually changing, due to architecture alterations, along with appropriate selection and replacement of models.

4.3.4 *Optimization Objective*

The main objective of our algorithm is to find a neural network with maximum accuracy possible. As $Acc(\widetilde{nn}(T, \omega(t))$ represents the accuracy of a neural network $\widetilde{nn}$. Given $\widetilde{T}$ as the set of all possible architectures and $\widetilde{NN}$ as the set of all possible neural networks, the objective is to find neural network $\widetilde{nn}'$ ($\widetilde{nn}' \in \widetilde{NN}$) with architecture ($T' \in \widetilde{T}$), such that

$$max_{\widetilde{nn}' \in \widetilde{NN}}\ Acc(\widetilde{nn}'(T', \omega(t)) \tag{9}$$

The optimization objective helps to formulate the $f_{select}()$ function to which the whole population is subject to after every iteration. To maximise accuracy, $f_{select}()$ chooses the best performing CNNs, able to reach higher accuracy in the population.

4.4 POPULATION DIVERSITY FOR DYNAMIC OPTIMIZATION

To reduce computational complexity, we have a fixed sized genotype representation of the topology by fixing the number of clusters for every topology. For different problems with each a different dataset, the number of clusters and constrains in a cluster may vary. Even though the number of clusters is fixed, each cluster can have a variable number of layers, resulting in a different total number of layers in every randomly created topology. In a population based evolutionary methodology, there are many diversity maintenance techniques as discussed in Section 4.2.

In this work, diversity refers to the distance between individual neural network topologies. We do not measure the distance between individuals explicitly, instead we define two coarse-grained levels of diverseness. That is, two individuals are *dissimilar* if the total number of layers or layer types are different from each other. Two individuals are *similar* to each other when the total number of combined layers as well as type of each layer is same for both. Individual layer parameters (π_l) may be different for every layer. The layer parameters play a big role in making a neural network more efficient than others even when having exactly the same layer types. Two individuals with *similar* diversity level does not imply they have similar prediction accuracy. We explore these layer parameters during the algorithm through the mutation operator, but mutation does not influence the diversity as the number of layers remains unchanged.

4.4.1 *Genetic Operators*

By defining two levels of diverseness, we can differentiate the behavior of mutation and recombination operators w.r.t. the diversity level it introduces in the population. We implement a disruptive genetic recombination operator to introduce more diversity by creating children with a different number of total layers, whereas mutations operate on a layer's parameters only, therefore not contributing to a change in diversity levels of the population. Both operators are described below.

Mutation

Our mutate operator randomly selects a layer from the neural network topology and changes only one the layer parameters (π_l) by a small value. Change in the number of neurons of selected layer is constrained by ρ_m%. The number of layers in the offspring remains the same creating a *similar* individual in terms of diversity. The mutate operations are designed to be function preserving taken from [65, 66], which means that the disturbance on the training process and on the current performance of the child topology is minimal. As the training continues, coefficients values of the child topology change and these little changes may contribute to a better performing topology towards the convergence of the optimization algorithm.

Recombination

The recombination operates on two individuals, randomly selects a cluster position and swaps the whole cluster between both the topologies. Figure 4.3 exemplifies a swap operator with a topology having four clusters.

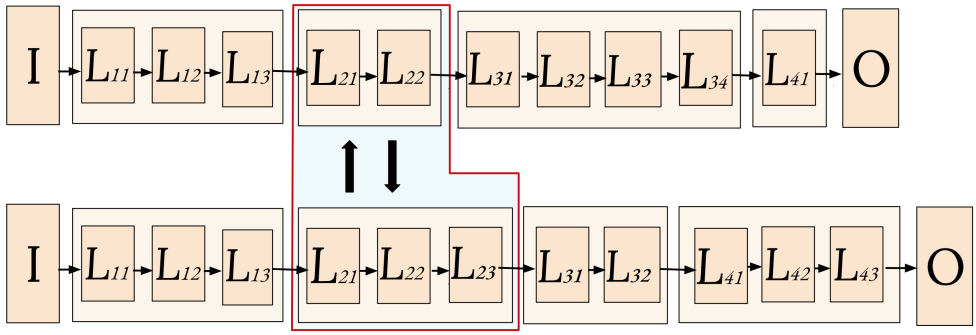


Figure 4.3: Recombination operator applied to two neural networks, where the cluster C_2 gets swapped. The cluster has a different number of layers in each model.

The reason for this being a disruptive operator comes from the fact that even though the layers being swapped are roughly at the same position in the layer chain, the number of layers present in each cluster are different. One cluster of two convolutional layers might get swapped with another cluster containing five convolutional layers, thus creating diverse *dissimilar* offsprings. The clusters at those same positions are designed to keep the same input-output feature map sizes, so the recombination does not result in a corrupt neural network. There is some disturbance caused to the training process by the recombination operator, but as training continues, the loss incurred is observed to have recovered in after a few iterations.

4.4.2 *Adaptive diversity*

The desired level of population diversity for the optimization varies depending on the rate of change in the GSOF at any given time. High diversity is advantageous when the rate of change of the objective function is high and vice versa. In the proposed methodology, we adaptively influence the diversity via recombination probability modification throughout the iterative process. Recombination probability is an individual's selection probability to undergo a recombination operation. When the shape of the objective function is known apriori, it is possible to setup an offline adaptive control function with expected rate of change to guide it. In absence of this prior knowledge, the average rate of change of the objective function over a short interval gives a good indication of diversity needed at any given time point. We call this an online adaptive diversity control function.

Offline Adaptive

For an offline adaptive recombination probability function, we select an exponential decay function which loosely represents the inverted accuracy function during training. Where α is the decay factor, recombination probability, P_r as a function of time is defined as:

$$P_r(t) = P_r(0) * \alpha^t, : 0 < \alpha < 1 \tag{10}$$

Online Adaptive

For the online adaptive recombination probability function, the change in objective function is monitored and recombination probability is modified based on its current rate of change. This generic function can be applied to any GSOF based optimization. With γ as the scale factor, recombination probability is modified by the following:

$$P_r(t) = P_r(0) * \gamma * \frac{\partial(Acc(\widetilde{nn}))}{\partial t} \tag{11}$$

4.4.3 *Algorithm*

The EPT algorithm (Algorithm 3) is thereby slightly modified, with addition of the function `updateCrossoverProbability()` to manage the population diversity. This function is called during each iteration to modify the recombination probability rate depending on the chosen approach, i.e. according to one of the equation: Equation (10) or Equation (11). The updated algorithm is presented in Algorithm 4.

4.5 EXPERIMENTAL STUDY

In this section, we evaluate the diversity based techniques for EPT algorithm using the PAMAP2 [24] dataset for human activity recognition and outline the setup of our experiments. As in the the previous chapter, the Java based Jenetics library [69] is used for evolutionary computation and the Python based Caffe2 [33] library for training and testing. The ONNX [32] format is utilized to represent and transfer the neural networks across different modules. These experiments are executed on one GPU (GeForce RTX 2080) to train the neural networks, however the algorithm is scalable and is able to use multiple GPUs in parallel during each iteration.

Algorithm 4: Evolutionary Piecemeal Training

Evolutionary Inputs: N_g, N_p, P_r, P_m, α
Training Inputs : τ_{params}, δ

$\wp_o \leftarrow \text{InitializePopulation}(N_p)$
for $i \leftarrow o$ .... N_g **do**
  $\wp_i \leftarrow \text{PiecemealTrain}(\wp_i, \tau_{params}, \delta)$
  $E_v \leftarrow \text{EvaluateAccuracy}(\wp_i)$
  $\wp_{best} \leftarrow \text{BestSelection}(\alpha, \wp_i, E_v)$
  $\wp_r \leftarrow \text{random}((1-\alpha) * \wp_i)$
  update $\wp_i \leftarrow \wp_{best} + \wp_r$
  $P'_r \leftarrow \text{updateRecombinationProbability}(P_r, i, E_v)$
  $\wp_{rc} \leftarrow \text{RecombinePopulation}(\wp_i, P'_r)$
  $\wp_{mu} \leftarrow \text{MutatePopulation}(\wp_i, P_m)$
  $\wp_{remaining} \leftarrow \text{UnchangedPopulation}(\wp_i)$
  update $\wp_i \leftarrow \wp_{mu} + \wp_{rc} + \wp_{remaining}$
end
$E_v \leftarrow \text{EvaluateAccuracy}(\wp_{N_g})$
return *BestCandidates*(E_v)

4.5.1 *Setup*

The topology structure for PAMAP2 is the same as the search space as described in previous chapter. Table 3.2 (in Chapter 3) indicates each cluster's details and various constraints. Every layer is followed by a ReLu activation layer and number of neurons are modified in steps of 8 during the mutation operation.

The parameters of the algorithm for all variants were kept the same and are summarized in Table 4.1. These parameters were determined during preliminary experiments. Training of all neural networks was done using the Adam optimizer [70], with a learning rate of $1e^{-4}$ and batch size 50.

4.5.2 *Results*

Figure 4.4 illustrates how the population of neural networks is evolving while training during one run each of offline adaptive, online adaptive and standard evolutionary algorithms. Each algorithm takes approximately 10 hours to complete with the majority of time spent in training a neural network. The population size is fixed during the genetic iterations, so the

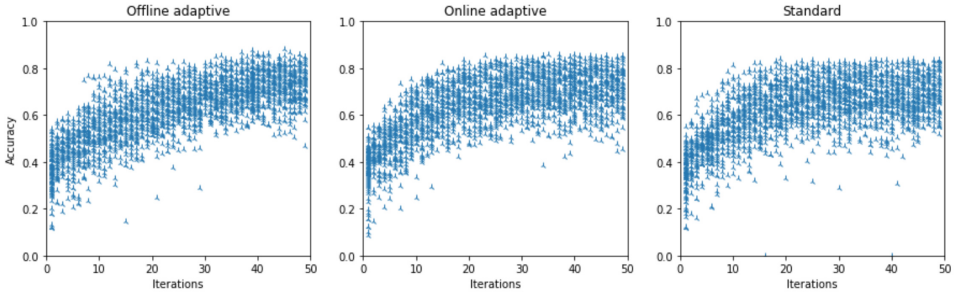


Figure 4.4: Accuracy values for all of the neural networks evolving during an experiment each for offline adaptive, online adaptive and standard evolutionary algorithms.

total number of training operations are the same for all the algorithms. We consider the standard evolutionary algorithm to have static mutation and recombination probabilities. Experimental results of average accuracy and best accuracy after completion of the optimization algorithm are presented in Table 4.2. All presented numbers are averages of 10 independent runs. Figure 4.6 shows the performance of offline adaptive, online adaptive and the standard evolutionary algorithm with best and average accuracy found during each iteration. The information regarding standard deviation is omitted from the average accuracy graph, in order to have a graph which is easy to read.

Table 4.1: Algorithm parameter values in experiments

Parameter		**Value**
Mutation change rate	ρ_m	0.12
Mutation selection probability	P_m	0.3
Initial Recombination selection probability	$P_r(0)$	0.4
Adaptive offline decay factor	α	0.95
Adaptive online scale factor	γ	60
Population size	N_p	50
No of iterations	N_g	50
Population replacement rate	Ω	0.03
Training interval size	δ_k	20,000

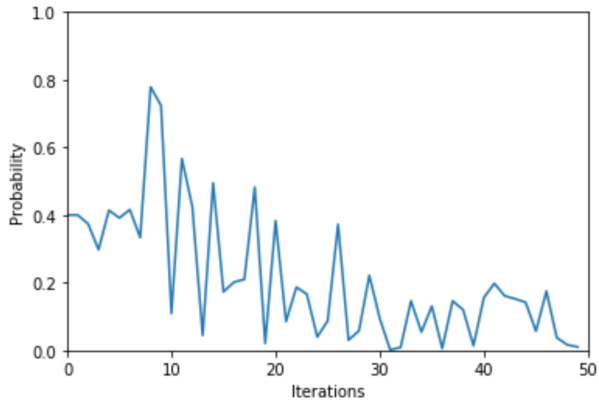


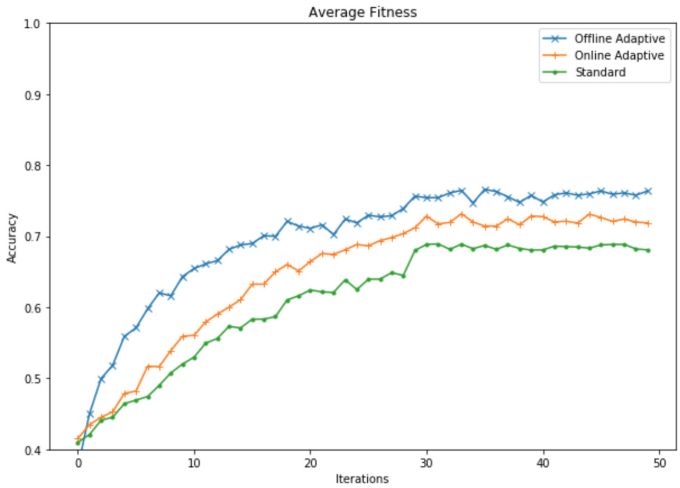
Figure 4.5: Recombination probability during iterations of the online adaptive evolutionary algorithm.

It is clear from the results that both of the adaptive diversity control evolutionary algorithms outperform the standard evolutionary algorithm. Among the adaptive varieties, the offline version performs slightly better than the online version. It is to be expected as the rate of change of accuracy function can have small *blips* because of the stochastic nature of the training process, which leads to lower or higher recombination probabilities for short intervals. Figure 4.5 shows that the graph of recombination probability with respect to the algorithm iterations is not as smooth as the exponential decay function of the offline adaptive algorithm. However, the trend is similar and we see decreasing values of recombination probability over the iterations.

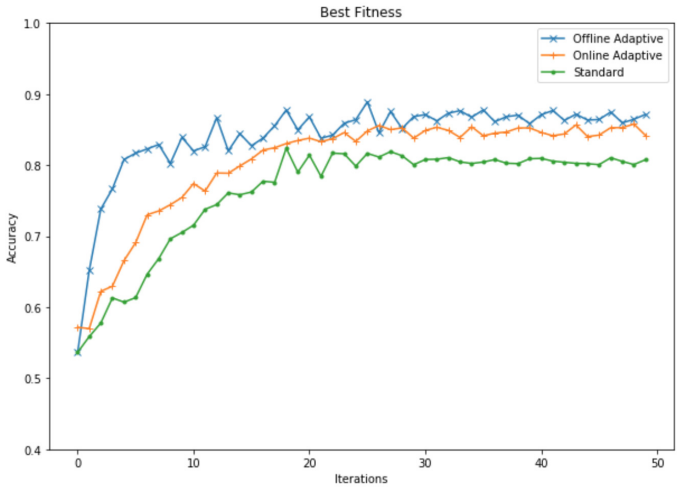
We see better results in the offline adaptive version, where the recombination probability curve is smooth and diversity is tightly controlled over

Table 4.2: Experimental results of the mean average accuracy and best accuracy in the final iteration

Algorithm	**Average Accuracy**	**Best Accuracy**
Offline adaptive	0.739 ($\pm$0.012)	0.859 ($\pm$0.010)
Online adaptive	0.718 ($\pm$0.009)	0.842 ($\pm$0.016)
Standard	0.688 ($\pm$0.015)	0.809 ($\pm$0.012)



(a) Average accuracy of population



(b) Best Accuracy

Figure 4.6: Training curves. Average accuracy refers to the average performance of whole population at the given optimization iteration. Best accuracy refers to best found performance of an individual model in the population.

iterations. Results illustrate that whenever the GSOF is known, it is preferable to design the diversity control function based on this knowledge. However, an evolutionary algorithm with an online adaptive diversity function still performs better than the standard evolutionary algorithm with fixed recombination probability suggesting that in the saturation phase, having less diversity is better to explore the local search space.

The best neural networks found through these algorithms can be further processed, modified or trained as needed outside this algorithm. It is important to note that the results presented in previous chapter were based on the offline adaptive version of the EPT algorithm. The best ones found for both PAMAP2 and CIFAR-10 were modified to add Batch Normalization layers after every convolutional layer and trained further to achieve higher accuracy (See Table 3.3 and Table 3.4). The work presented in this chapter is not an improvement on the EPT algorithm. It is rather a discussion on the challenges faced during the design of the original algorithm and put forth the solutions that worked in making the algorithm achieve efficient neural architectures.

4.6 SUMMARY

This chapter provided insights into characteristics of the EPT algorithm, including the challenges faced due to the dynamic nature of the objective function. We explored an adaptive diversity control based methodology for evolutionary dynamic optimization of GSOFs. By defining coarse grained diversity levels and designing genetic operators specific to each level, we could influence and control the population diversity meaningfully. The diversity control approach for EPT was validated by performing experiments on the PAMAP2 dataset. In the next chapter, this algorithm is extended to include multiple objectives for the search, specifically keeping edge devices and their resource limitations at the forefront.

5

MULTI-OBJECTIVE EVOLUTIONARY PIECEMEAL TRAINING

As claimed in the previous chapters, the EPT algorithm can be further extended to include multiple objectives as well as incorporate a search space with complex neural architectures. To prove the efficacy of the extension to the original methodology, we first consider the reduction of the number of parameters of the neural network as an additional search goal, for the PAMAP2 dataset. In the second set of experiments, the algorithm is extended to include hardware specific objectives, namely the throughput, the memory and the energy consumption on a target hardware. Additionally, the design space is augmented with the possibility to include skip connections, allowing the architecture search for ResNet style CNNs. The algorithm converges with a Pareto Front, which is a set of CNNs with pareto optimality w.r.t. all the given objectives. In a pareto optimal set, none of the objectives can be further improved without worsening some of the other objectives, thereby producing neural architectures with different trade-offs for the objectives in consideration.

This Chapter is based on:

- **D. Sapra** and A. D. Pimentel *"Designing convolutional neural networks with constrained evolutionary piecemeal training"* [40], in *Applied Intelligence*, © Springer.

and,

- S. Minakova, **D. Sapra**, T. Stefanov and A. D. Pimentel *"Scenario Based Run-time Switching for Adaptive CNN-based Applications at the Edge"* [102], in *ACM Transactions on Embedded Computing Systems*, © ACM.

5.1 INTRODUCTION

At the very beginning of our research on efficient NAS, the ability of the algorithm to extend to multi-objective search was taken into consideration. Evolutionary algorithms are well studied in the multiple-objective search domain and can be very flexible in the management of adding (or removing) objectives as required, for each execution of the algorithm. In this chapter, the original EPT methodology, with only accuracy as an optimization objective, is extended to consider multiple objectives for the search.

In the initial experiment, we consider the reduction of the number of parameters of the neural network as an additional search goal. The accuracy maximization and parameter minimization can be conflicting objectives for an efficient neural network. Smaller CNNs tend to have lower accuracy and high accuracy is generally obtained by larger neural networks. However, too many parameters tend to cause over-fitting, which may lead to poor generalization, and therefore, highlight the importance of a constrained search process not only for hardware resource usage, but also to avoid over-fitting. We apply the proposed extension to the PAMAP2 dataset and perform the multi-objective search for efficient architectures.

Nevertheless, the deployment of neural networks in a wide range of settings requires more than parameter reduction for an efficient application execution in the long term. For example, by optimizing only the number of mathematical operations through parameter reduction, it can not be guaranteed that the application meets the power constraint of a battery-operated device. With the advent of connected IoT (Internet of Things) networks [103], we see an urgent demand for intelligent applications that run directly on the edge devices, which typically do not have large computation or storage capabilities. The execution of a CNN based application on the edge can be extremely challenging, due to high demands placed by the application while limited by the capabilities of the underlying hardware of an edge device.

Keeping these factors in mind, the second set of experiment was designed to focus on the objectives related to the capabilities of the target hardware, in addition to the accuracy of the neural network. This experiment is then considered to be a hardware-aware EPT algorithm, where the hardware metrics are incorporated into the search algorithm as optimization objectives. There can be a vast variety of requirements that an embedded system needs to fulfill, a subset of which is taken into consideration for this ex-

periment. The most common of the demands on a CNN-based application running on an edge device are:

1. **high accuracy**. The CNN should be able to efficiently perform its intended task;
2. **high throughput**. The applications running on an edge device might require a (near) real-time response;
3. **low memory cost**. An edge device typically has limited memory available;
4. **low energy cost**. When an edge device is battery-powered, such as a drone, it is important to have a low energy footprint.

Together we call them ATME characteristic (Accuracy, Throughput, Memory, Energy), and in the hardware-aware EPT algorithm, we consider each of them as an optimization objective. The accuracy, typically measured in percent, characterizes the fraction of correct predictions generated by a CNN from the total number of predictions generated by the CNN. The throughput, typically measured in frames per second (fps), characterizes the speed with which the CNN is able to process input data and produce output data. The memory cost, typically measured in Megabytes (MB), specifies the total amount of memory required to execute a CNN. The energy cost, measured in Joules, specifies the amount of energy consumed by a CNN to process one input frame.

It is conspicuous that the ATME characteristics are not representative for the CNN in isolation, but the combination of neural architecture and its execution on one specific hardware. The accuracy is solely dependent on the learned coefficients during the training process, so it is never affected by the hardware it runs on. On the other hand, the TME characteristics are typically dependant on the capabilities of the target device. The results of the hardware-aware EPT algorithm is presented for three datasets, namely VOC, PAMAP2 and CIFAR-10, for the NVIDIA Jetson TX2 embedded platform [104].

With a multiple objective based search, selection of the best candidates is concluded through *Pareto optimization*, where any objective cannot be improved without worsening some of the other objectives. The set of candidates selected in such a fashion are collectively called a *Pareto Front*.

The Pareto Front obtained upon convergence sets forth the various possible CNNs to deploy on the wearable embedded device. It allows the designer to be aware of the architecture choices available in terms of which

neural architecture provides a trade-off between the size of the model versus the accuracy. One of these CNNs can be strategically deployed depending upon the desired functional goals and available resources on the device.

By presenting these two disparate multiple-objective experiments, we also demonstrate the flexibility of the EPT algorithm. The algorithm makes it feasible to have an adjustable number of objectives to design a CNN based application that can be efficiently deployed on an edge device. Moreover, the search space for the CIFAR-10 and the VOC datasets in the second set of experiments includes residual connections to allow a search space consisting of ResNet [5] style neural networks. This further illustrates the flexibility of the EPT algorithm in its ability to work with different styles of neural architectures.

5.2 RELATED WORK

Evolutionary-based multi-objective NAS algorithms have been widely explored for multi-objective search as their ability to easily incorporate an extra objective is well known [105]. LEMONADE [59] is an evolutionary based multi-objective algorithm, which utilizes the Lamarckian inheritance mechanism. NSGA-Net [55] utilizes the Non-dominated Sorting Genetic Algorithm II (NSGA-II) to construct the Pareto Front with the aim of learning the trade-off between model classification error with its computational complexity. Our work also utilizes an evolutionary algorithm for architecture modification, where the key difference is in the way we look at NAS from a different perspective, by exploring the possibility of finding optimal architectures during the training process itself as opposed to accuracy prediction or training as a separate performance estimation strategy.

Other approaches for multi-objective search have also been researched in recent times, for example, Reinforcement Learning (RL) based approaches incorporate the objectives directly into the reward function of an agent. The agent is continuously rewarded for finding better architectures, and thus the search is slowly steered towards neural networks with higher performance. MONAS [106] and MnasNet [107] are examples of Reinforcement Learning based approaches for multi-objective NAS. While MONAS uses validation accuracy and energy expenditure on the target model to create a Pareto Front, MnasNet explicitly incorporates the latency information in the main search objective to discover models with a good trade-off across accuracy and latency. Any RL approach resolutely demands a suitable agent, which frequently happens to be a complicated model or perhaps another

neural network. The construction of the agent and its optimization involve substantial effort towards designing and subsequent fine tuning.

In order to deploy the one-shot architecture search in a multi-objective scenario, the objectives are incorporated into the loss function of the training process. For example, FBNet [10] and DenseNAS [11] are multi-objective one-shot differentiable NAS frameworks. In the former method, the loss function is the weighted product of cross-entropy loss, incurred during training, with the latency of the target device. Whereas in the latter work, the loss function is the weighted aggregation of cross-entropy loss along with the latency-based regularization. However, it is not possible to add all objectives to the search in this manner, thereby making them inflexible to embrace an additional objective on the go.

In addition to the NAS methodologies, there are various memory reduction techniques, for instance pruning [108], quantization [109] and compression [110–112], which can be deployed to either minimizing the number of FLoating Point Operations (FLOPs), reduce the bit-width representation of parameters or use compression algorithms. Though these can effectively reduce memory cost and thereby energy consumption, they may lead to loss in the CNN accuracy. Moreover, in some cases, these techniques may result in worse performance, specially if the hardware is not optimized for them. For example, a pruning approach may lead to sparse representation for the coefficients [113], and the hardware needs to handle the sparsity appropriately to take advantage of the smaller memory footprint and reduced computation offered through pruning.

Therefore, multi-objective NAS algorithms provide a better mechanism to find a suitable neural architecture by not only searching for suitable CNNs for a target environment, but also providing insight into the trade-off between different objectives. However, the multi-objective NAS can be modified in a manner that optimizes for these memory-reduction techniques within a NAS algorithm [114]. For instance, APQ [115] proposes a multi-step NAS methodology, where an accuracy predictor for quantized CNNs is designed first. This accuracy predictor is further utilized in the evaluation of neural networks, during an evolutionary hardware-aware NAS for CNNs with various quantization policies.

5.3 METHODOLOGY

In this section, we first explain the approach used to evaluate all the objectives for a multi-objective NAS. Further, the modifications to the EPT

Algorithm are outlined, which allow the algorithm to be extended in order to conduct a multi-objective search.

5.3.1 *Evaluation of the Objectives*

Accuracy

The most popularly used metric, classification accuracy, is computed as the number of correctly processed input frames to the total number of the CNN input frames. It is important to note that even though we refer to evaluation of a CNN as accuracy, it is possible to use any other evaluation metric suitable to the application, as discussed in Chapter 2. For instance, F-1 score, precision, recall, PR-AUC (Area under curve for precision recall) are some of the metrics used for CNNs for imbalanced datasets.

Number of Parameters

The number of parameters of a CNN is the sum of parameters per layer, which is computed as:

$$P = \sum_{l_i \in L} |par_i| \tag{12}$$

Where $|par_i|$ is the total number of the learnable parameters of layer l_i and L is the total number of layers of the CNN.

There are various ways to analyse the hardware metrics, through mathematical analysis, simulation and direct measurement on the embedded device. Any of the suitable hardware models can be used in the evaluation phase of the EPT algorithm. We have used a model developed by the LERC group at Leiden University, where a mathematical analysis model combined with a few direct measurements on the target hardware was utilized [102]. Here we briefly describe the model which evaluates the memory consumption, throughput and energy for the neural architecture executing on the resource constrained hardware.

Memory

The CNN memory cost M is computed as:

$$M = \sum_{l_i \in L} (|par_i| * size_{p \in par} + \sum_{e_{ij} \in O_i} |Y_i| * size_{y \in Y_i}) \tag{13}$$

Where $|par_i|$ is the total number of the learnable parameters of layer l_i; $size_{p \in par}$ is the amount of memory in MB, occupied by one learnable parameter; Y_i is the data tensor, produced by layer l_i for its every output edge $e_{ij} \in O_i$; $size_{y \in Y_i}$ is the amount of memory in MB, occupied by one element of data in Y_i.

Throughput and Energy

The CNN throughput T is computed as:

$$T = N / \sum_{l_i \in L} t_i \tag{14}$$

where N is the CNN batch size, i.e., the number of frames, processed by every CNN layer l_i [116]; $\sum_{l_i \in L} t_i$ is the time in seconds, required to perform execution of the CNN *CNN(L, E)*, represented as a sequence of |L| computational steps, where at every step a CNN layer $l_i \in L$ is executed; t_i is the time required to execute layer $l_i \in L$. Analogously, the CNN energy cost ξ is computed as:

$$\xi = \sum_{l_i \in L} \xi_i / N \tag{15}$$

where ξ_i is the energy cost (in Joules) associated with the execution of CNN layer l_i. The execution time t_i and energy cost ξ_i, associated with CNN layer l_i, utilized in Equation (14) and Equation (15), are notoriously hard to evaluate analytically [117]. Therefore, in this methodology, t_i and ξ_i are obtained by performing measurements on the target edge device for every different type of layer and for some of the basic layer configurations.

5.3.2 *Workflow and Algorithm Extension*

The original EPT methodology is extended in order to make it suitable for multiple objectives besides the accuracy. Figure 5.1 highlights the changes done to the primary workflow, for the multi-objective search. Initialization and piecemeal-training steps remain the same, however, additional evaluations are performed to deduce other objectives. In the first set of experiments, the parameter count of the CNN model becomes the second evaluation criterion, along with the accuracy of the model. In hardware-aware EPT with four ATME objectives, all of them are evaluated at this step.

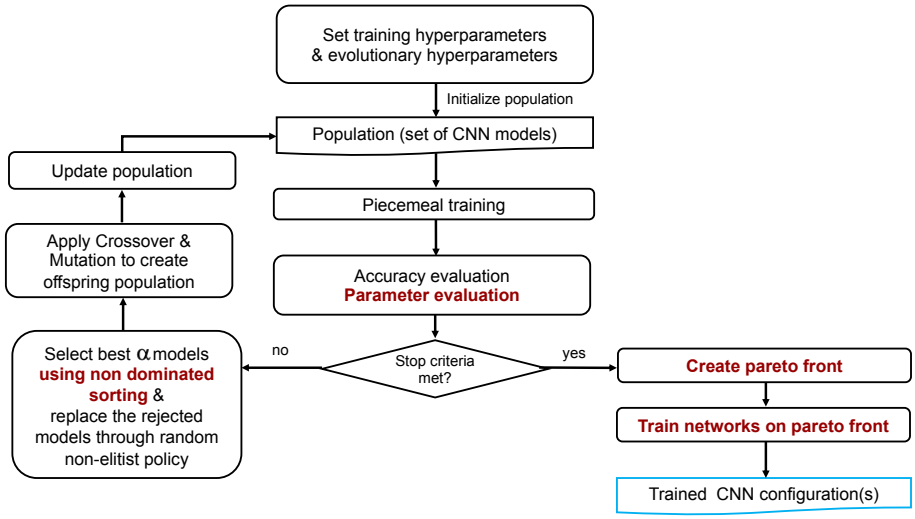


Figure 5.1: Workflow for the Evolutionary Piecemeal Training, extended version to include multiple objectives.

The selection policy is updated to be based on non-dominated sorting [118], which takes into account all the objectives to sort the models in the order of their relative quality of performance. Replacement policy, evolutionary operator application and population update steps are unchanged in the extended algorithm.

After the iterations are complete, a Pareto optimal set is selected from the population based on all evaluated objectives. The CNNs in this Pareto optimal set, also called as Pareto Front, are selected to be eventually completely trained. In a Pareto Front, an objective cannot be further improved until one or more of the other objectives are made worse. Therefore, all the models in the Pareto set are considered to be equally adequate to be marked as the best model. In this scenario, the final selection lies in the hands of system designer, and may also be based on higher priority placed on one of the objectives.

We outline the modified and extended algorithm for multi-objective search in Algorithm 5. The changes to the original Algorithm 3 reflect the same changes that were highlighted in the multi-objective workflow in Figure 5.1. `EvaluateParameters()` evaluates all the evaluation metrics other than accuracy. `NSGA2Selection()` replaces the previous selection algorithm with α still kept at a very high value. This function is based on the popular multi-objective selection algorithm, NSGA-II [118], which takes all objectives into

Algorithm 5: Multi-Objective Evolutionary Piecemeal Training

Evolutionary Inputs: N_g, N_p, P_r, P_m, α
Training Inputs : τ_{params}, δ

$\wp_o \leftarrow InitializePopulation(N_p)$
for $i \leftarrow o$ .... N_g **do**
  $\wp_i \leftarrow PiecemealTrain(\wp_i, \tau_{params}, \delta)$
  $Ev_{acc} \leftarrow EvaluateAccuracy(\wp_i)$
  $Ev_{params} \leftarrow EvaluateParameters(\wp_i)$
  $\wp_{best} \leftarrow NSGA2Selection(\alpha, \wp_i, Ev_{acc}, Ev_{params})$
  $\wp_r \leftarrow random((1-\alpha) * \wp_i)$
  update $\wp_i \leftarrow \wp_{best} + \wp_r$
  $\wp_{rc} \leftarrow Recombine(\wp_i, P_r)$
  $\wp_{mu} \leftarrow Mutate(\wp_i, P_m)$
  update $\wp_i \leftarrow \wp_{mu} + \wp_{rc} + \wp_{remaining}$
end
$Ev_{acc} \leftarrow EvaluateAccuracy(\wp_{N_g})$
$Ev_{params} \leftarrow EvaluateParameters(\wp_{N_g})$
return *ParetoFront*(Ev_{acc}, Ev_{params})

account when selecting the best individuals. This algorithm returns the *Pareto Front* from the population.

In the first set of experiments, parameter minimization becomes the second search objective, along with the accuracy maximization for the search. Let $Params(\widetilde{nn}(T, \omega(t))$ represent the number of parameters of a neural network $\widetilde{nn}$. Then, the objective can be formulated to find a neural network $\widetilde{nn}''$ with architecture ($T'' \in \widetilde{T}$), such that

$$min_{\widetilde{nn}'' \in \widetilde{NN}} \; Params(\widetilde{nn}''(T'', \omega(t)) \tag{16}$$

From Equation (9), the original objective of the single objective EPT is to find neural network $\widetilde{nn}'$ ($\widetilde{nn}' \in \widetilde{NN}$) with architecture ($T' \in \widetilde{T}$), such that

$$max_{\widetilde{nn}' \in \widetilde{NN}} \; Acc(\widetilde{nn}'(T', \omega(t))$$

When the optimization objectives are conflicting with each other, as is the case with accuracy maximization and parameter minimization, $f_{select}()$ can not be as simple as selecting the best candidates based on linear sorting on one specification. The selection function now has to consider a sorting

based on non-domination of any single objective and select the best candidates. The NSGA-II selection algorithm [118] ensures that both optimization objectives are catered to during the search.

Similarly, for hardware-aware EPT, there are four objectives for the search, namely, accuracy maximization, throughput maximization, memory minimization and energy consumption minimization. The NSGA-II selection algorithm [118] in this situation then is used for four objectives simultaneously.

5.4 EXPERIMENTS

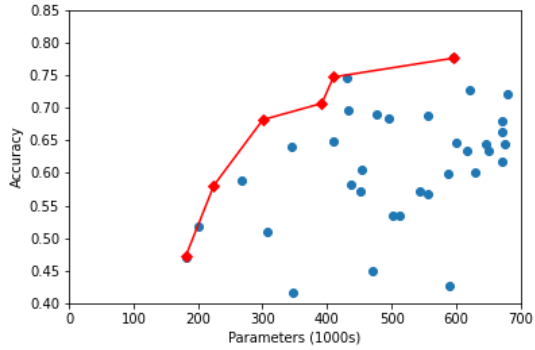
In this section we demonstrate the efficiency of multi-objective EPT through two independent experiments. As explained earlier, there are two sets of experiments. The first experiment takes accuracy and number of parameters into consideration, and the second experiment focuses on the target hardware metrics along with accuracy. The details of each of these experiments are provided below in this section.

5.4.1 *Two-objective EPT*

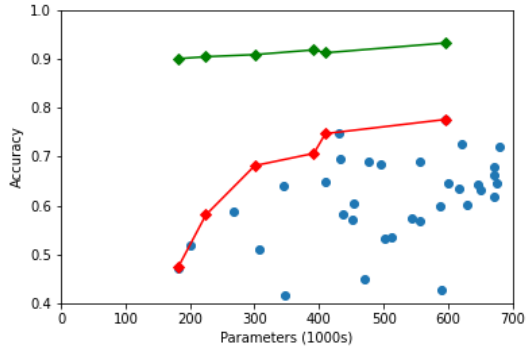
The experiment for the extended version of the algorithm with multiple objectives for the search has been performed on the PAMAP2 dataset. The definition of the architecture search space, and initialization of hyper-parameters for both evolutionary operators and training is exactly the same as the original algorithm.

The conflicting search objectives were to maximize accuracy while simultaneously to minimize the number of parameters of the CNN model. Once the algorithm has finished all the iterations, we plot the graph for accuracy versus the number of parameters for all models in the population, as shown in Figure 5.2 (a). The Pareto Front as selected by the algorithm is marked in red. The candidates that lie on the Pareto Front exhibit the trade-off between two objectives and one cannot be considered better over the other w.r.t both the objectives.

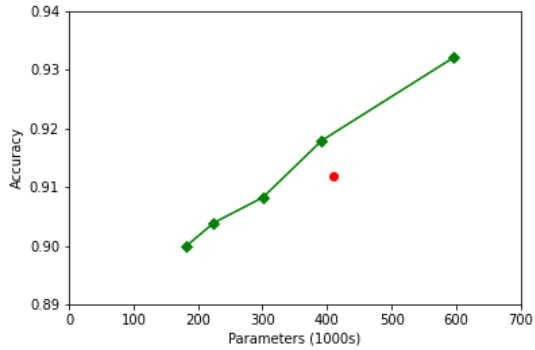
Subsequently, all the candidates on the Pareto Front were processed and trained further, after the addition of the Batch Normalization layers. Figure 5.2 (b) now presents two Pareto Fronts, first one being the Pareto Front determined by the algorithm (in red), and the second curve (in green) depicts the associated "trained" Pareto Front. The latter curve plots the accu-



(a) Pareto Front from the search



(b) Pareto Front trained further



(c) Updated final Pareto Front

Figure 5.2: Pareto Fronts for accuracy vs parameters of CNNs. Figure 5.2 (a) shows the Pareto Front created during the search with EPT. The scattered points represent CNNs from the final population upon convergence. The candidates from Pareto Front are further trained and Figure 5.2 (b) shows the "trained" Pareto Front. The Pareto Front is finally updated (Figure 5.2 (c)) to remove points that do not fall on it anymore.

racy for the same neural networks on the former curve, but as completely trained models.

However, once the training is complete, it is highly probable that some of the models do not follow the rules of a Pareto optimal set anymore. Figure 5.2 (c) shows a closer look at the "trained" Pareto Front, where the point marked in red clearly does not belong to the Pareto Front any longer. Those points are then removed from consideration and eventually a final Pareto Front is deduced.

The Pareto Front that is finally achieved gives a good indication of the trade-off between size of a neural network versus the performance. In the Pareto Front for PAMAP2, there were 5 points, ranging from 89.99% accuracy to 93.34% accuracy with models consisting of $\approx$ 200k to $\approx$ 600k parameters.

5.4.2 *Hardware-aware EPT*

The results for the second set of experiments on the hardware-aware EPT algorithm is presented for three datasets, namely VOC, PAMAP2 and CIFAR-10, for NVIDIA Jetson TX2 embedded platform [104]. As detailed in Chapter 2, VOC and CIFAR-10 are image classification datasets and PAMAP2 is for human activity recognition. NVIDIA Jetson TX2 is a resource limited hardware platform with a GPU available to accelerate the execution of a CNN. The objectives for this search are fourfold: Accuracy and throughput maximization along with memory and energy consumption minimization.

To explore the design space in this experimental study, we first define the search space in the form of cluster constraints as for all the datasets. These search spaces are shown in Table 5.1 for the VOC dataset, Table 5.2 for the PAMAP2 dataset, and Table 5.3 for the CIFAR-10 dataset. In these tables, the Cluster Type in the first column lists the abbreviated layer types. Conv, MaxP, GlbAvgP, GlbMaxP, FC are abbreviations for convolution, max-pool, global average pool, global max pool and fully connected, respectively. Conv+Res is a special cluster where all layers are convolutional, but there is a residual connection [5] from the input edge to the cluster until the output edge. This residual connection is maintained (or repaired) as needed during the architecture modification through evolutionary operators. The Conv+Res cluster is designed based on the ResNet v1 [5] family of neural networks. Since the CNNs are automatically generated based on the provided constraints by the NAS, they are not identical to any popular ResNet variant, such as, ResNet-18 or ResNet-128. The rest of the columns define

Table 5.1: VOC Search Space

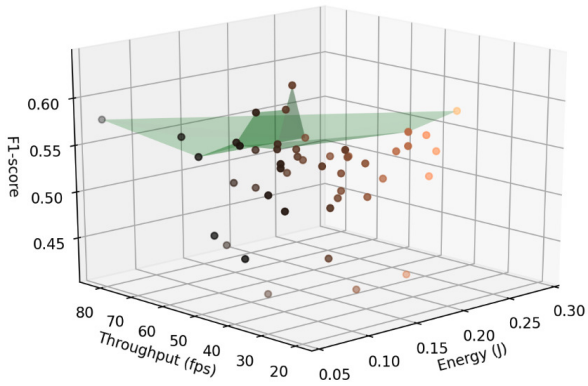
Cluster Type	Layers		Neurons		Kernel	
	β^{min}	β^{max}	η^{low}	η^{up}	κ^{min}	κ^{max}
C_1:Conv	1	3	16	96	3x3	7x7
C_2:MaxP	-	-	-	-	2x2	-
C_3:Conv+Res	1	5	16	96	3x3	7x7
C_4:MaxP	-	-	-	-	2x2	-
C_5:Conv+Res	1	5	32	128	3x3	7x7
C_6:MaxP	-	-	-	-	2x2	-
C_7:Conv+Res	1	5	32	128	3x3	7x7
C_8:MaxP	-	-	-	-	2x2	-
C_9:Conv+Res	1	5	64	256	3x3	7x7
C_{10}:MaxP	-	-	-	-	2x2	-
C_{11}:GlbAvgP	-	-	-	-	2x2	-

Table 5.2: PAMAP2 Search Space

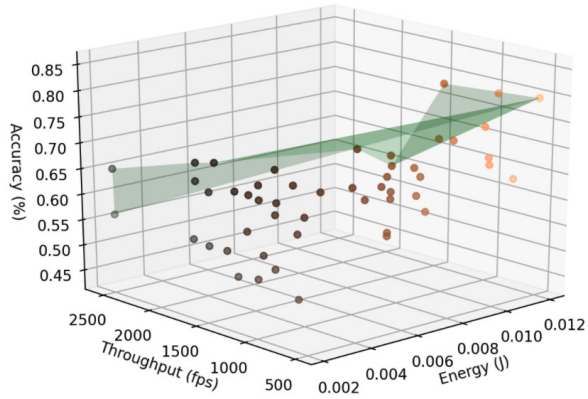
Cluster Type	Layers		Neurons		Kernel	
	β^{min}	β^{max}	η^{low}	η^{up}	κ^{min}	κ^{max}
C_1:Conv	2	7	64	128	3x1	7x1
C_2:MaxP	-	-	-	-	2x1	-
C_3:Conv	2	7	96	256	3x1	7x1
C_4:GlbMaxP	-	-	-	-	2x1	-
C_5:FC	1	4	128	512	-	-

Table 5.3: CIFAR-10 Search Space

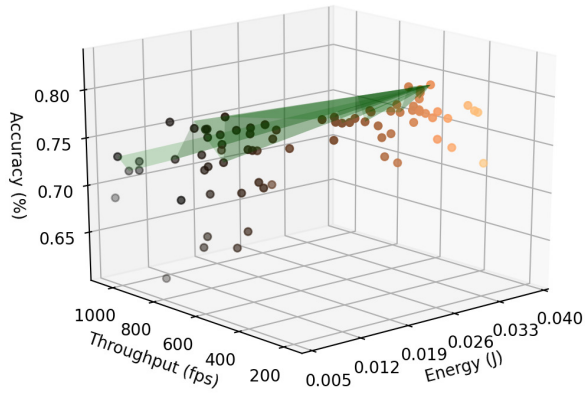
Cluster Type	Layers		Neurons		Kernel	
	β^{min}	β^{max}	η^{low}	η^{up}	κ^{min}	κ^{max}
C_1:Conv	1	3	32	64	3x3	7x7
C_2:Conv+Res	2	4	32	128	3x3	7x7
C_3:MaxP	-	-	-	-	2x2	-
C_4:Conv+Res	2	4	64	256	3x3	7x7
C_5:Conv+Res	2	4	64	256	3x3	7x7
C_6:MaxP	-	-	-	-	2x2	-
C_7:Conv+Res	2	5	128	512	3x3	7x7
C_8:Conv+Res	2	5	128	1024	3x3	7x7
C_9:MaxP	-	-	-	-	2x2	-
C_{10}:FC	1	3	256	1024	-	-



(a) Pascal VOC Pareto Front



(b) PAMAP2 Pareto Front



(c) CIFAR-10 Pareto Front

Figure 5.3: Pareto Fronts based on 3 evaluation parameters, namely, accuracy (F1-score for the Pascal VOC), throughput and energy

cluster specific bounds, namely, the number of layers, the neurons per layer, and the kernel sizes.

Table 5.4 lists the values for all parameters used for training the neural networks and for the EPT algorithm settings.

Table 5.4: Algorithm parameters for Multi-Objective EPT

Parameter		VOC	PAMAP2	CIFAR-10
Mutation change rate	ρ_m	0.10	0.12	0.12
Mutation probability	P_m	0.3	0.3	0.3
Initial Crossover probability	$P_r(0)$	0.3	0.4	0.3
Population size	N_p	60	50	100
No of iterations	N_g	30	60	120
Population replacement rate	Ω	0.02	0.03	0.02
Training Parameters	τ_{params}			
Training size per iteration		1 epoch	1/5 epoch	1/8 epoch
Optimizer		Adam	Adam	Adam
Learning rate		$1e^{-3}$	$1e^{-4}$	$1e^{-3}$
Batch size		10	50	64

Next, an exploration of the defined search space was performed using Algorithm 5. This exploration resulted in a Pareto Front, consisting of CNNs with evaluated objectives, such that an objective can not be improved further without worsening at least one other objective. Figure 5.3 (a), Figure 5.3 (b) and Figure 5.3 (c) illustrate the Pareto Front for the Pascal VOC, the PAMAP2 and the CIFAR-10, respectively. These Pareto Fronts do not include memory evaluations to allow for a comprehensible visualization, since the actual Pareto Fronts created in this experiment are four dimensional. For the Pascal VOC dataset, which is an imbalanced set, the F1-score was used as the efficiency evaluation metric to compare the partially trained CNNs during the search (see Chapter 2 for more details).

The exploration took 6 days with 8 GPUs for the image recognition application (i.e., the Pascal VOC dataset). It took 2.5 days on 4 GPUs for the CIFAR-10 dataset, and 10 hours on 1 GPU for the HAR application (the PAMAP2 dataset). The CNNs in the Pareto Fronts were modified further, by adding a batch normalization layer after every convolutional layer. Subsequently, these models were trained for 250 epochs for the Pascal VOC and the CIFAR-10 and 100 epochs for the PAMAP2. Once the CNNs are trained,

all the objectives are evaluated again to make sure they correctly reflect the modifications applied to the CNNs.

To provide some examples from the Pareto Front, for the Pascal VOC, after completion of the training, the F1-scores were between 77.6 and 72 whereas the memory consumption had a wide range from 0.384 to 0.078 Joules. Similarly, among the CNNs in the Pareto Front for the PAMAP2, the accuracy was between 94.17% to 91.34%, while the memory required was between 4 and 10 MB. For the CIFAR-10, the accuracy values spanned in the middle of 94.86% and 92.84%, whilst the maximum possible throughput was between 230 and 750 fps.

These Pareto Fronts give a system designer an important tool to pick the most suitable neural network to deploy on the resource constrained device. For example, when the memory is limited on the hardware platform, a designer may consider it to be acceptable to use a model with slightly lower accuracy, specially where a lower memory footprint may also result in a better response time of the device.

5.5 SUMMARY

This chapter was focused on the multi-objective extension for the previously proposed EPT algorithm. The flexibility of the algorithm was clearly demonstrated by optimizing simultaneously the neural network architecture for multiple task specific objectives, such as number of parameters, accuracy and hardware specific metrics. This allows for a better resource usage of the neural network on the designated hardware. We envision that the Pareto Front obtained for multiple hardware-specific objectives will allow the designer to have better design choices and more flexibility, in switching from one CNN to another systematically, for both the given task and the hardware. Additionally, the search space was extended to include residual connections to have ResNet style CNNs to be deployed on edge devices. The Pareto Fronts obtained from the hardware-aware EPT experiments are further utilized in the next chapter, which derives scenarios and aims to create an adaptive application, in sync with changes occurring in the environment.

Part II

ADAPTIVE APPLICATIONS

The second part of the thesis investigates methodologies and strategies to ensure adaptivity of CNN-based applications running on edge devices.

Chapter 6, Scenario Based Run-time Switching, introduces the concept of multiple scenarios for an application, where each scenario represents a set of priorities for an operation mode in the target environment. By allowing the scenarios to switch at runtime, the application can adapt to environmental changes during its operation.

Chapter 7, Neural Network Reuse and Composition Update, explores the techniques which work towards ensuring longevity of CNNs already deployed on an IoT network of interconnected edge devices. This work considers neural networks as dynamic entities, which can be continuously adapted and maintained in line with dynamic system behaviors during a long operation time.

6

SCENARIO BASED RUN-TIME SWITCHING

Execution of neural networks on edge devices places numerous demands on the application, a few of them are high accuracy, high throughput, low memory cost, and low energy consumption. As demonstrated through the Pareto Front in Chapter 5, these requirements are very difficult to satisfy at the same time. A CNN execution at the edge typically involves trade-offs, such as high throughput maybe achieved at the cost of decreased accuracy. In a typical system design process, such trade-offs are considered once and a neural network is chosen to be deployed on the hardware and remain fixed during the CNN-based application execution. Alternatively, they are adapted to suit the properties of the CNN input data or target hardware, for example through compression or quantization. However, the requirements of the application can also be fundamentally influenced by the changes in the application environment during its execution, for instance a change of the battery level in the edge device. Therefore, CNN-based applications will benefit from a mechanism which allows a dynamic adaptation towards their extra-functional characteristics with respect to the changes in the application environment at run-time. This chapter presents a scenario-based run-time switching (SBRS) methodology, which implements such a mechanism.

This Chapter is based on:

- S. Minakova, **D. Sapra**, T. Stefanov and A. D. Pimentel *"Scenario Based Run-time Switching for Adaptive CNN-based Applications at the Edge"* [102], in *ACM Transactions on Embedded Computing Systems*, © ACM.

6.1 INTRODUCTION

During the design process of a CNN-based application, the selection of the requirements specific to the applications is typically carried out once, and remains static during the execution. In practice, these priorities are often influenced by the application environment, and may well need to be adjusted during the application execution. The provision for this sort of adaptivity can ensure that the application is compliant with its environment while executing on the edge during all of its operation time.

As an example, a CNN-based road traffic monitoring application, deployed on an Unmanned Aerial Vehicle (UAV) [119], may have different priorities at different operation times, depending on the road situations and the level of the device's battery. When there is heavy traffic, it is highly desirable to have high throughput from the application as well as a high accuracy to process the input data, which in turn might assert a higher energy cost. On the other hand, a high throughput is not warranted in case of a traffic jam. In a similar manner, when the battery of the UAV is running low, the application would work optimally by prioritizing energy efficiency over high throughput. This example shows that CNN-based applications would benefit from a mechanism which allows dynamic adaptation of its characteristics with respect to the changes in the application environment at run-time (such as a change of the situation on the roads or a change of the device's battery level).

Moreover, the aforementioned mechanism should also be able to maintain a high level of responsiveness. For example, the switch to an energy-efficient mode should be as swift as possible when the battery is running low, during the execution of the CNN-based application on the UAV. However, to the best of our knowledge, neither existing Deep Learning methodologies [106–111, 117, 120–123] for resource-efficient CNN execution at the edge, nor existing embedded systems design methodologies [124–126] for execution of run-time adaptive applications at the edge, provide such a mechanism.

In this chapter, we introduce a novel scenario-based run-time switching (SBRS) methodology for CNN-based applications for execution on edge devices. In this methodology, a CNN-based application is associated with several scenarios, where each scenario represents a set of priorities for an operation mode at run-time. Every scenario corresponds to a CNN, which is specifically designed to conform to certain requirements for accuracy, throughput, memory cost, and energy cost. During the execution of the

application, the environment (or its monitor) can trigger a switch from current scenario to another scenario, thereby adapting the characteristics of the CNN-based application to the changes in its environment.

To capture multiple application scenarios and allow for run-time switching between these scenarios, the CNN-based application is represented using the novel SBRS Model of Computation (SBRS MoC). We note that, being associated with multiple scenarios, each of which is essentially a neural network, such an application within the SBRS methodology can have a high memory cost. With resource limitations on edge devices, high memory cost is undesired for such applications at the edge. To reduce the application memory cost, the SBRS MoC also includes the efficient reuse of components (layers and edges) among the different scenarios, and also within every scenario. Additionally, to ensure high application responsiveness to a scenarios switch request, the SBRS methodology also includes a transition protocol (SBRS TP). The SBRS TP specifies switching from the old application scenario to a new application scenario so that both old and new scenarios remain consistent, and the new scenario can begin execution as soon as possible. The SBRS TP also ensures that there is part overlapping between the execution of the old and new scenarios, in order to improve responsiveness.

In this chapter, we only present the portion of the SBRS methodology related to scenario derivation and later present a comparative study with another adaptive methodology, called MSDNet, which is closest to the SBRS methodology in terms of adaptive execution. The development of the SBRS MoC and transition protocol has been conducted by Leiden University as a collaboration, and these parts of the SBRS methodology are, therefore, not presented in this thesis.

6.2 RELATED WORK

The platform-aware neural architecture search (NAS) methodologies, proposed in [106, 107, 120–123] and reviewed in survey [117], allow for automated generation of different CNNs, which are characterized with different accuracy, throughput, energy cost and memory cost. However, these methodologies do not propose a mechanism for run-time switching between these CNNs, while such mechanism is necessary to ensure that application needs are best served at every moment in time. In contrast to these NAS methodologies, the SBRS methodology proposes such a mechanism, and ensures that application needs are best served at every moment in time.

The methodologies presented in [127–131] propose resource-efficient and runtime-adaptive CNN execution at the edge. These methodologies represent a CNN as a dynamic computational graph, where for every CNN input sample only a subset of the graph nodes is utilized to compute the corresponding CNN output. The subset of graph nodes is selected during the application run-time by special control mechanisms (e.g., control nodes, augmenting the CNN graph topology). The utilization of only a subset of graph nodes at every CNN computational step can increase the CNN throughput and accuracy, and typically reduces the CNN energy cost. However, these methodologies cannot adapt a CNN to changes in the application environment, like changes of the device's battery level, which affect the CNN needs during the run-time. The adaptation in these methodologies is driven either by the complexity of the CNN input data [128–132] or by the number of floating-point operations (FLOPs), required to perform the CNN functionality [127, 131], while the changes in the application environment often cannot be captured in the CNN input data or estimated using FLOPs. In contrast to these methodologies, the SBRS methodology adapts a CNN-based application to the changes in the application environment, and therefore, allows to best serve the application needs, affected by such changes.

A number of embedded systems design methodologies, proposed in [124–126], allow for efficient execution of runtime-adaptive scenario-based applications at the edge. These methodologies represent an application, executed at the edge in a specific model of computation (MoC), able to capture the functionality of a runtime-adaptive application associated with several scenarios, and ensure efficient run-time switching between the application scenarios. However, these methodologies cannot be (directly) applied to CNN-based applications due to a significant semantic difference between the MoCs, utilized in these methodologies and the CNN model [133], typically utilized by CNN-based applications. First of all, the MoCs utilized in these embedded systems design methodologies lack means for explicit definition of various CNN-specific features, such as CNN parameters and hyper-parameters, while explicit definition of these features is required for the application analysis. Secondly, the MoCs utilized in these methodologies are not accepted as input by existing Deep Learning frameworks, such as Keras [34] or TensorRT [134], widely used for efficient design, deployment and execution of CNN-based applications at the edge.

In the SBRS methodology, we propose a novel application model, inspired by the embedded systems design methodologies [124–126], to rep-

resent a run-time adaptive CNN-based application and ensure efficient switching between the CNN-based application scenarios. However, unlike these, the SBRS methodology 1) explicitly defines and utilizes CNN-specific features for efficient execution of CNN-based applications at the edge, and 2) allows for utilization of existing deep learning frameworks for design, deployment, and execution of the CNN-based application at the edge.

6.3 MOTIVATIONAL EXAMPLE

In this section, we show the necessity of devising a new methodology for execution of adaptive CNN-based applications at the edge. To do so, we present a simple example of a CNN-based application where the requirements change at run-time due to the changes in its environment. The application is discussed in the context of the existing methodologies reviewed in the "Related Works" Section, and the scenario-based run-time switching (SBRS) methodology.

The example application performs CNN-based image recognition on a battery powered Unmanned Aerial Vehicle (UAV). The UAV battery capacity defines a power budget, which is available for both the flight and CNN-based application execution. The distribution of the power budget between the flight and application is irregular, and depends on the weather conditions, which can change during the run-time (the UAV flight). In calm weather, the UAV requires less power to fly and can thus spend more power on the CNN-based application. Conversely, when the weather is windy, the UAV requires a large amount of power to fly, and therefore has less power available for the CNN-based application. The weather prediction at the application design time is an impossible task. Nevertheless, the CNN-based application should be designed such that it: 1) meets the power constraint, imposed on the application by the UAV battery and affected by weather conditions; 2) demonstrates high image recognition accuracy (the higher the better).

Figure 6.1 illustrates an example of how the execution of such CNN-based application will transpire, when designed using the existing methodologies and the SBRS methodology. Subplots (a), (b), (c) juxtapose the power available for the application execution (dashed line), against the power used by the application (solid line) during the UAV flight, which lasts 2 hours. The power available for the application execution is dependant on the UAV battery capacity and weather conditions. In this example, we assume that the CNN-based application is allowed to use up to 12 Watts of power in

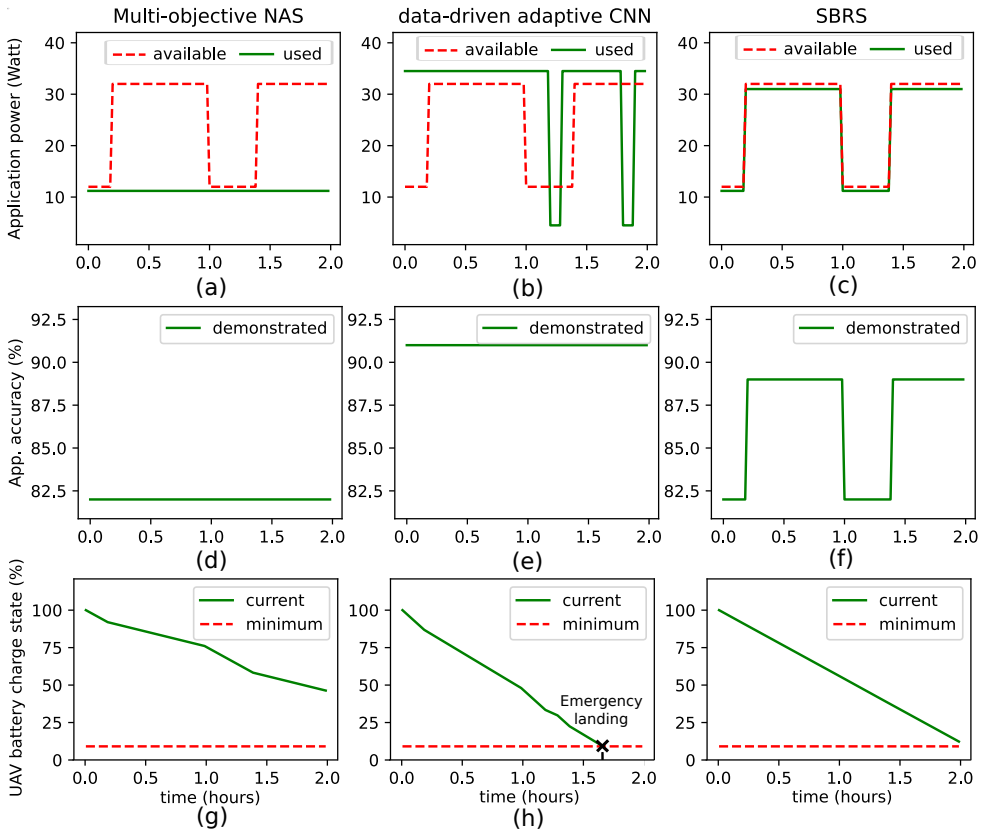


Figure 6.1: Execution of a CNN-based application, affected by the application environment and designed using different methodologies

turbulent weather (0 to 0.1 hours and 1.0 to 1.5 hours) and up to 32 Watts of power in calm weather (0.1 to 1.0 hours and 1.5 to 2.0 hours). However, the actual power used by the application is ultimately determined by the application design methodology. Further, the subplots (d), (e), (f) show the image recognition accuracy demonstrated by the application. Subplots (g), (h), (i) show the current charge state (solid line) and minimum charge level (dashed line) of the UAV battery. If the current battery charge reaches the minimum allowed battery level, it may lead to an emergency landing of the UAV.

As a first case, we discuss the multi-objective NAS methodologies [106, 107, 120–123] for the execution of the example application, that are typically designed and utilized without considering a run-time changing environment. In these methodologies, a CNN is obtained via an automated multi-objective search and characterized with constant accuracy and power con-

sumption. To guarantee that the application meets a power constraint, such a CNN has to account for the worst-case scenario, i.e., when the weather is always windy and therefore only 12 Watts are available for the application execution at any moment. In the illustrative example, such a CNN is characterized with 11.2 Watts of power and 82% accuracy (see Figure 6.1 (a) and Figure 6.1 (d), respectively). As shown in Figure 6.1 (g), when the UAV reaches its destination after 2 hours of flight, it still has ≈50% battery charge left. On the one hand, it means that the application always meets the power constraint. On the other hand, the application could have spent ≈40% remaining UAV battery charge by utilizing a more accurate CNN, though demanding additional power. In other words, *the multi-objective NAS methodologies in [106, 107, 120–123] can guarantee that the application meets the given platform-aware constraints, but cannot guarantee efficient use of available platform resources*.

As a second case, when the application is designed using data-driven adaptive methodologies, such as [128–132], the CNN execution is sensitive to the input data complexity. To process "easy" images, they may use a lower resolution or fewer layers, whereas processing "hard" images requires more computation. In this manner, an adaptive CNN-based application is able to adapt its power consumption depending on the input data complexity, while demonstrating similar accuracy for all the inputs. However, such a CNN cannot adapt to the changing environmental conditions, which can not be explicitly captured in the input images. The application power consumption can change during the application run-time, based on the input images, although these changes may conflict with the application's requirements, driven by the weather conditions. In Figure 6.1 (b), the CNN consumes significant amount of power, between 1.0 and 1.25 hours, despite the necessity to switch to the low power mode. This may lead to increased UAV power consumption over the flight duration and, eventually, to a violation of the application power constraint, causing an emergency landing as illustrated in Figure 6.1 (h). Thus, *the data-driven adaptive methodologies in [128–132] are not suitable for CNN-based applications executed at the edge in a changing environment, because these can neither properly adapt the application to the environment variations, nor guarantee that the application constantly meets platform-aware constraints*.

Another case of adaptive CNN-based application methodologies, is where the application can adaptively change the number of floating-point operations (FLOPs) spent on the image recognition, such as those in [127, 131]. However, as shown in numerous works [120, 135, 136] FLOPs is an inaccu-

rate indicator for real-world platform-aware characteristics such as power consumption or throughput. These characteristics depend on many other factors, for instance, the ability of the platform to perform parallel computations, time and energy overheads caused by the data transfers, internal hardware limitations, etc. Consequently, the number of FLOPs spent during the application run-time, neither guarantee that the application meets power constraints nor estimate the application efficiency in terms of real-world platform-aware characteristics. In other words, *even though, the methodologies in [127, 131] enable run-time CNN adaptivity, these cannot be directly deployed for applications with real-world platform-aware requirements and constraints*.

To summarize, the existing works lack a methodology to design an adaptive CNN-based application, for real-world platform-aware requirements and constraints, specifically affected by the environment variations at run-time. The motivation behind the SBRS methodology is to enable such run-time adaptivity. To design an application using the SBRS methodology, we perform multi-objective NAS, similar to those in [106, 107, 120–123]. However, unlike these methodologies, we derive multiple CNNs for each scenario. For example, the first scenario for the example application for windy weather, can have an associated CNN with 11.2 Watts power consumption and 82% accuracy. The second scenario, for calm weather, is represented by a CNN with 31.0 Watts power consumption and 89% accuracy. At run-time, the application switches between these scenarios, based on the weather conditions. Additionally, the SBRS methodology explicitly defines the switching mechanism based on triggers generated due to an environment change at run-time. The execution of the CNN-based application with the SBRS is shown in Figure 6.1 (c), (f), (i). Particularly, Figure 6.1 (i) highlights that the application meets the given power constraint, i.e. the UAV battery charge does not go below the minimum level before 2 hours, and the SBRS uses all available power to achieve higher application accuracy in comparison with Figure 6.1 (d). Thus, *by switching among the scenarios, the SBRS guarantees that a CNN-based application, affected by the environment, meets platform-aware constraints while efficiently exploiting the available platform resources to improve its accuracy*.

6.4 SBRS METHODOLOGY

In this section, we present the scenario-based run-time switching (SBRS) methodology, which allows for run-time adaptation of a CNN-based application to changes in the application environment, for an edge device.

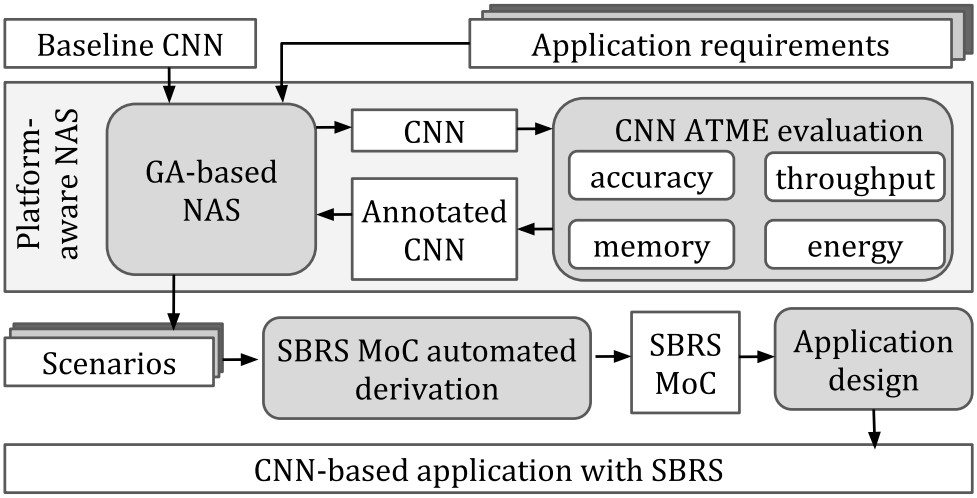


Figure 6.2: SBRS methodology

The general structure of the methodology is illustrated in Figure 6.2. The methodology accepts as an input a baseline CNN and one or more requirements sets, associated with the CNN-based application. A baseline CNN is an existing CNN (e.g., AlexNet [116], ResNet [116], or another), proven to achieve good results at solving a CNN-based application task (e.g., classification). The requirements sets describe a scope of needs, associated with the devised application. Every application requirements set $r = (r_a, r_t, r_m, r_e)$ specifies the application priority for high accuracy (r_a), high throughput (r_t), low memory cost (r_m), and low energy cost (r_e), respectively. One application can have one or several sets of requirements, characterising the application needs at different times of the application execution. The requirements sets are defined by the application designer at the application design time. As an output, the methodology provides a CNN-based application with the SBRS capabilities, able to adapt its characteristics to the changes in the application environment during the application run-time.

The SBRS methodology consists of three main steps, performed offline. At Step 1, for every set of application requirements r, accepted as an input, an application scenario is derived, i.e., a CNN that conforms to the given set r of application requirements. To perform this step, we use the automated hardware-aware EPT algorithm, as explained in detail in Chapter 5. At Step 2, the scenarios generated by Step 1 are utilized to automatically derive a SBRS MoC of a CNN-based application with scenarios. The SBRS MoC captures the scenarios associated with the CNN-based application,

and allows for run-time switching among these scenarios. Moreover, the SBRS MoC features efficient reuse of the components (layers and edges) among and within application scenarios, thereby ensuring efficient utilization of the platform memory by the CNN-based application with the SBRS. Finally, at Step 3, the SBRS MoC derived at Step 2 is used to design the final implementation of the CNN-based application with the SBRS. The final implementation of the CNN-based application is deployed on the edge device to perform the application functionality with run-time adaptive switching among the application scenarios, and follows the transition protocol (SBRS TP). As mentioned earlier, the SBRS MoC and the SBRS TP are presented in the original paper on which this chapter is based [102], but are not part of this thesis.

6.4.1 *Scenarios derivation*

In this section, we discuss the automated derivation of application scenarios, which essentially generates a collection of CNNs. Each CNN services a different set of requirements, that are determined by its associated scenario. The derivation process builds upon the hardware-aware Evolutionary Piecemeal Training (EPT) methodology presented in Chapter 5, which searches for the best CNN in terms of ATME characteristics. ATME refers to four objectives, namely, Accuracy, Throughput, Memory and Energy, which are important for the execution of a CNN-based application on a resource-constrained edge device. The hardware-aware EPT algorithm results in a Pareto Front, which represents the set of neural networks with the ATME characteristics, where none of the objectives can be further improved without worsening some of the other objectives

The scenario selection task, which follows the pareto set creation, refers to the selection of the appropriate model designated for each scenario. Every intended scenario is depicted by a requirements set $r = (r_a, r_t, r_m, r_e)$, where r_a, r_t, r_m, r_e refers to the importance of accuracy, throughput, memory and energy, respectively. Together, these variables constitute the influence factor of each objective in the scenario by assigning a weight value to the requirements such that $r_a + r_t + r_m + r_e = 1.0$. For example, in a scenario where only high accuracy is pivotal, i.e. $r_a = 1.0$, the requirements set is $r = (1.0, 0.0, 0.0, 0.0)$. However, in a scenario where all the objectives are equally important, the requirements set becomes $r = (0.25, 0.25, 0.25, 0.25)$. For a complex scenario where the throughput and energy are critical fac-

tors and accuracy is still moderately significant, the requirements set may be represented as $r = (0.2, 0.4, 0.0, 0.4)$.

The next task is to post-process all the CNN models in the pareto set, for instance adding *BatchNorm* layers after every *Conv* layer. These CNNs are not fully trained yet by the Algorithm 5, hence they are further trained, to achieve the best possible accuracy. Subsequently, hardware metrics can once more be evaluated at this point, especially if the structure of the CNN was modified, such as by adding or removing some layers. For every CNN model in the pareto set, each objective is separately ranked from 1 to N, where 1 is the best value of an objective (in the set), and N, on the other hand, is the worst. The ranking dominance concept, introduced in [137], has been extended here with weighted aggregation of ranks based on requirements set to derive a suitable CNN model to represent a scenario.

For a model CNN_i, having a rank R_{Oi} for a given objective O, and associated requirement value r_o, its weighted rank wR_{Oi} for the objective in consideration is computed as $r_o * R_{Oi}$. Subsequently, for each scenario, the weighted ranks are aggregated using the following equation

$$wR_{scn} = \sum_{\forall O \in \Theta} (r_o * R_{Oi}) \tag{17}$$

where Θ is the set of all objectives. For the specific objectives in this work, i.e. Accuracy (Λ), Throughput (T), Memory(M) and Energy (ξ) for a model CNN_i, the equation translates to

$$wR_{scn} = (r_a * R_{\Lambda_i}) + (r_t * R_{T_i}) + (r_m * R_{M_i}) + (r_e * R_{\xi_i}) \tag{18}$$

After computation of the weighted rank, wR_{scn}, for each scenario, the lowest rank value is considered to be the best model representing that scenario. The weighted ranks and their respective aggregation is computed for each scenario in the application. In a situation where two or more models have the lowest rank value, a random model amongst them may be chosen. Alternatively, the ranks can be computed again with a slightly altered requirements set, such as assigning slightly higher importance to the accuracy requirement. Figure 6.3 exemplifies the process of a scenario selection where the scenario requirements set is $(r_a = 0.4, r_t = 0.3, r_m = 0.1, r_e = 0.2)$, i.e., in this scenario all requirements have varying degrees of importance: high accuracy being the most crucial and memory being the least important one.

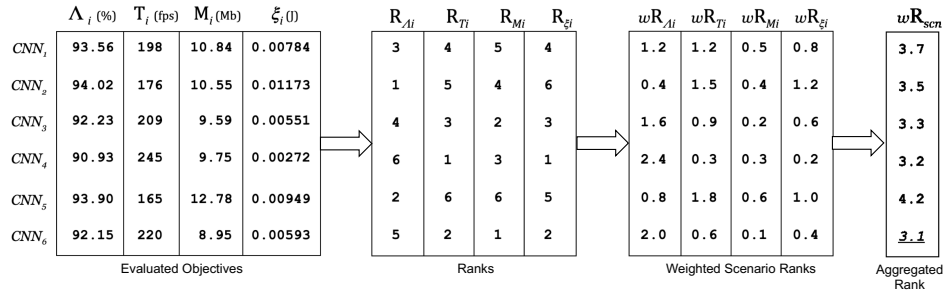


Figure 6.3: An example of scenario selection. First, a simple ranking is applied to evaluated objectives. Next, the scenario requirements set ($r_a = 0.4, r_t = 0.3, r_m = 0.1, r_e = 0.2$) is used to compute the weighted ranks for the given scenario. Finally, the aggregated rank is calculated and the model with the lowest rank value (CNN_6) is selected as the model associated with this scenario.

6.5 EXPERIMENTAL STUDY

To evaluate the SBRS methodology, we perform an experiment by applying the methodology to three CNN-based applications with scenarios. The Pareto Fronts derived from the hardware-aware EPT algorithm, as explained in Chapter 5, are utilized to automatically derive a set of scenarios for all three CNN-based applications. The merits of the SBRS methodology are demonstrated through three applications from two different domains, namely Human Activity Recognition (HAR) and image classification. We used the PAMAP2 dataset for HAR and the Pascal VOC and the CIFAR-10 datasets for image classification.

As indicated in the motivational example, none of the existing works can currently design an adaptive CNN-based application, where platform-aware requirements and constraints are specifically considered for the environment changes occurring at run-time. Within this context, none of the existing works is completely comparable to the SBRS methodology. Nonetheless, we perform a partial comparison between the SBRS methodology and the most relevant existing work. Among the existing works, the MSDNet adaptive CNN methodology [131] is the most relevant to the SBRS methodology.

The MSDNet, which is similar to the SBRS methodology, associates a CNN-based application with multiple alternative CNNs that are characterized with different trade-offs between accuracy and resource utilization, and can be used to process application inputs of any complexity. Addi-

tionally, both the MSDNet and the SBRS methodologies provide means to reduce the memory cost of a CNN-based application by reusing the memory among the alternative CNNs. In this sense, the MSDNet and the SBRS methodologies can be compared via 1) CNNs, designed for a specific dataset and edge platform; 2) run-time adaptive trade-offs between application accuracy and resource utilization; 3) memory efficiency. Such a comparison is undertaken using the image recognition CIFAR-10 dataset.

6.5.1 *Application requirements*

The main features and requirements for each CNN-based application are listed in Table 6.1. The table summarizes the application name, the tasks they perform along with the baseline CNN that was deployed to perform the application tasks and the real-world datasets, which were used to train and validate the applications' baseline CNNs. The last column shows the sets of application requirements $r_i, i \in [1, S]$, where every set r_i characterizes a scenario, associated with the CNN-based application, S is the total number of CNN-based application scenarios. The applications use extremely different baseline CNNs (from the deep and complex ResNet based topology [5] to the small and shallow PAMAP topology) and diverse datasets (from the large Pascal VOC dataset to the small PAMAP2 and CIFAR-10 datasets). The ResNet based baseline topologies for the VOC and the CIFAR-10 applications are custom Resnets, both of which are smaller than the popular ResNet-18. This leads to diversity in scenarios and the SBRS MoCs, derived for these applications and, hence, provides a sufficient basis for evaluation of the effectiveness of the SBRS methodology.

6.5.2 *Automated scenarios derivation*

Firstly, all the objectives of fully-trained CNNs in the Pareto Front achieved from the hardware-aware EPT (Chapter 5) were ranked individually. This exercise was performed for the NVIDIA Jetson TX2 embedded platform [104]. Secondly, the rank based weighted aggregation was performed for each scenario, using the requirement sets from Table 6.1 for the three applications. The selected CNNs for each scenario after rank aggregation are presented in Table 6.2, Table 6.3, and Table 6.4 for Pascal VOC, PAMAP2 and CIFAR-10, respectively.

As the evaluation metric, the accuracy was computed for PAMAP2, and CIFAR-10, while PR-AUC (Area under precision-recall curve) was used for

Table 6.1: Application requirements sets for the scenarios

App.	task	baseline CNN	dataset	app. requirements sets
Pascal VOC	Image recongition	ResNet [5]	Pascal VOC [23]	r_1=(1.0, 0.0, 0.0, 0.0) r_2=(0.7, 0.0, 0.3, 0.0) r_3=(0.6, 0.1, 0.0, 0.3) r_4=(0.5, 0.5, 0.0, 0.0) r_5=(0.1, 0.1, 0.4, 0.4)
PAMAP2	Human activity monitoring	PAMAP (CNN-2) [73]	PAMAP2 [24]	r_1=(1.0, 0.0, 0.0, 0.0) r_2=(0.2, 0.4, 0.0, 0.4) r_3=(0.5, 0.0, 0.0, 0.5) r_4=(0.5, 0.5, 0.0, 0.0)
CIFAR-10	Image recognition	ResNet [5]	CIFAR-10 [22]	r_1=(1.0, 0.0, 0.0, 0.0) r_2=(0.25, 0.25, 0.25, 0.25) r_3=(0.5, 0.25, 0.0, 0.25) r_4=(0.5, 0.0, 0.0, 0.5)

the Pascal VOC (See Chapter 2 for more details). The hardware-aware EPT took 6 days with 8 GPUs for the Pascal VOC dataset. While it took 2.5 days on 4 GPUs for the CIFAR-10 dataset, and 10 hours on 1 GPU for the PAMAP2 dataset.

The throughput, typically measured in frames per second (fps), characterizes the speed with which the CNN is able to process input data and produce output data. The memory cost, typically measured in Megabytes (MB), specifies the total amount of memory required to execute a CNN. The energy cost, measured in Joules, specifies the amount of energy consumed by a CNN to process one input frame.

The scenarios that were eventually automatically derived in the experiments, showcase a compelling representation of the application requirements. For instance, the Pascal VOC scenarios have contrasting requirements in r_1 and r_5; r_1 demands best possible model efficiency, while on the other hand, r_5 demands low memory and energy usage. In line with the requirements, the scenario for r_1 has the best associated CNN in terms of high PR-AUC score, though with a high memory and energy cost. Whereas, the CNN for r_5 consumes significantly less memory and energy than the former, but with a lower PR-AUC score. In yet another example, if the CNNs for r_1 and r_2 are compared, it is observed that both demand high efficiency, while r_2 additionally demands a lower memory footprint. The scenario that was derived for r_2 requires almost 25% less memory at the cost of a small dip in the PR-AUC score.

Table 6.2: VOC scenarios

Req. set	PR-AUC	Throughput (fps)	Memory (MB)	Energy (J)
r_1	77.78	15.41	292.61	0.384
r_2	76.28	21.78	210.69	0.281
r_3	77.69	20.26	242.72	0.291
r_4	73.99	59.27	155.48	0.101
r_5	72.85	75.07	130.21	0.078

Table 6.3: PAMAP2 scenarios

Req. set	Accuracy (%)	Throughput (fps)	Memory (MB)	Energy (J)
r_1	94.17	510.20	10.02	0.0083
r_2	91.34	1333.33	4.30	0.0033
r_3	92.56	970.87	4.86	0.0037
r_4	92.93	1052.63	4.11	0.0039

Table 6.4: CIFAR-10 scenarios

Req. set	Accuracy (%)	Throughput (fps)	Memory (MB)	Energy (J)
r_1	94.86	231.80	52.87	0.0242
r_2	92.84	754.15	13.07	0.0055
r_3	93.46	538.79	18.30	0.0081
r_4	94.46	403.71	28.07	0.0121

For the PAMAP2 application, a similar CNN ensemble with various requirement sets is automatically derived. For example, r_1 and r_2 requirement sets place contradicting demands: r_1 demands higher accuracy, whereas r_2 has more focus on energy and throughput. The derived CNN for r_1 has high accuracy, while the CNN for r_2 has lower accuracy, but $\approx$2.5x better throughput and more than halves the energy usage.

Comparably, CNNs were derived for the CIFAR-10 application in the same manner. To illustrate, r_1 and r_2 requirement sets purposefully differ from each other in their demands. The r_1 requires high accuracy, whereas the r_2 considers all of the measured characteristics to have the same importance. Comparing the derived CNNs for r_1 and r_2, it is clearly observable that r_1 CNN has a high accuracy, while r_2 CNN with a lower accuracy, performs better on all other parameters. These experiments clearly illustrate that the scenario derivation enables automatic generation of diverse CNNs with different ATME characteristics.

6.5.3 *Comparative study*

In this section, the SBRS methodology is compared to the MSDNet adaptive CNN methodology [131]. The MSDNet proposes an adaptive CNN-based application which allows multiple exit points in a large neural network, depending upon the input complexity and hardware resource budget allocated to the application. Similarly to the SBRS methodology, the MSDNet associates a CNN-based application with multiple alternative CNNs that are characterized with different trade-offs between accuracy and resource utilization, and can be used to process application inputs of any complexity. In this sense, the MSDNet and the SBRS methodologies can be compared via 1) CNNs, designed for a specific dataset and edge platform; 2) run-time adaptive trade-offs between application accuracy and resource utilization; 3) memory efficiency.

The CNNs obtained using the SBRS methodology and the MSDNet methodology, for the purpose of comparison, perform image classification on the CIFAR-10 dataset. We refer to these CNNs as to the SBRS points and the MSDNet points, respectively. The MSDNet points, i.e., subgraphs or *exits* of the MSDNet CNN, are derived using the official implementation of the MSDNet methodology, executed with design and training parameters specified already for the CIFAR-10 dataset. In total, there are six MSDNet points. For the SBRS points, we obtained eight SBRS points that are pareto-optimal in terms of the ATME characteristics from the hardware-aware EPT algo-

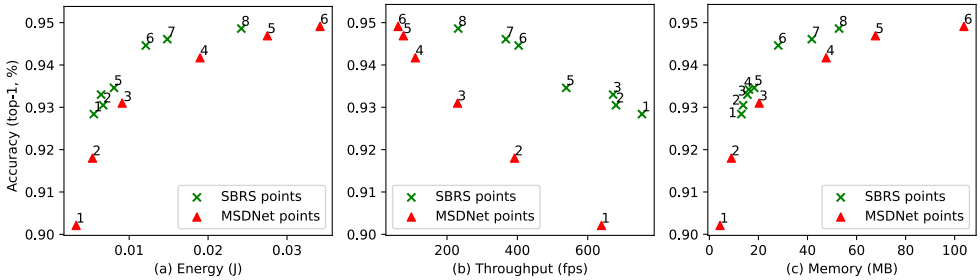


Figure 6.4: Comparison among the SBRS and the MSDNet [131] points

rithm. These points are not the final scenarios as portrayed in Table 6.4, but the pareto-optimal CNNs resulting from the hardware-aware EPT (though, post-processed further and fully-trained). The scenarios presented were derived based on a weighted ranking from this pareto set of CNNs.

To compare the MSDNet points with the SBRS points, we have evaluated the ATME characteristics of all the points on the same hardware. The platform-aware characteristics (throughput, memory, and energy) are measured on the NVIDIA Jetson TX2 edge platform [104].

The SBRS and the MSDNet points comparison is shown in Figure 6.4. Considering that it is not easy to draw and understand four-dimensional plots, the comparison is represented as three two-dimensional plots, subplots (a), (b) and (c), each comparing one of the platform-aware CNNs characteristics to the CNNs accuracy. The accuracy (the higher the better) is always on the vertical axis with different platform-aware characteristics on the horizontal axis: energy (the lower the better), throughput (the higher the better) and memory cost (the lower the better), respectively. Each subplot shows the six points for the MSDNet and those SBRS points that are pareto-optimal in terms of respective platform-aware characteristics.

Beside the visualization, these plots also provide insight into the key difference between the SBRS methodology and the MSDNet. It can be clearly observed in Figure 6.4 that the SBRS points are able to achieve similar accuracy when compared to the MSDNet points, but with lower energy cost, higher throughput, and lower memory cost. We believe that the reason for this direct distinction is caused by the optimization, applied (through the hardware-aware NAS) by the SBRS methodology, to every SBRS point to meet the platform-aware needs, while the MSDNet CNN does not provide such optimization. The plots in Figure 6.4 undoubtedly reveal that the SBRS points are a better choice for using them as scenarios in the SBRS methodology compared to the MSDNet points because none of the MSD-

Net points pareto-dominates the SBRS points but many of the SBRS points pareto-dominate the MSDNet points.

Additionally, to study the efficiency of the proposed methodology, we compare accuracy and throughput characteristics of the MSDNet CNN and the SBRS MoC, both constructed for an example CNN-based application. The example application performs classification on the CIFAR-10 dataset, and is affected by the application environment at run-time.

The MSDNet CNN is constructed according to the design and training parameters specified for the CIFAR-10 dataset in the original MSDNet work [131]. It has six exits, characterized with different accuracy and throughput. During the application run-time, the MSDNet CNN can yield data from different exits, thereby offering various trade-offs between the application accuracy and throughput. We evaluate these trade-offs by executing the MSDNet CNN with an *anytime prediction* setting. This setting allows the MSDNet CNN to switch among its subgraphs (exits), thereby adapting the MSDNet CNN to changes in the application environment.

We note that in the original work the switching among the MSDNet CNN exits is driven by a resource budget given in FLOPs, not by a throughput requirement. However, conceptually, it is possible to extend the MSDNet CNN with a throughput-driven adaptive mechanism. In this experiment, we emulate execution of the MSDNet CNN with such a mechanism in order to enable direct comparison of the MSDNet CNN with the SBRS MoC.

The SBRS MoC was created using the same settings as presented in Table 5.3, and three sets of application requirements, specifically looking only at accuracy and throughput for this comparative study. In the first set $r_1 = \{0.1, 0.9, 0.0, 0.0\}$, the application prioritizes high throughput over high accuracy. In the second set $r_2 = \{0.5, 0.5, 0.0, 0.0\}$, high throughput and high accuracy are equally important for the application. In the third set $r_3 = \{0.9, 0.1, 0.0, 0.0\}$, the application prioritizes high accuracy over high throughput. The obtained the SBRS MoC has three scenarios corresponding to the three sets of requirements r_1, r_2, and r_3. During the application run-time the SBRS MoC can switch among its scenarios, thereby offering various trade-offs between application accuracy and throughput, and adapting the application to changes in the application environment at run-time.

The comparison between accuracy and throughput characteristics of the aforementioned MSDNet CNN and the SBRS MoC, is visualized in Figure 6.5. The horizontal axis shows throughput (in fps). The vertical axis shows accuracy (in %). The two step-wise curves in Figure 6.5 represent the relationships between accuracy and throughput, exhibited by the MS-

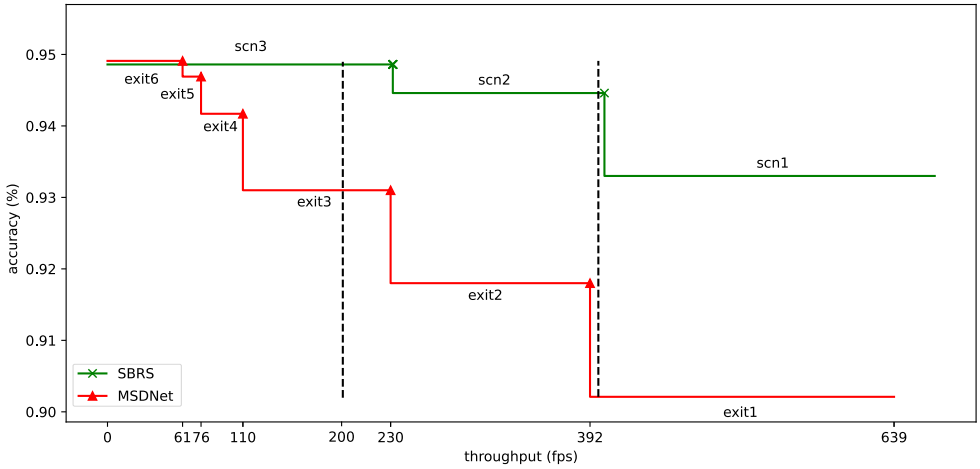


Figure 6.5: Comparison between the SBRS MoC and the MSDNet CNN [131], performing classification on the CIFAR-10 dataset with throughput-driven adaptive mechanism

DNet CNN and the SBRS MoC. Each flat segment of the step-wise curves represents a scenario in the SBRS MoC or an exit in the MSDNet CNN. For example, the flat segment of the MSDNet curve, characterized with throughput between 231 and 392 fps and accuracy of 0.918%, represents exit 2 of the MSDNet CNN. Each cross marker or triangle marker represents a switching point between the SBRS MoC scenarios or the MSDNet CNN exits, respectively.

As explained above, run-time switching among the scenarios or exits occurs when the application is affected by changes in its environment at run time. Figure 6.5 illustrates such changes in the application environment as the two vertical dashed lines, representing demands of minimum throughput, imposed on the application by the environment at run time. For instance, at the start of the application execution, the environment demands that the application must have a throughput of no less than 200 fps with as high as possible accuracy. In this case, the MSDNet CNN yields data from exit 3, demonstrating 0.931% accuracy, and the SBRS MoC executes in scenario 3, demonstrating 0.949% accuracy. Later, the application environment changes and demands that the application must have throughput of no less than 394 fps. Thus, the MSDNet CNN starts to yield data from exit 1, demonstrating 0.902% accuracy, and the SBRS MoC switches to the next scenario, demonstrating 0.946% accuracy.

As depicted in Figure 6.5, the SBRS MoC exhibits higher accuracy than the MSDNet CNN for any throughput requirement, except when the application has to exhibit throughput lower or equal to 61 fps. In the latter case, the accuracy of the SBRS MoC is comparable (0.05% lower) to the accuracy of the MSDNet CNN. We believe that the difference in accuracy between the SBRS MoC and the MSDNet CNN occurs because the scenarios in the SBRS MoC are optimized for both high accuracy and high throughput, whereas the exits of the MSDNet are only optimized for high CNN accuracy. Optimization for the platform-aware requirements performed during the SBRS MoC design enables for more efficient utilization of the platform resources, and therefore for more efficient execution of the application when high throughput is required.

6.6 SUMMARY

In this chapter, a novel methodology has been introduced, which provides run-time adaptation for CNN-based applications executed at the edge to changes in the application environment. We evaluated the proposed SBRS methodology by designing three real-world run-time adaptive applications in the domains of Human Activity Recognition (HAR) and image classification, and executing these applications on the NVIDIA Jetson TX2 edge device. The experimental results show that for real-world applications, the SBRS methodology enables the efficient automated design of CNNs, characterized with different accuracy, throughput, memory cost and energy consumption. It further enables adaptive execution on the edge by allowing different CNNs to be switched during run-time, depending on the changes in the environment.

Additionally, we compared the SBRS methodology to the run-time adaptive MSDNet CNN methodology, which is the most relevant to the SBRS methodology among the related work. The comparison is performed by CNNs designed for the CIFAR-10 dataset and executed on the Jetson TX2 edge device. The comparison illustrates that the application designed using the SBRS methodology outperforms the MSDNet CNN when executed under tight platform-aware requirements, and demonstrates comparable accuracy against the MSDNet CNN when the platform-aware requirements are relaxed. The difference can be attributed to the fact that unlike the MSDNet CNN, the SBRS methodology optimizes the application in terms of both high accuracy and platform-aware characteristics.

7

NEURAL NETWORK REUSE AND COMPOSITION UPDATE

Deep learning has inadvertently pioneered the transition of big data into big knowledge. Neural networks absorb and incorporate knowledge from large scale data through training and can be regarded as a representation of the knowledge learnt. There are multitude of use cases where this acquired knowledge can be used to enhance future applications or speed up the training of new models. Yet, the efficient sharing, exploitation and reusability of this knowledge remains a challenge. Motivated from the EPT algorithm where neural networks are treated as dynamic entities, and are frequently modified to enable an efficient NAS algorithm, this chapter introduces a framework for deep learning models that facilitates the reuse of model architectures in an adaptivity-centric viewpoint. The framework has capability to transfer coefficients between models for knowledge composition and updates, and to apply compression and pruning techniques for efficient storage and communication. We discuss the framework and its application in the context of Knowledge Centric Networking (KCN) and demonstrate the framework potential through various experiments, i.e. when knowledge has to be updated to accommodate new (raw) data or to reduce complexity. In terms of adaptivity, this framework assists the neural networks based applications to be useful beyond just one operation. It can provide the support needed for longevity of existing and deployed deep learning models at the edge.

This Chapter is based on:

- **D. Sapra** and A. D. Pimentel *"Deep learning model reuse and composition in knowledge centric networking"* [138], in *Proceedings of the International Conference on Computer Communications and Networks*, © IEEE.

7.1 INTRODUCTION

In the age of the Internet of Things (IoT), where we have a complex network of connected devices, sensors and computing units, there is a large amount of data churning up every minute. As these networks grow with more devices and users, the rapid growth of data can overwhelm the underlying communication channels and resources. The data can be highly redundant and obtaining data might not be the end objective in such networks. Converting data to knowledge allows utilization in beneficent ways for different tasks, such as visual monitoring in smart homes, remote assistance in medical care or analyzing environmental data for agriculture. Machine learning is increasingly being deployed to convert raw data into meaningful knowledge. This conversion is majorly performed on a central server or knowledge creator node with high computation capabilities, which can create bottlenecks with high volumes of data communication. To overcome these issues, Knowledge Centric Networking (KCN) was conceptualized in [139], which proposes a paradigm shift, in a network, from data centric communication to knowledge centric communication. KCN emphasizes three key aspects about knowledge: creation, composition and distribution.

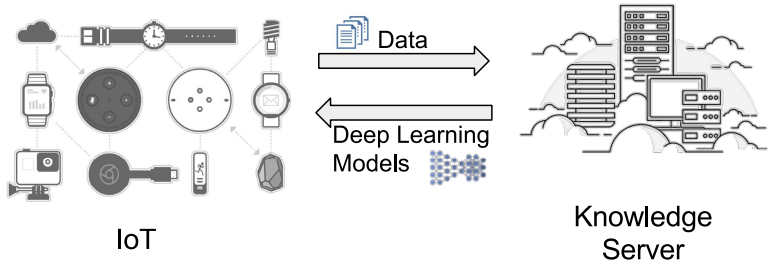


Figure 7.1: Traditional IoT network with centralized knowledge server for intelligence. Deep learning models are trained centrally using data from devices and then distributed.

Deep learning models or Neural Networks (NNs) are a popular knowledge modality for big data, storing the knowledge in the form of a brain-like architecture and thousands to millions of coefficients, which are trained from a large amount of data. They can be viewed as assimilating and extracting knowledge by storing data statistics and domain specific characteristics through training. Figure 7.1 illustrates how IoT networks and deep learning models are deployed and used in the traditional sense. There is a central knowledge server that creates deep learning models from de-

vice data, analyzes new incoming data and decides when to update the knowledge model. This knowledge server frequently distributes appropriate neural networks to all devices, in order to promote the use of better and more intelligent applications. KCN visualizes the knowledge creation at the edge wherever possible, and these deep learning models serve as the basic communication paradigm. New dynamic data is continuously sensed and devices are added to the network all the time. Sharing the assimilated knowledge models, building newer knowledge models on top of existing ones, and distributing decision making capabilities are aspired as this would make the network more resilient and adaptable to changes over the course of its active lifetime.

In such a distributed intelligent network, there is a need for frequent knowledge updates, which leads to the demand of an adaptive nature of the application over a longer period of time. From the work presented in first part of this thesis, it has already been observed that neural networks are dynamic entities. Their architectures are continuously modified during the search to traverse a large search space of neural architectures. The same rationale of treating a neural network as a dynamic model can be extended to the KCN environment.

In a domain specific application, the high correlation of neural networks can be exploited by different devices to reuse and combine each other's knowledge to create an adaptive application. Standardizing the knowledge update process with the aim of reusing this high correlation, motivates the creation of our framework. Our framework coordinates efficient communication, exchange and update of deep learning models, while being adaptable to accommodate new data being generated and insights learnt.

We differ from domain adaptation [140], knowledge distillation [141], and similar teacher-student algorithms [142], which attempt to create new models for similar tasks in different domains or constraints. In essence, these techniques can be used with our framework to simplify the creation of student models and streamlining their distribution through the network.

In short, our framework facilitates creation of a new network and modification of existing ones, allows combination of multiple models to compose a new model, replaces part(s) of a model with other sub-model(s), isolates model layers that can be individually transferred while supporting packaging and compression for distribution. There are multitude of ways in which deep learning models can be modified, both weights and architecture of the neural network can be updated, e.g. add/prune layers, add residual connections, change layer activation.

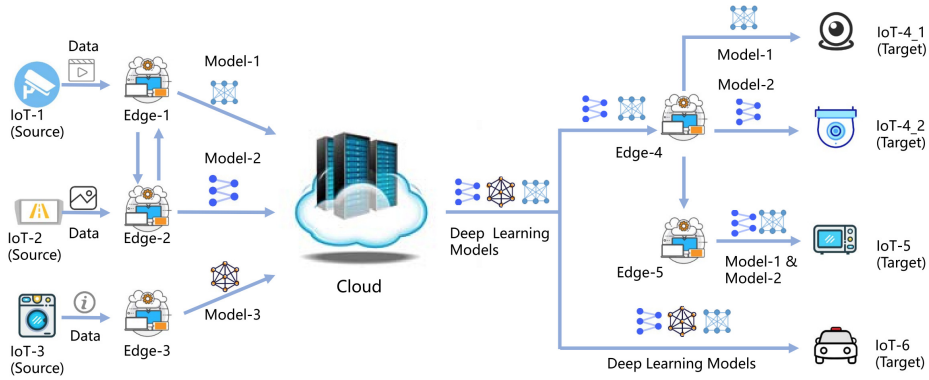


Figure 7.2: KCN paradigm on IoT network with edge computing. Deep learning models can be created on edge and transmitted from edge to edge, edge to cloud as well as edge to IoT device [143].

All of these tasks are unrelated to the training of models or knowledge creation directly, even so, these tasks are needed to keep the network's knowledge contemporary and maintain communication brevity with frequent updates in the context of KCN. Applied properly, these tasks are necessary to create an adaptable environment, where neural networks can be modified to suit the changing needs over a long period of time.

This chapter introduces a framework for neural networks to facilitate knowledge exchange, reuse and frequent updates. Additionally, we discuss various deep learning model update techniques from existing literature and formulate them in the context of Knowledge Centric Networking for adaptive environments. The framework is validated with experiments to update CNNs by varying data information over time as well as model complexity.

7.2 KNOWLEDGE CENTRIC NETWORKING (KCN)

In traditional IoT networks with cloud computing support, all devices transfer the data to the cloud and delegate heavy computational tasks to the central server. Data then converge and are progressively used for model training and fine tuning. This approach implicates heavy data transmission and storage requirements on the network. The huge amount of data threatening to throttle the network has prompted various research ideas, e.g. data compression and quantization techniques [144, 145]. Despite very efficient data compression techniques, the network still can be overwhelmed to meet the demand for high efficiency with highly redundant data transmissions. A

novel concept of KCN was envisioned and proposed in [139], which emphasises the communication of knowledge models instead of raw data. IoT devices continuously sensing the environment and generating huge amounts of temporal data, can use additional edge devices, which are closer to them or maybe embedded along with sensors in the IoT device, for knowledge extraction. KCN is based on the Edge computing paradigm [146], which promotes more computation on edge devices and less on the cloud, thereby reducing costs for data bandwidth and storage. Figure 7.2 illustrates this concept: deep learning models can be created and updated at the edge and transmitted to other IoT devices, edge devices or cloud. Besides improving latency and scalability of the system, KCN also reduces exposure to privacy and security attacks by removing the private or sensitive data moving around in the connected network.

This network allows a different granularity and hierarchy of knowledge creation, re-composition, update and exchange. Edge nodes can extract local knowledge from the sensed data and then upload to the cloud. The cloud server collects models from different devices and can perform composite knowledge updates. It can then further distribute the updated models to interactive and decision making devices, which might be considered as front-ends of the system. Edge nodes can also request generated models at another edge node directly to perform its own task efficiently. This results in frequent knowledge communication in the network as well as recurring updates to knowledge models stored at different devices.

With significant recent advances in deep learning models and deployment of more and more IoT devices, it is probable that KCN will become the key to control the *too much data* problem. The work in this chapter attempts to take a step towards treating neural networks as dynamic entities, which can be updated and adapted to the changing environment, as and when required. It is important to note that the techniques discussed in this chapter are not limited to a KCN based IoT network. They can be applied in any situation that warrants an adaptive deep learning based application.

7.3 RELATED WORK

The concept of incorporating knowledge based communication into networking is not new. There have been numerous works proposing the idea, such as [147–149]. However, the focus of these works is towards network communication analysis and control, while largely ignoring the utilization of the knowledge modality in all other aspects of an IoT network. In direct

contrast, KCN perceives the knowledge as the chief content operating in the network, from front-end to back-end and from sensing to action. Knowledge creation, composition and distribution are the primary expected features of every device on the network.

Some works proposed in recent years investigate various different aspects of the knowledge centric paradigm to further the research on KCN infrastructure. For knowledge distribution in KCN, [143] investigated inter-model compression for compact representation of models to further reduce the data bandwidth requirements. Our works are similar in respect that we believe interoperability is the forefront of knowledge communication and exchange, but diverge very quickly in the scope they are investigated. Our focus is on knowledge modification as opposed to knowledge compression for distribution. Deep learning model compression is a small part of our framework whose aim is to allow coefficients isolation and efficient packaging in situations where only some parts of the model need to be exchanged.

Foreseeing the problem of the knowledge-based forwarding on the knowledge router, [150] proposed a novel data structure for the knowledge routing table index. This results in faster movement of knowledge on a physical network and its efficient routing in terms of shorter latency, low memory consumption, and fast routing table update. Similarly, [151] proposed an intelligent routing algorithm for knowledge models in KCN based vehicular ad hoc networks. These works, though dissimilar to ours, are vital to achieve KCN deployment in physical IoT networks. These works strengthen the viability of a KCN infrastructure implementation and in turn makes our framework more practical and serviceable in a KCN based system.

The concepts of lifelong learning and never-ending learning [152, 153] have also been around since a while. These ideas focus on individual models to be better learners of a variety of data types *by being able to learn how to learn*, and thus keep evolving with the evolving environment. Even though the concepts have been proposed for individual models, they can also be reformulated for the KCN paradigm where the dynamic intelligence of the network is able to improve implicitly over the course of time.

The implementation of systems that will last a long time is also possible through progressive learning methodologies for neural networks such as for multi-class classification [154], face recognition [155] and Speech Enhancement[156]. In progressive learning, the neural network starts learning from a small set of data but can expand automatically on introduction of new classes while still retaining the knowledge of previous classes. Unlike our framework, these techniques let the model to grow with each

update and do not perform model reduction or consider power-memory constraints that are common with devices in the IoT network.

7.4 FRAMEWORK DESIGN

In this section, the framework and its features are introduced, while discussing the design choices that we contemplate to work best in different situations that can occur in knowledge-based IoT networks. There is a plethora of algorithms and techniques available in the literature that modify a neural network in different possible ways. We draw some techniques from this pool for their suitability to KCN. We not only describe different forms of model modifications in this section but also envision how and where they can be used for efficient knowledge exchange. We specify which techniques are supported by our framework and we try to point out their applicability and possible use case settings. The novelty of this framework lies in the detailed study and consolidation of deep learning modification and communication techniques applicable to a progressive learning and frequent update paradigm. We recognize that some of our work is about picturing various situations in the context of KCN and building a framework around it. We believe that formulating knowledge composition, update and exchange methodologies will cater to the dynamic IoT network to be better serviced for a longer period of time and it is a step forward towards moving the KCN paradigm from a vision to reality.

7.4.1 *Coefficients Update*

The simplest form of knowledge update occurs in a neural network by training the existing model with more data. The model architecture remains exactly the same, yet all coefficients get updated to reflect knowledge from new data. Our framework implicitly handles the coefficients update by updating the model file with new weight values after training. Training with new data is done at the edge and it is expected that this operation is recurrent in nature on most, if not all, edge devices. With the knowledge model being the important storage and communication entity, the data is expected to be discarded after training. The frequency of training is driven by storage capacities at the edge devices and training can be triggered with sufficient accumulated data. Depending on data type and format, there may be a data cleanup and pre-processing pipeline in place before the training oper-

ation. The updated model is then available to be distributed or exchanged through the network as needed.

7.4.2 *Architecture Update*

Designing a neural network is not trivial, and not all architectures are equal in terms of their capacities and knowledge representation capabilities. Deeper networks, with a higher number of layers, allow a more complex and non-linear function to be learnt from data. However, apart from needing huge amounts of training data to be able to meaningfully learn, they require a large memory and powerful computation units, usually lacking in edge devices.

There exists a performance-resource trade off in embedded and low power systems, which is usually dissimilar for different types of devices. While exchanging neural networks between different nodes, there is a chance that at the receiving device, the model is too large or demands more computation resources than available, setting off a need to reduce the model complexity. In some other cases, the initial data available is not enough to train a large model, so a small model may be built to kick start the knowledge extraction process which also prevents over-fitting on the small data-set at the same time. A larger model is built later on to fully utilize all data that has been sensed, which can be fabricated by expanding the current model capacity instead of training from scratch. This progressive expansion is also expected to reduce the upfront overhead of computational costs that training a large model entails.

In all feasible and available possibilities for architecture update, it is assumed vital for a model to be updated in a function preserving fashion, so that the model does not leak its knowledge. All algorithms implemented in the framework have been chosen to consider either the function preservation or minimal loss possible. Some of them were motivated by the genetic operators in evolutionary neural architecture search (Chapter 3), where function preservation is crucial. The small loss of performance with a major update is expected to be gained back by training more and more as new data keep arriving around the clock.

Our framework handles architecture updates by considering each layer as a named node and storing its coefficients as a separate but connected block. Each node can be isolated and its parameters, coefficients and formats updated individually. It automatically checks for data format changes to be done in subsequent layers of the modified node to keep the model valid

and consistent. The named nodes are important and should be unique to be able to be used as identifiers for all the update and modification operations.

Increasing the model capacity

Out of all possible ways to increase the model's capacity, our framework currently handles two ways to increase the capacity of the model: increase the number of layers of the neural network or increase the number of units or neurons in each or some of the layers. Our framework emulates Net2Net [65] for function preserving expansion of the network. There are two operations available in Net2Net: Net2Deeper and Net2Wider, to increase neural network depth and layer size respectively. Any convolutional or fully connected layer is replaced by two layers of the same type and size, with one layer having the same coefficients as the original layer and the new layer's coefficient matrix as initialized to an identity matrix. To increase the number of units in each layer, the coefficient matrix is expanded to the required size and then random layer units are selected and duplicated in the expanded coefficient matrix. Any increase in number of units causes the output of the layer to increase by the same amount, which means that there is a parallel increase in the number of inputs to the subsequent layer. This causes the coefficient matrix to expand as well, which is then appended with randomly initialized values. These new layers and units remain free to train further to take on any value later. Hence, the effect of increasing layers and units in this way is only to provide a good initialization to the newly created snippets in the knowledge. With further training on the data, the coefficients get updated to preserve any new knowledge and slowly diverge from their initial values.

Decreasing the model capacity

Contrary to the capacity increasing operations, it is not beneficial to delete layers or sub-units of the model in an ad-hoc manner. Each layer of the model holds information or features that subsequent layers use to build their own sub-knowledge. Various pruning techniques have been proposed in the last decades to reduce model complexity, redundancy and overfitting. In recent works, [157] and [158] suggested to remove all connections whose weight was lower than a threshold and retrain the rest of the network to fine-tune the model again. This approach leads to a reduction in model size to the tune of $9\times - 13\times$ without any loss of accuracy but leads to lightly populated coefficient matrices. This reduces memory footprint of the model but fails to reduce the computation cost because of irregular sparsity in the

pruned network. To overcome this issue, our framework emulates the *independent pruning* technique from [159]. This approach prunes the layer size by removing the least important units of a layer. The relative importance of a unit in each layer is calculated by the sum of its absolute weights. This value gives an expectation of the magnitude of the output of each unit, also called a feature map. Feature maps with smaller weights tend to produce outputs with weak activations when compared to the other units in that layer. Based on the target reduction size, the units with the smallest sum values and their corresponding feature maps are removed from the model. To regain the accuracy that was lost by pruning, a prune-then-retrain strategy needs to be adopted. In our framework, we prune filters of multiple layers at once followed by further training, though it is also possible to iteratively prune small slices of the model and retrain the network repeatedly. The iterative process may yield better results, but it requires more data and training epochs to reach original performance.

Another way in which our framework decreases the network capacity is by performing quantization on the coefficient values. Quantization refers to the process of reducing the number of bits that represent a number. In the context of deep learning for research and deployment, the predominant numerical format used has been 32-bits floating point. Our framework currently converts 32-bits floating point numbers to 16-bits, thereby halving the memory requirements for the model on devices. As shown by many research works, replacement of high precision numbers by lower-precision numerical formats can be done without incurring significant loss in accuracy [160, 161], thus preserving most of the knowledge encapsulated in the converted model.

Replacing Layers

Our framework allows for replacing a layer block in a model with a layer block from another model, which was trained on similar data but on another device. For this replacement to work, the input and output format of the switching block have to be same in both source and target models while the size of the block can be different. Figure 7.3 illustrates the layer replacing approach in our framework. If model complexity reduction is desired, a block of n layers is replaced by a block of layers of size $<n$ from the source model, given the layers are at the same cluster position as defined in Chapter 3, so that the input and output formats of the layer blocks are identical. The inverse action is also possible to increase model capacity. This situation in KCN is plausible when multiple knowledge models in the IoT

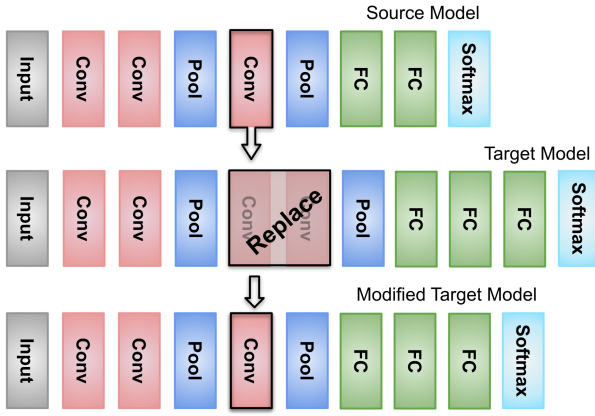


Figure 7.3: Example of layer replacement. A layer from the source model replaces a similar layer section in the target model to achieve a new model containing fewer layers.

network are trained on same data but resulted in different deep learning models based on each device's own optimization for model creation and composition over the course of time. Accuracy lost by performing this action can be regained by retraining with more data, though sometimes the new model might never be as proficient as the old one.

The model complexity of neural networks can also be reduced by replacing a fully connected layer of a convolutional neural network with the Global Average Pooling layer. The network in Network architecture [162] and GoogLenet [163] achieve state-of-the-art results on several benchmarks by adopting this idea. Within our framework, replacing layers is achieved by creating a new architecture for the model, which attempts to copy all its coefficients from the old model. Coefficients for nodes which cannot be found in the source model are then randomly initialized, these are essentially now the new layers of our model.

7.4.3 *Activation function update*

Our framework allows changes to other parameters such as layer activations, drop-out units and normalization techniques as well. Activation functions are used for introducing non-linearity into the neural network model so that the network can progressively learn more effective feature representations. Rectified Linear Units(ReLU)[164–166] are the most popular activations as deep networks with ReLUs are more easily optimized than

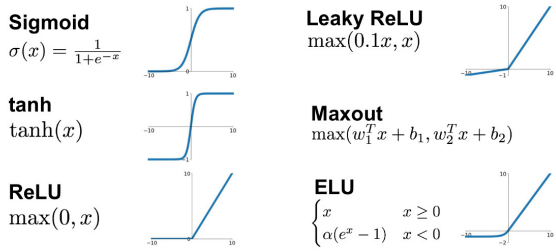


Figure 7.4: Popular activation functions for neural networks.

networks with *sigmoid* or *tanh* units [167]. Figure 7.4 shows some popular activation functions.

Changing an activation function in the neural network has a huge impact on the model behavior and changing them for an already trained model is not a good idea. However, changing the activation function from related functions is desirable in some cases, such as using Leaky-ReLU will avoid the *dead ReLU problem* which happens when the ReLU activation always have values under 0, which completely blocks further learning. Concatenated RelU[168] and Parametric Rectified Linear Unit (PReLU) [169] are proposed to reduce redundancy and better generalize the traditional ReLU. In principle, changing the activation function is carried out in our framework by changing node parameters. However, the more disparate and dissimilar activation functions are, the more knowledge is lost in the process.

7.4.4 *Multi-source knowledge fusion*

Our framework facilitates fusion of multiple neural networks to assimilate knowledge from different sources into a larger model encapsulating *the bigger picture*. By using the same layer node names as source nodes, the fused model represents replicas of the parts of different models joined together within the new model. After the new model is created, it looks for associated source layer coefficients to be copied for later computation.

This feature is useful in IoT networks with temporal sensing such as with Human Activity Recognition (HAR) [18, 170, 171], which is a vital step in an application, such as skill assessment and smart home assistant. Each sensor has its own knowledge extraction module to analyze and obtain the local salient features from the data. This knowledge is pooled centrally and processed to extract inter-sensor dependencies and detect associated human activity in the given time period. Figure 7.5 illustrates the concept

of multi-source knowledge fusion with deep learning models for time series input sensors.

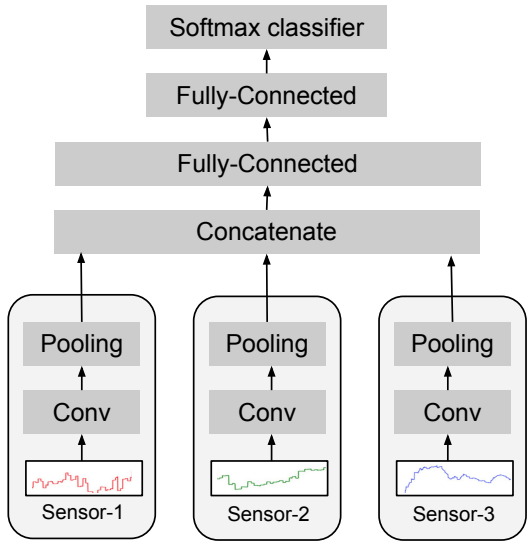


Figure 7.5: Knowledge fusion model with different sensors in Human Activity Recognition (HAR). Each time series data from sensors is locally processed and fused together at a central node to extract inter-sensor relations to detect human activity [170].

In the context of KCN, this kind of model will have difficulties to start knowledge extraction locally because all the sensor data needs to be trained together to extract knowledge about both independent and inter-dependent features for a meaningful activity recognition. All sensors might be served by same edge node to collect data and start training the model. Once sufficient performance is reached, the model parts might be isolated and distributed to relevant sensors. The sensors can then convert their local raw data into knowledge using their own sub-model and send their knowledge to the same edge node again to be fused with knowledge from other nodes. Our framework allows layer isolation and model fusion to seamlessly execute this possible workflow.

Another example of knowledge fusion, called multi-source transfer learning [172, 173] is popular in medical image analysis. It is based on an ensemble of models that are each created using single source transfer learning from a variety of domains with similar data characteristics. In single source transfer learning a number of consecutive layers are transferred from a chosen pre-trained model (teacher) to initialize its counterpart target model

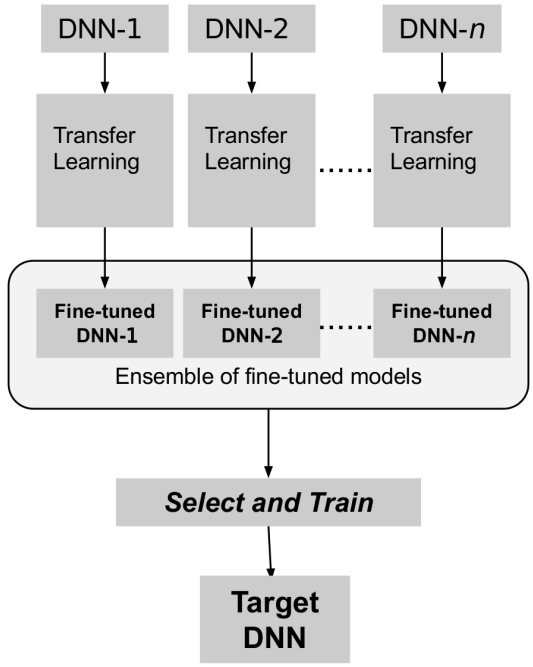


Figure 7.6: Knowledge fusion model with multi-source transfer learning using an ensemble of deep models from different sensors/devices. Knowledge is transferred from each source database to a target model. A selection process combines models into an ensemble that is used to train a single randomly initialized neural network.

(student). The rest of the student model is created anew and randomly initialized. The layers obtained from the teacher are usually frozen and the student model is trained on the target data-set to be fine-tuned for the intended domain.

In our framework, single source transfer learning is realized by duplicating the source model, freezing the coefficients of appropriate layers and then replacing the rest of the layers with new randomly initialized replacement layers. After re-training and fine tuning for the target task, the new models can be saved at the central server.

An ensemble of all of the student models is then used to train another model which is essentially now being generated by training from a knowledge ensemble instead of raw data. Figure 7.6 shows the multi-source transfer learning methodology. In KCN, this paradigm is very useful when a new device is added to the network, allowing it to gain knowledge from already available intelligence in the network and does not need to wait for a lot of

data to be collected before being fully utilized and deployed in its intended task. Our framework does not perform this training but facilitates the creation of ensembles of relevant student models. Using the ONNX file format, any available or desired deep learning framework can be used to train from the model ensemble.

7.4.5 *Isolation and compression*

As previously mentioned, our framework is capable of isolating layers and their coefficients for all types of modifications done on the models. Compression and packaging of raw coefficients data are vital for efficient transmission and exchange throughout the system, especially when distributing only a part of the model. We implemented basic support for general purpose compression and decompression using the popular ZLIB compression library [174]. But, other compression libraries can also be added to extend our framework.

7.5 VALIDATIONS

In this section, we validate the framework and its viability to act as a knowledge modification and update environment. We present the framework setup followed by two use cases, a smart camera network and a multi-sensor network, simulating the KCN environment. We discuss different use cases for the framework within these use cases.

We implemented the framework using Java 8 and the Protocol Buffers (protobuf) library [175] to build the ONNX components. Our framework reads ONNX files and then alters them as per the requested use case and writes the updated ONNX files onto the storage system again. For further training and validation, we imported the ONNX files into the Python based Caffe2 framework (from the Pytorch library) [33]. Our framework itself is reasonably lightweight and runs without any GPU support, though we utilized a GeForce RTX 2080[176] GPU to train all the deep models.

7.5.1 *Smart camera network for object recognition*

We simulated a KCN environment for a smart camera network for object recognition as outlined in various other frameworks as well [177–179] (see Figure 7.7). In the simulated environment, each camera is represented by a unique convolutional model called a *cam-model*. In other words, each *cam-*

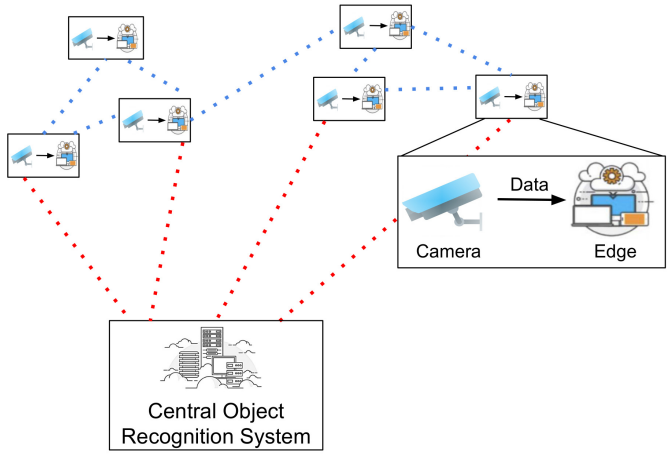


Figure 7.7: Smart camera network with a central knowledge server.

model in the environment appears for a virtual camera and defines its object recognition capabilities. In the experimental setup, we trained *cam-models* on CIFAR-10 [22], which is a popular object recognition dataset. CIFAR-10 consists of 32×32 pixels RGB images classified into 10 categories and is further divided into two sets of 50000 training and 10000 test images, to train and validate the model respectively.

A *cam-model* is an assembly of multiple convolutional layers interspersed with two maxpool layers for input size reduction, followed by fully connected layers. The convolutional layers are also termed as feature extractor, and a series of fully connected layers, work as a classifier, to correctly classify the image and provide a label.

Figure 7.8 illustrates a basic neural network structure used in the current setup. Each *cam-model* has its own distinct topology where the number of convolutional and fully connected layers as well as layer parameters such as

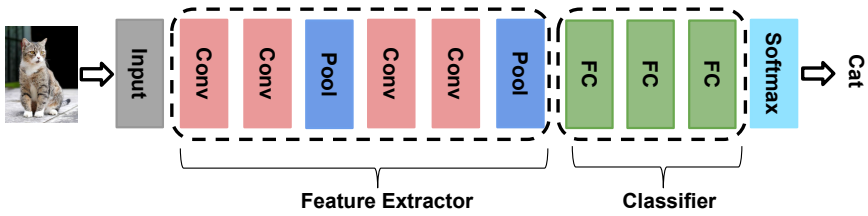


Figure 7.8: Deep neural network depicting convolutional feature extractor and fully connected classifier.

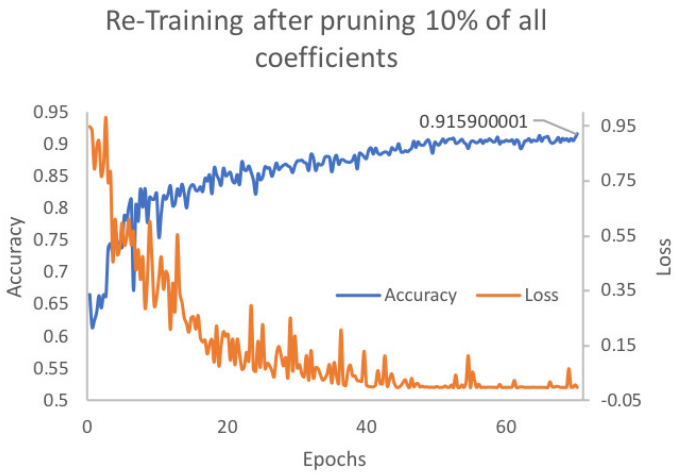


Figure 7.9: Training curves for pruned model by 10%

the number of units and kernel sizes are randomly sampled taking the predefined constraints into account. This is to reflect that in a real smart camera network, cameras added over a long period of time will have different local compute and storage capabilities and might have been initialized with a distinctive deep learning model.

All models were built with ReLU activations and were trained (and retrained) with a learning rate of 0.0005 and batch size of 90 using the Adam optimizer. We restricted the GPU memory usage to 5GB during training to limit the size of Neural Network from becoming too large. It is assumed that labelled data is available and accessible by relevant devices or the server to perform these tasks. We discuss some use cases for our framework below:

Pruning a model by 10%

Memory is usually limited on an edge device and therefore, pruning is a very efficient technique to reduce the model's memory footprint. To demonstrate that performance is not degraded when removing redundant information, we performed this experiment on a *cam-model* comprising of 13 convolutional layers and 3 fully connected layers having a total of ≈10 million coefficients.

We pruned the network to reduce all weights by 10%, resulting in a new model with ≈9 million coefficients. The resulting model uses less mem-

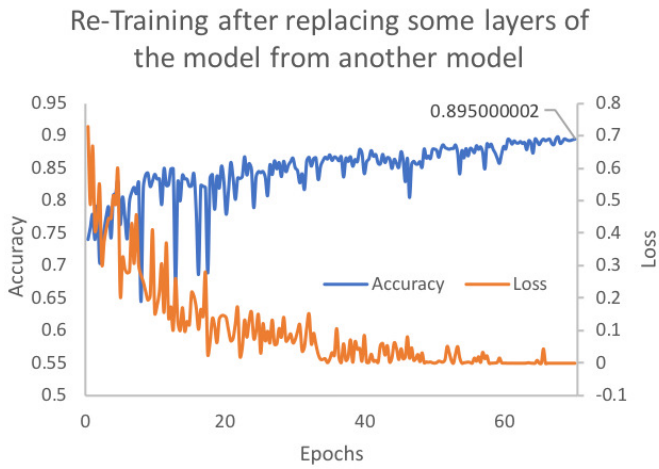


Figure 7.10: Training curves for modified model with some convolutional layers replaced from another pre-trained model

ory while also reducing the number of MAC (Multiply-Accumulate) operations needed in each run. The original model has 91.4% accuracy while the pruned model actually performed slightly better with 91.59% accuracy after re-training, see Figure 7.9. The increase in performance is caused by weak activations removal which were not contributing considerably to the model intelligence.

Model composition from two neural networks

For this use case, we selected a *cam-model*, with the aim of reducing the storage size of the model on the edge. It has 10 wide convolutional layers and 3 fully connected layers ($\approx$ 19 million coefficients) with test accuracy of 90.81%. We picked another *cam-model* with 13 (smaller) convolutional and 2 fully connected layers ($\approx$ 12 million coefficients) having an accuracy of 92.09%.

We chose a block of 2 convolutional layers from the latter model and used it to replace a block of 4 convolutional layers in the first one. All the blocks that were selected were operating on same input and output dimensions and were roughly in a similar phase of feature extraction.

The resulting model now has 8 convolutional layers and 3 fully connected layers with $\approx$ 14 million coefficients. We re-trained it further and the new model was able to achieve 89.5% accuracy, which is much less than its

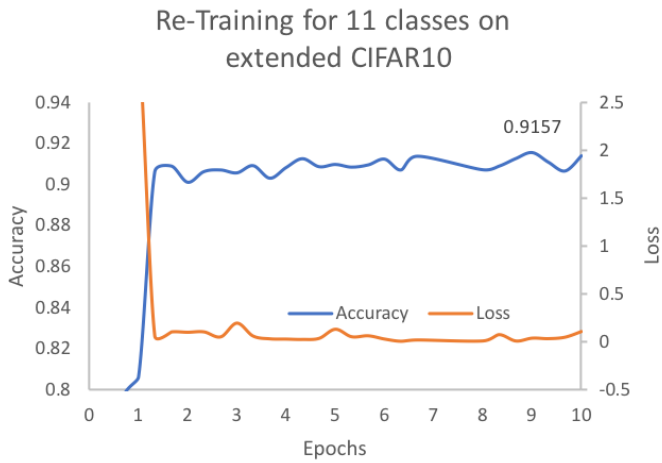


Figure 7.11: Training curves for modified pre-trained model to include a new class "Flower" to the existing CIFAR-10 dataset.

parent. Figure 7.10 shows the related training curves. This illustrates the point that composing a model from two different models is not always a preserving function, however the benefit is still observed by lowering the model complexity through a reduction in the number of layers, amount of arithmetic computations and storage size.

Increasing the number of output classes

As mentioned above, there are 10 output classes for the CIFAR-10 dataset. There are possible situations where sensed data or the environment has evolved and there is a need to define an extra output class. It is not desirable to train from scratch, especially when original data were not saved in the system. We added a class "Flower" to an existing, pre-trained model on CIFAR-10. The number of images available for the new class were kept at half of existing class samples to reflect the fact that in a dynamic environment new output classes will not be equally represented, specially in the early stages of new data being sensed. To fine tune the model, we preserved the feature extractor and expanded the last fully connected layer in the classifier to include new output, totalling the number of outputs to 11. The new model was fine-tuned by training with an input data set containing all 11 classes, though only the last few layers were available to be updated as we froze the feature extractor.

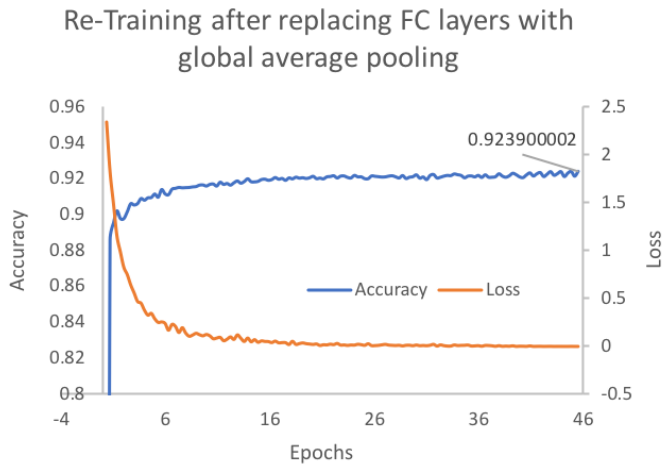


Figure 7.12: Training curves for modified pre-trained model with Global Average Pooling as classifier, replacing fully connected layers.

The randomly chosen *cam-model*, consists of 15 convolutional layers and 3 fully connected layers with 92.6% accuracy (with 10 classes). After only 10 epochs training, the reached accuracy for the extended CIFAR-10 dataset is 91.57%, see Figure 7.11.

We did not observe an accuracy increase after 10 epochs. And we noticed that there is a loss of performance, but handling the trade-off between speed of knowledge update and best performance achievable is a complex task to fulfill. The trade-off will generally vary with different data types, model complexities and device computation capabilities. As future work, the decision choices regarding how long to train and how many layers to freeze might also be dynamically incorporated into the framework to get the best model in terms of its performance when data characteristics change over a long period of time.

Replacing layers

Replacing *heavy* fully connected layers with Global Average Pooling decreases the number of parameters of the model, and reduces the computation cost. We took the same *cam-model* as above (with 10 output classes) and replaced the fully connected layers based classifier with a Global Average Pooling layer. In the given network it resulted in $\approx$ 1.5 million fewer multiply-accumulate operations, which leads to faster inference, along with lower power consumption, computation and memory demands on the device.

We re-trained the new network until the loss became stagnant, which was at 45 epochs. The original model accuracy is 92.6%, and even though the new model has fewer coefficients, it displayed a very small performance degradation by reaching an accuracy of 92.39%, see Figure 7.12.

The last two graphs (Figure 7.11 and Figure 7.12) are noticeably smoother than the first two because the feature extractor was frozen in these two experiments and only the classifier part of the network was actually re-trained.

7.5.2 *Multi-sensor based activity recognition*

This case demonstrates the instances based on multi-source knowledge fusion. We performed an activity recognition task based on the PAMAP2 dataset [24], which provides data recordings from four sensors, 13 channels each from three Inertial Measurement Units (IMU) and a single channel from a heart rate monitor. All these sensors are body worn and are on distinct locations such as hand, chest and ankle, jointly forming a small network which also involves communicating sensor data to a central controller which recognizes the activity being performed. The setup in our experiment is based on the CNN-IMU architecture from [73] as shown in Figure 7.13.

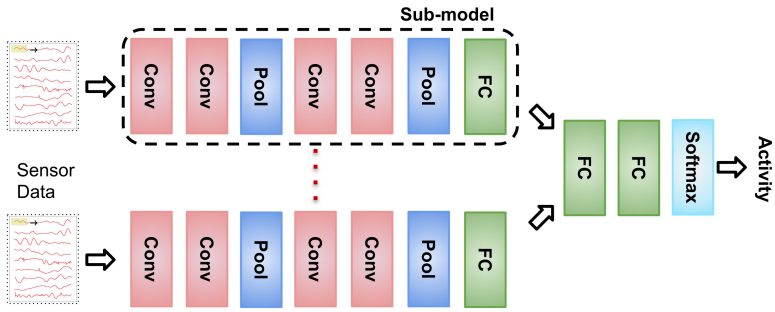


Figure 7.13: Activity Recognition based on multiple sensor data. Each sensor has a convolutional sub-model, fused at the end with fully connected layers.

Each sensor has its own branch of four convolutional layers intermixed with two maxpool layers, followed by a fully connected layer. The output from these branches is concatenated and goes through fully connected layers that predict the activity. The whole network is trained together at once with the RMSProp optimizer using a batch size of 50 and learning rate of 10^{-4} after downsampling the IMUs' recordings to 30 Hz and a sliding window of 3s (100 samples) and a step size of 660ms (22 samples).

Isolating branches Using our framework, we isolated sub models from the branches and their respective layers into individual ONNX files. The sub model can be deployed close to the sensor itself, hence removing the need for sensors to transmit all the data to the central controller. For a 13 channel IMU, data needed to analyse each 3s window in our setup is equivalent to approximately 8kB/s. By computing the sub models close to the sensor, only the output of the last layer is sent over to the central server, which is approximately 3kB/s, resulting in a 60% decrease of bandwidth requirement. The bandwidth saving becomes more important when there are many sensors in the network.

7.6 SUMMARY

In this chapter, we introduced a framework for deep learning models that facilitates knowledge update, composition and reuse in the scope of KCN. We emphasized that our framework can be used to allow neural networks in the context of IoT, for dynamic knowledge modality. We envisioned multiple possible situations where neural networks will need to be remodeled to suit evolving intelligence of the system and discussed ways to solve some of those challenges with an appropriate methodology. We also demonstrated with evaluations that our framework is able to update models in a variety of circumstances that are likely to occur in KCN. We showed that it is possible to use neural networks as a dynamic knowledge modality, which can be continuously modified and maintained in line with dynamic system behaviors and changing requirements.

8

CONCLUSION

This PhD thesis presented research performed by the author towards efficient neural architectures for resource-constrained edge devices. Neural network deployment triggers a never-ending demand for resources, and when deployed at the edge, these resources are perpetually in short supply. Neural networks demand generous computation capabilities, high memory and consume abundant energy in order to perform adequately. On the other hand, edge devices are low cost hardware and often run on an internal battery. Consequentially, there are constraints on resources available, such as memory, processing power, and energy.

Even with these challenges, neural networks operating on edge devices have many advantages. Hence, the first part of the thesis presented work for an efficient search methodology for a suitable CNN, which can thus operate within the constraints imposed by the target hardware. The second part of the thesis explored the subject of adaptivity in CNN-based application executing at the edge. Two different aspects of adaptivity were examined. The first one investigated adaptive switching of different deep-learning models based on the changes in the run-time environment of the application. Whereas the second one analysed neural networks as dynamic models and suggested techniques to keep them updated to reuse them over a longer period of their deployment lifetime.

The main contributions of this thesis, in terms of the frameworks proposed, are summarised below:

- The first framework proposed is called Evolutionary Piecemeal Training (EPT) to search for an efficient neural network architecture, which has highest possible accuracy within the architectural constraints. The constraints are placed to ensure that the discovered neural network can fit the resources available in the target hardware. Based on an evolutionary algorithm, it is subsequently extended to include mul-

tiple search objectives, namely Accuracy, Throughput, Memory, and Energy (ATME).

- The multi-objective algorithm forms the basis of the second framework: Scenario-Based Run-time Switching (SBRS). Which allows a CNN-based application to have environment triggered run-time switching between different neural networks, each having a unique ATME characteristic. This allows an application to be flexible and adaptable while executing on an edge device. This thesis presents scenario derivation in the SBRS framework. Additionally, but not as a part of this thesis, SBRS includes a combined Model of Computation (MoC) and a switching protocol.
- The third framework considers the versatility of a NN beyond one execution or one device. In a long active lifetime of an application, it is imperative that the NNs deployed need to be updated and reused. The framework facilitates knowledge exchange, reuse and frequent updates, by implementing various NN update techniques. These three frameworks together allow neural architectures to be adaptable and versatile for efficient utilization on the edge devices.

8.1 ANSWERS TO RESEARCH QUESTIONS

In this section we reflect on the research questions set forth in Chapter 1 and evaluate how this thesis addressed them.

RQ1: *How can we design an efficient NAS algorithm that reduces the search time, and has the capability to optimize for multiple objectives?*

To answer this question, we developed a Neural Architecture Search (NAS) methodology, called Evolutionary Piecemeal Training (EPT). Chapter 3 introduced the evolutionary based algorithm, which treated the NAS as an optimization problem. The search space of all neural networks in the algorithm had boundaries placed on the model size, so that the resulting CNNs never becomes too large to restrict their deployment on edge devices. Initially, the optimization objective of the EPT algorithm was only to maximize the test accuracy of the resulting CNN. Though the vision to extend the algorithm to include multiple objectives motivated the selection of an

evolutionary based algorithm for this task. Evolutionary algorithms already have a large body of research presented in the multi-objective optimization domain.

With most of the other evolutionary NAS algorithms, evaluating the accuracy of every CNN was a time consuming and resource hungry task, since it was performed after the full-training of a CNN. While most algorithms looked at training as an isolated and a separate task, the EPT algorithm examined the evolutionary algorithm from a different perspective, where the search for neural architectures was performed *during* a modified and extended training process instead. This method was able to converge in few GPU hours as opposed to tens of thousands of GPU hours needed by conventional evolutionary NAS algorithms. The algorithm performance was validated on the CIFAR-10 and the PAMAP2 datasets.

Further, the EPT algorithm was discussed in more details in Chapter 4. Defining the optimization objective as a *Gradually Saturating Objective Function* (GSOF), this chapter explained the challenges faced in designing the EPT algorithm and the adaptive diversity control based approach that worked during the design process. The proposed approach was validated with the PAMAP2 dataset.

An extension to the original EPT algorithm was presented in Chapter 5 for multi-objective NAS, which was validated by two sets of experiments. In the first set of experiments, there were two objectives incorporated in the algorithm, for the PAMAP2 dataset. Along with the prediction accuracy, the reduction of the number of parameters of the neural network was considered as an additional objective. For the second set of experiments, also termed as the hardware-aware EPT, the algorithm was extended to include hardware specific objectives, namely memory footprint, energy consumption and throughput. The hardware-aware EPT evaluated the performance of every CNN on a specific target hardware. This version of experiments was evaluated for the VOC, the CIFAR-10 and the PAMAP2 datasets for NVIDIA Jetson TX2 platform.

RQ2: *How can we ensure that a CNN-based application is able to efficiently adapt its extra-functional characteristics synchronously with the changes in its environment at run-time?*

A novel Scenario Based Run-time Switching methodology for CNN-based applications was proposed in Chapter 6. The extra-functional characteris-

tics of the CNN-based application can be captured by different scenarios, where each scenario has unique ATME (Accuracy, Throughput, Memory, Energy) requirements. Each scenario was associated with a CNN, which are derived from the hardware-aware EPT algorithm. During the application operation, an environmental change can trigger the application to switch between scenarios, thereby adapting its extra-functional characteristics with the changes to the environment. The algorithm was validated by designing three run-time adaptive applications for the VOC, the CIFAR-10 and the PAMAP2 datasets. In addition, the SBRS methodology was compared with the MSDNet methodology for the CIFAR-10 dataset. Both of the methodologies were executed on an NVIDIA Jetson TX2 device. When executed under tight hardware-aware requirements, the SBRS methodology outperformed the MSDNet methodology. The difference was probably caused by the fact that the SBRS methodology optimized the application in terms of both accuracy and hardware-aware characteristics, whereas the MSDNet only optimized accuracy.

RQ3: *Is it possible for neural network to be treated as a dynamic entity during its active lifetime? If so, how can we ensure that a CNN, deployed at the edge, can be regularly updated and maintained?*

This research question has two parts to it. The first question was partially answered by the first part of the thesis, where neural networks were treated as dynamic models by the EPT algorithm. The neural architectures are modified by the evolutionary operators, in each iteration, to cover a huge search space. The reason for frequent updates in the EPT algorithm is not adaptivity, however, the same rationale of treating a neural network as a dynamic model can be extended to a KCN based IoT environment.

The framework presented in Chapter 7 attempted to answer the second question, by expanding the ideas from the first part of the thesis and other existing literature. The framework suggested approaches to enable continuous deep learning model reuse and updates during a longer lifetime of a CNN-based application at the edge. Chapter 7 demonstrated, with evaluations, that the framework was able to update CNNs in a variety of circumstances that are likely to occur in KCN. Hence, we showed that neural networks can be utilised as a dynamic entity, which can be continuously modified and maintained in sync with dynamic environments.

8.2 FUTURE WORK

Revisiting the Chapter 2, where we discussed cell-based neural architectures. The preliminary exploratory work that was performed to examine their suitability towards edge devices demonstrated very promising results [17]. However, this work was not a full-fledged or in-depth analysis based on a complex NAS methodology. Hence, the first obvious future work is to extend the search spaces for the EPT algorithm to include cell-based architectures. The search for a cell-based architecture will also be interesting to evaluate from two separate fronts. The first one to discover an efficient cell structure itself and the second one to examine the placement of these cells.

Another interesting subsequent research direction will be to extend the SBRS methodology, where scenarios can share more parameters amongst them and thus allow for a faster switching. In its present form, many coefficients have to be swapped in the memory to switch scenarios. There can be a few approaches to carry out this task, such as a shared training for some of the common layers for the neural architectures derived through the hardware-aware EPT algorithm. Another approach can be to modify the EPT algorithm itself so that the architectures which have a common *ancestor* are trained jointly (similar to a joint training approach for neural network ensembles). Nevertheless, this problem has an intriguing research proposition for the future.

With significant recent advances in deep learning models and deployment of more and more IoT devices, it is probable that KCN will become the key to control the *too much data* problem. There are still some open problems that need to be solved in order to see KCN being deployed in reality. Specifically, knowledge creation and modification strategies that are geared towards very low-power embedded devices and real time constraints. These issues require further investigation and extension of our framework to integrate resource/performance trade-offs based model modification techniques and faster update mechanisms for real time requirements.

BIBLIOGRAPHY

[1] Mahbubul Alam, Manar D Samad, Lasitha Vidyaratne, Alexander Glandon, and Khan M Iftekharuddin. "Survey on deep neural networks in speech and vision systems." In: *Neurocomputing* 417 (2020).

[2] Daniel W Otter, Julian R Medina, and Jugal K Kalita. "A survey of the usages of deep learning for natural language processing." In: *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[3] Daniel Zhang, Saurabh Mishra, Erik Brynjolfsson, John Etchemendy, Deep Ganguli, Barbara Grosz, Terah Lyons, James Manyika, Juan Carlos Niebles, Michael Sellitto, et al. "The ai index 2021 annual report." In: *arXiv preprint arXiv:2103.06312* (2021).

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks." In: *Advances in Neural Information Processing Systems*. 2012.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

[6] Fjodor Van Veen. "The neural network zoo." In: *The Asimov Institute* (2016).

[7] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. "Overview of supervised learning." In: *The elements of statistical learning*. Springer, 2009.

[8] Jing Li, Ji-hang Cheng, Jing-yuan Shi, and Fei Huang. "Brief introduction of back propagation (BP) neural network algorithm and its improvement." In: *Advances in computer science and information engineering*. Springer, 2012.

[9] VNI Cisco. "Cisco visual networking index: Forecast and trends, 2017–2022." In: *White Paper* 1 (2018).

[10] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019.

[11] Jiemin Fang, Yuzhu Sun, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang. "Densely connected search space for more flexible neural architecture search." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020.

[12] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. "Understanding of a convolutional neural network." In: *2017 International Conference on Engineering and Technology (ICET)*. 2017.

[13] Léon Bottou. "Large-scale machine learning with stochastic gradient descent." In: *Proceedings of COMPSTAT*. 2010.

[14] Yao Shu, Wei Wang, and Shaofeng Cai. "Understanding Architectures Learnt by Cell-based Neural Architecture Search." In: *International Conference on Learning Representations*. 2020.

[15] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. "SNAS: stochastic neural architecture search." In: *arXiv preprint arXiv:1812.09926* (2018).

[16] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. "Neural architecture search: A survey." In: *Journal of Machine Learning Research* 20.55 (2019).

[17] Ilja van Ipenburg, Dolly Sapra, and Andy D Pimentel. "Exploring Cell-based Neural Architectures for Embedded Systems." In: *2nd International Workshop on IoT, Edge, and Mobile for Embedded Machine Learning*. 2021.

[18] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. "Deep learning for sensor-based activity recognition: A survey." In: *Pattern Recognition Letters* 119 (2019).

[19] Baoxiang Pan, Kuolin Hsu, Amir AghaKouchak, and Soroosh Sorooshian. "Improving Precipitation Estimation Using Convolutional Neural Network." In: *Water Resources Research* 55.3 (2019).

[20] Feng Ling, Doreen Boyd, Yong Ge, Giles M Foody, Xiaodong Li, Lihui Wang, Yihang Zhang, Lingfei Shi, Cheng Shang, Xinyan Li, et al. "Measuring River Wetted Width from Remotely Sensed Imagery at the Sub-pixel Scale with a Deep Convolutional Neural Network." In: *Water Resources Research* (2019).

[21] Andrew Davies, Stephen Serjeant, and Jane M Bromley. "Using Convolutional Neural Networks to identify Gravitational Lenses in Astronomical images." In: *Monthly Notices of the Royal Astronomical Society* (2019).

[22] Alex Krizhevsky, Geoffrey Hinton, et al. *Learning multiple layers of features from tiny images*. Tech. rep. Citeseer, 2009.

[23] M. Everingham et al. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html. 2012.

[24] Attila Reiss and Didier Stricker. "Introducing a new benchmarked dataset for activity monitoring." In: *2012 16th International Symposium on Wearable Computers*. 2012.

[25] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. "Maxout networks." In: *arXiv preprint arXiv:1302.4389* (2013).

[26] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. "Striving for simplicity: The all convolutional net." In: *arXiv preprint arXiv:1412.6806* (2014).

[27] Rui Xi, Mengshu Hou, Mingsheng Fu, Hong Qu, and Daibo Liu. "Deep dilated convolution on multimodality time series for human activity recognition." In: *2018 International Joint Conference on Neural Networks (IJCNN)*. 2018.

[28] Nils Y Hammerla, Shane Halloran, and Thomas Plötz. "Deep, convolutional, and recurrent models for human activity recognition using wearables." In: *arXiv preprint arXiv:1604.08880* (2016).

[29] László A Jeni, Jeffrey F Cohn, and Fernando De La Torre. "Facing imbalanced data–recommendations for the use of performance metrics." In: *2013 Humaine association conference on affective computing and intelligent interaction*. 2013.

[30] Jesse Davis and Mark Goadrich. "The relationship between Precision-Recall and ROC curves." In: *Proceedings of the 23rd international conference on Machine learning*. 2006.

[31] Takaya Saito and Marc Rehmsmeier. "The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets." In: *PloS one* 10.3 (2015).

[32] *ONNX: Open Neural Network Exchange Formet*. 2019. URL: https://onnx.ai/.

[33] *PyTorch: An open source deep learning platform*. 2019. URL: https://pytorch.org/.

[34] *Keras*. 2015. URL: https://keras.io.

[35] *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: http://tensorflow.org/.

[36] *Apache MXNet: A flexible and efficient library for deep learning*. 2019. URL: https://mxnet.apache.org.

[37] *The Microsoft Cognitive Toolkit*. 2019. URL: https://docs.microsoft.com/en-us/cognitive-toolkit/.

[38] *ONNX Runtime: cross-platform, high performance scoring engine for ML models*. 2019. URL: https://github.com/microsoft/onnxruntime.

[39] Dolly Sapra and Andy D Pimentel. "Constrained evolutionary piecemeal training to design convolutional neural networks." In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. 2020.

[40] Dolly Sapra and Andy D Pimentel. "Designing convolutional neural networks with constrained evolutionary piecemeal training." In: *Applied Intelligence* (2021).

[41] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. "Evolving deep neural networks." In: *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. 2019.

[42] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. "Regularized evolution for image classifier architecture search." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019.

[43] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. "Efficient Neural Architecture Search via Parameter Sharing." In: *International Conference on Machine Learning*. 2018.

[44] Barret Zoph and Quoc V Le. "Neural architecture search with reinforcement learning." In: *arXiv preprint arXiv:1611.01578* (2016).

[45] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database." In: *2009 IEEE conference on computer vision and pattern recognition*. 2009.

[46] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.

[47] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. "Densely connected convolutional networks." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

[48] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. "A genetic programming approach to designing convolutional neural network architectures." In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017.

[49] Yu-Dong Zhang, Suresh Chandra Satapathy, David S Guttery, Juan Manuel Górriz, and Shui-Hua Wang. "Improved breast cancer classification through combining graph convolutional network and convolutional neural network." In: *Information Processing & Management* 58.2 (2021).

[50] Tansel Dokeroglu, Ender Sevinc, Tayfun Kucukyilmaz, and Ahmet Cosar. "A survey on new generation metaheuristic algorithms." In: *Computers & Industrial Engineering* 137 (2019).

[51] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. "Large-scale evolution of image classifiers." In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. 2017.

[52] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. "Accelerating neural architecture search using performance prediction." In: *arXiv preprint arXiv:1705.10823* (2017).

[53] Zefeng Chen, Yuren Zhou, and Zhengxin Huang. "Auto-creation of Effective Neural Network Architecture by Evolutionary Algorithm and ResNet for Image Classification." In: *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. 2019.

[54] Lingxi Xie and Alan Yuille. "Genetic cnn." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017.

[55] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. "NSGA-NET: a multi-objective genetic algorithm for neural architecture search." In: *arXiv preprint arXiv:1810.03522* (2018).

[56] Junhao Huang, Weize Sun, and Lei Huang. "Deep neural networks compression learning based on multiobjective evolutionary algorithms." In: *Neurocomputing* 378 (2020).

[57] Francisco E Fernandes Jr and Gary G Yen. "Pruning Deep Convolutional Neural Networks Architectures with Evolution Strategy." In: *Information Sciences* 552 (2021).

[58] Marco Stang, Christopher Meier, Vinzenz Rau, and Eric Sax. "An evolutionary approach to hyper-parameter optimization of neural networks." In: *International Conference on Human Interaction and Emerging Technologies*. Springer. 2019.

[59] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. "Efficient Multi-Objective Neural Architecture Search via Lamarckian Evolution." In: *International Conference on Learning Representations*. 2019.

[60] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. "SMASH: one-shot model architecture search through hypernetworks." In: *arXiv preprint arXiv:1708.05344* (2017).

[61] Boyang Deng, Junjie Yan, and Dahua Lin. "Peephole: Predicting network performance before training." In: *arXiv preprint arXiv:1712.03351* (2017).

[62] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. "Learning transferable architectures for scalable image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.

[63] Hanxiao Liu, Karen Simonyan, and Yiming Yang. "Darts: Differentiable architecture search." In: *arXiv preprint arXiv:1806.09055* (2018).

[64] Han Cai, Ligeng Zhu, and Song Han. "ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware." In: *International Conference on Learning Representations*. 2019.

[65] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. "Net2net: Accelerating learning via knowledge transfer." In: *arXiv preprint arXiv:1511.05641* (2015).

[66] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. "Pruning filters for efficient convnets." In: *arXiv preprint arXiv:1608.08710* (2016).

[67] Dolly Sapra and Andy D Pimentel. "An evolutionary optimization algorithm for gradually saturating objective functions." In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2020.

[68] Li Yang and Abdallah Shami. "On hyperparameter optimization of machine learning algorithms: Theory and practice." In: *Neurocomputing* 415 (2020).

[69] *Jenetics library*. 2019. URL: https://jenetics.io/.

[70] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014).

[71] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. "mixup: Beyond empirical risk minimization." In: *arXiv preprint arXiv:1710.09412* (2017).

[72] Terrance DeVries and Graham W Taylor. "Improved regularization of convolutional neural networks with cutout." In: *arXiv preprint arXiv:1708.04552* (2017).

[73] Fernando Moya Rueda, René Grzeszick, Gernot Fink, Sascha Feldhorst, and Michael ten Hompel. "Convolutional neural networks for human activity recognition using body-worn sensors." In: *Informatics*. Vol. 5. 2. 2018.

[74] Christopher LE Swartz and Yoshiaki Kawajiri. "Design for dynamic operation-A review and new perspectives for an increasingly dynamic plant operating environment." In: *Computers & Chemical Engineering* (2019).

[75] Yaochu Jin and Jürgen Branke. "Evolutionary optimization in uncertain environments-a survey." In: *IEEE Transactions on evolutionary computation* 9.3 (2005).

[76] Krzysztof Trojanowski and Zbigniew Michalewicz. "Evolutionary optimization in non-stationary environments." In: *Journal of Computer Science & Technology* 1 (2000).

[77] Trung Thanh Nguyen, Shengxiang Yang, and Juergen Branke. "Evolutionary dynamic optimization: A survey of the state of the art." In: *Swarm and Evolutionary Computation* 6 (2012).

[78] Juan Zou, Qingya Li, Shengxiang Yang, Jinhua Zheng, Zhou Peng, and Tingrui Pei. "A dynamic multiobjective evolutionary algorithm based on a dynamic evolutionary environment model." In: *Swarm and evolutionary computation* 44 (2019).

[79] David Fagan and Michael O'Neill. "Exploring Target Change Related Fitness Reduction in the Moving Point Dynamic Environment." In: *International Conference on Theory and Practice of Natural Computing*. 2017.

[80] Shengxiang Yang. "Non-stationary problem optimization using the primal-dual genetic algorithm." In: *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.* Vol. 3. 2003.

[81] Anabela Simões and Ernesto Costa. "Memory-based CHC algorithms for the dynamic traveling salesman problem." In: *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. 2011.

[82] Aimin Zhou, Yaochu Jin, and Qingfu Zhang. "A population prediction strategy for evolutionary dynamic multiobjective optimization." In: *IEEE transactions on cybernetics* 44.1 (2013).

[83] Wee Tat Koo, Chi Keong Goh, and Kay Chen Tan. "A predictive gradient strategy for multiobjective evolutionary algorithms in a fast changing environment." In: *Memetic Computing* 2.2 (2010).

[84] Iason Hatzakis and David Wallace. "Dynamic multi-objective optimization with evolutionary algorithms: a forward-looking approach." In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. 2006.

[85] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves." In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015.

[86] Hongfeng Wang, Dingwei Wang, and Shengxiang Yang. "A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems." In: *Soft Computing* 13.8-9 (2009).

[87] Shengxiang Yang and Xin Yao. "Population-based incremental learning with associative memory for dynamic environments." In: *IEEE Transactions on Evolutionary Computation* 12.5 (2008).

[88] Hendrik Richter and Franz Dietel. "Change detection in dynamic fitness landscapes with time-dependent constraints." In: *2010 Second World Congress on Nature and Biologically Inspired Computing (NaBIC)*. 2010.

[89] Shaaban Sahmoud and Haluk Rahmi Topcuoglu. "Sensor-based change detection schemes for dynamic multi-objective optimization problems." In: *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2016.

[90] Tim Blackwell and Jürgen Branke. "Multiswarms, exclusion, and anti-convergence in dynamic environments." In: *IEEE transactions on evolutionary computation* 10.4 (2006).

[91] Chenyang Bu, Wenjian Luo, and Lihua Yue. "Continuous dynamic constrained optimization with ensemble of locating and tracking feasible regions strategies." In: *IEEE Transactions on Evolutionary Computation* 21.1 (2016).

[92] Wenjian Luo, Juan Sun, Chenyang Bu, and Ruikang Yi. "Identifying species for particle swarm optimization under dynamic environments." In: *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2018.

[93] Daniel Parrott and Xiaodong Li. "Locating and tracking multiple dynamic optima by a particle swarm model using speciation." In: *IEEE Transactions on Evolutionary Computation* 10.4 (2006).

[94] Hani Pourvaziri and B Naderi. "A hybrid multi-population genetic algorithm for the dynamic facility layout problem." In: *Applied Soft Computing* 24 (2014).

[95] Dirk Sudholt. "The benefits of population diversity in evolutionary algorithms: a survey of rigorous runtime analyses." In: *Theory of Evolutionary Computation*. 2020.

[96] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. "Exploration and exploitation in evolutionary algorithms: A survey." In: *ACM computing surveys (CSUR)* 45.3 (2013).

[97] Helen G Cobb. *An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments*. Tech. rep. Naval Research Lab Washington DC, 1990.

[98] John J Grefenstette et al. "Genetic algorithms for changing environments." In: *PPSN*. Vol. 2. 1992.

[99] Frank Vavak, KA Jukes, Terrence C Fogarty, et al. "Performance of a genetic algorithm with variable local search range relative to frequency of the environmental changes." In: *Genetic Programming* (1998).

[100] HC Andersen. "An investigation into genetic algorithms, and the relationship between speciation and the tracking of optima in dynamic functions." In: *Brisbane, Australia: Honors, Queensland Univ* (1991).

[101] Andrea Toffolo and Ernesto Benini. "Genetic diversity as an objective in multi-objective evolutionary algorithms." In: *Evolutionary computation* 11.2 (2003).

[102] Svetlana Minakova, Dolly Sapra, Todor Stefanov, and Andy D Pimentel. "Scenario Based Run-time Switching for Adaptive CNN-based Applications at the Edge." In: *ACM Transactions on Embedded Computing Systems (TECS)* (2021).

[103] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang. "Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing." In: *Proceedings of the IEEE* 107.8 (2019).

[104] NVIDIA. *Jetson TX2*. 2016. URL: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2.

[105] Carlos A Coello Coello, Gary B Lamont, David A Van Veldhuizen, et al. *Evolutionary algorithms for solving multi-objective problems*. Vol. 5. 2007.

[106] Chi-Hung Hsu, Shu-Huan Chang, Jhao-Hong Liang, Hsin-Ping Chou, Chun-Hao Liu, Shih-Chieh Chang, Jia-Yu Pan, Yu-Ting Chen, Wei Wei, and Da-Cheng Juan. "Monas: Multi-objective neural architecture search using reinforcement learning." In: *arXiv preprint arXiv:1806.10332* (2018).

[107] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. "Mnasnet: Platform-aware neural architecture search for mobile." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019.

[108] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. "Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017).

[109] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations." In: 18.1 (2017).

[110] Vinu Joseph, Ganesh L Gopalakrishnan, Saurav Muralidharan, Michael Garland, and Animesh Garg. "A Programmable Approach to Neural Network Compression." In: *IEEE Micro* 40.5 (2020).

[111] Brandon Reagen, Udit Gupta, Robert Adolf, Michael Mitzenmacher, Alexander Rush, Gu-Yeon Wei, and David Brooks. "Weightless: Lossy Weight Encoding For Deep Neural Network Compression." In: *International Conference on Machine Learning*. 2018.

[112] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. "A Survey of Model Compression and Acceleration for Deep Neural Networks." In: *IEEE Signal Processing Magazine* (2018).

[113] Adrián Alcolea Moreno, Javier Olivito, Javier Resano, and Hortensia Mecha. "Analysis of a pipelined architecture for sparse DNNs on embedded systems." In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28.9 (2020).

[114] Xin He, Kaiyong Zhao, and Xiaowen Chu. "AutoML: A Survey of the State-of-the-Art." In: *Knowledge-Based Systems* 212 (2021).

[115] Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, Hanrui Wang, Yujun Lin, and Song Han. "Apq: Joint search for network architecture, pruning and quantization policy." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020.

[116] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. "The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches." In: *ArXiv* abs/1803.01164 (2018).

[117] An-Chieh Cheng, Jin-Dong Dong, Chi-Hung Hsu, Shu-Huan Chang, Min Sun, Shih-Chieh Chang, Jia-Yu Pan, Yu-Ting Chen, Wei Wei, and Da-Cheng Juan. "Searching Toward Pareto-Optimal Device-Aware Neural Architectures." In: *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2018.

[118] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan. "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II." In: *International conference on parallel problem solving from nature*. 2000.

[119] Christos Kyrkou, George Plastiras, Theocharis Theocharides, Stylianos I Venieris, and Christos-Savvas Bouganis. "DroNet: Efficient convolutional neural network detector for real-time UAV applications." In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2018.

[120] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Bowen Shi, Qi Tian, and Hongkai Xiong. "Latency-aware differentiable neural architecture search." In: *arXiv preprint arXiv:2001.06392* (2020).

[121] Weiwen Jiang, Xinyi Zhang, Edwin H-M Sha, Lei Yang, Qingfeng Zhuge, Yiyu Shi, and Jingtong Hu. "Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search." In: *Proceedings of the 56th Annual Design Automation Conference*. 2019.

[122] Mohamed S Abdelfattah, Łukasz Dudziak, Thomas Chau, Royson Lee, Hyeji Kim, and Nicholas D Lane. "Best of both worlds: Automl codesign of a cnn and its hardware accelerator." In: *Proceedings of the 57th Annual Design Automation Conference*. 2020.

[123] Chuan-Chi Wang, Ying-Chiao Liao, Ming-Chang Kao, Wen-Yew Liang, and Shih-Hao Hung. In: *PerfNet: Platform-Aware Performance Modeling for Deep Neural Networks*. 2020.

[124] Ricardo Bonna, Denis S Loubach, George Ungureanu, and Ingo Sander. "Modeling and Simulation of Dynamic Applications Using Scenario-Aware Dataflow." In: 24.5 (2019).

[125] Jiali Teddy Zhai, Sobhan Niknam, and Todor Stefanov. "Modeling, Analysis, and Hard Real-Time Scheduling of Adaptive Streaming Applications." In: *IEEE TCAD* (2018).

[126] O. Moreira. "Temporal analysis and scheduling of hard real-time radios running on a multi-processor." PhD thesis. Technical University Eindhoven, 2012.

[127] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. "Slimmable Neural Networks." In: *International Conference on Machine Learning*. 2019.

[128] L. Liu and J. Deng. "Dynamic Deep Neural Networks: Optimizing Accuracy-Efficiency Trade-Offs by Selective Execution." In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[129] Yue Wang, Jianghao Shen, Ting-Kuei Hu, Pengfei Xu, Tan Nguyen, Richard Baraniuk, Zhangyang Wang, and Yingyan Lin. "Dual Dynamic Inference: Enabling More Efficient, Adaptive and Controllable Deep Inference." In: *IEEE Journal of Selected Topics in Signal Processing* (2020).

[130] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. "Adaptive Neural Networks for Efficient Inference." In: *International Conference on Machine Learning*. 2017.

[131] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens Van Der Maaten, and Kilian Q Weinberger. "Multi-Scale Dense Networks for Resource Efficient Image Classification." In: *International Conference on Learning Representations*. 2018.

[132] Ilias Theodorakopoulos, Vasileios Pothos, Dimitris Kastaniotis, and Nikos Fragoulis. *Parsimonious Inference on Convolutional Neural Networks: Learning and applying on-line kernel activation rules*. 2017.

[133] Martín Abadi, Michael Isard, and Derek G Murray. "A Computational Model for TensorFlow: An Introduction." In: *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*. 2017.

[134] NVIDIA. *TensorRT framework*. 2021. URL: https://developer.nvidia.com/tensorrt.

[135] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. "FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019.

[136] L. Lai, N.Suda, and V. Chandra. "Not All Ops Are Created Equal!" In: 2018.

[137] Saku Kukkonen and Jouni Lampinen. "Ranking-Dominance and Many-Objective Optimization." In: *2007 IEEE Congress on Evolutionary Computation*. 2007.

[138] Dolly Sapra and Andy D Pimentel. "Deep Learning Model Reuse and Composition in Knowledge Centric Networking." In: *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. 2020.

[139] Dapeng Wu, Zhenjiang Li, Jianping Wang, Yuanqing Zheng, Mo Li, and Qiuyuan Huang. "Vision and Challenges for Knowledge Centric Networking." In: *IEEE Wireless Communications* (2019).

[140] Muhammad Ghifary, W Bastiaan Kleijn, and Mengjie Zhang. "Domain adaptive neural networks for object recognition." In: *Pacific Rim international conference on artificial intelligence*. 2014.

[141] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network." In: *arXiv preprint arXiv:1503.02531* (2015).

[142] Jinyu Li, Michael L Seltzer, Xi Wang, Rui Zhao, and Yifan Gong. "Large-scale domain adaptation via teacher-student learning." In: *arXiv preprint arXiv:1708.05466* (2017).

[143] Ziqian Chen, Ling-Yu Duan, Shiqi Wang, Yihang Lou, Tiejun Huang, Dapeng Oliver Wu, and Wen Gao. "Toward Knowledge as a Service Over Networks: A Deep Learning Model Communication Paradigm." In: *IEEE Journal on Selected Areas in Communications* 37.6 (2019).

[144] Chenwei Feng, Mingxia Lin, Xinlin Xie, and Mingjiang Zhang. "Data Compression Scheme for Fronthaul Based on Vector Quantization." In: *Proceedings of the 2019 International Conference on Robotics, Intelligent Control and Artificial Intelligence*. 2019.

[145] Mohammad R Khosravi and Sadegh Samadi. "Data compression in ViSAR sensor networks using non-linear adaptive weighting." In: *EURASIP Journal on Wireless Communications and Networking* 2019.1 (2019).

[146] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. "Edge computing: Vision and challenges." In: *IEEE Internet of Things Journal* 3.5 (2016).

[147] Florin Coras Ma'ruf, Vina Ermagan, Hugo Latapie, Chris Cassar, John Evans, Fabio Maino, Jean Walrand, and Albert Cabellos. "Knowledge-Defined Networking." In: *ACM SIGCOMM Computer Communication Review* 47.3 (2017).

[148] Zubair Md Fadlullah, Fengxiao Tang, Bomin Mao, Nei Kato, Osamu Akashi, Takeru Inoue, and Kimihiro Mizutani. "State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems." In: *IEEE Communications Surveys & Tutorials* 19.4 (2017).

[149] Muhammad Usama, Junaid Qadir, Aunn Raza, Hunain Arif, Kok-Lim Alvin Yau, Yehia Elkhatib, Amir Hussain, and Ala Al-Fuqaha. "Unsupervised machine learning for networking: Techniques, applications and research challenges." In: *IEEE Access* 7 (2019).

[150] Qi Zhang, Xiaofeng Jiang, Shuangwu Chen, Jinsen Xie, Jian Yang, and Ling Xing. "An Information Feature Extraction and Rapid Updating Scheme for Knowledge Centric Networking." In: *2019 International Conference on Computing, Networking and Communications (ICNC)*. 2019.

[151] Tao Zhang, Xingyan Chen, and Changqiao Xu. "Intelligent Routing Algorithm Based on Deep Belief Network for Multimedia Service in Knowledge Centric VANETs." In: *2018 International Conference on Networking and Network Applications (NaNA)*. 2018.

[152] Daniel L Silver, Qiang Yang, and Lianghao Li. "Lifelong machine learning systems: Beyond learning algorithms." In: *2013 AAAI spring symposium series*. 2013.

[153] Tom Mitchell, William Cohen, Estevam Hruschka, Partha Talukdar, Bo Yang, Justin Betteridge, Andrew Carlson, B Dalvi, Matt Gardner, Bryan Kisiel, et al. "Never-ending learning." In: *Communications of the ACM* 61.5 (2018).

[154] Rajasekar Venkatesan and Meng Joo Er. "A novel progressive learning technique for multi-class classification." In: *Neurocomputing* 207 (2016).

[155] Liang Lin, Keze Wang, Deyu Meng, Wangmeng Zuo, and Lei Zhang. "Active self-paced learning for cost-effective and progressive face identification." In: *IEEE transactions on pattern analysis and machine intelligence* 40.1 (2017).

[156] Tian Gao, Jun Du, Li-Rong Dai, and Chin-Hui Lee. "Densely connected progressive learning for lstm-based speech enhancement." In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018.

[157] Song Han, Jeff Pool, John Tran, and William Dally. "Learning both weights and connections for efficient neural network." In: *Advances in neural information processing systems*. 2015.

[158] Song Han, Huizi Mao, and William J Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." In: *arXiv preprint arXiv:1510.00149* (2015).

[159] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. "Pruning filters for efficient convnets." In: *International Conference on Learning Representation*. 2017.

[160] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. "Fixed point optimization of deep convolutional neural networks for object recognition." In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015.

[161] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. "Deep learning with limited numerical precision." In: *International Conference on Machine Learning*. 2015.

[162] Min Lin, Qiang Chen, and Shuicheng Yan. "Network in network." In: *arXiv preprint arXiv:1312.4400* (2013).

[163] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.

[164] Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit." In: *Nature* 405.6789 (2000).

[165] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. "What is the best multi-stage architecture for object recognition?" In: *IEEE International Conference on Computer vision*. 2009.

[166] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines." In: *Proceedings of the 27th international conference on Machine Learning*. 2010.

[167] Bekir Karlik and A Vehbi Olgac. "Performance analysis of various activation functions in generalized MLP architectures of neural networks." In: *International Journal of Artificial Intelligence and Expert Systems* 1.4 (2011).

[168] Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. "Understanding and improving convolutional neural networks via concatenated rectified linear units." In: *International Cconference on Machine Learning*. 2016.

[169] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015.

[170] Ming Zeng, Le T Nguyen, Bo Yu, Ole J Mengshoel, Jiang Zhu, Pang Wu, and Joy Zhang. "Convolutional neural networks for human activity recognition using mobile sensors." In: *6th International Conference on Mobile Computing, Applications and Services*. 2014.

[171] Jianbo Yang, Minh Nhut Nguyen, Phyo Phyo San, Xiao Li Li, and Shonali Krishnaswamy. "Deep convolutional neural networks on multichannel time series for human activity recognition." In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015.

[172] Stergios Christodoulidis, Marios Anthimopoulos, Lukas Ebner, Andreas Christe, and Stavroula Mougiakakou. "Multisource transfer learning with convolutional neural networks for lung pattern analysis." In: *IEEE journal of biomedical and health informatics* 21.1 (2016).

[173] Jinpeng Li, Shuang Qiu, Yuan-Yuan Shen, Cheng-Lin Liu, and Huiguang He. "Multisource Transfer Learning for Cross-Subject EEG Emotion Recognition." In: *IEEE transactions on cybernetics* (2019).

[174] *Zlib: A Massively Spiffy Yet Delicately Unobtrusive Compression Library*. 2019. URL: https://www.zlib.net/.

[175] *Protocol Buffers*. 2019. URL: https://developers.google.com/protocol-buffers.

[176] *NVIDIA GEFORCE RTX 2080 GPU*. 2019. URL: https://www.nvidia.com/en-in/geforce/graphics-cards/rtx-2080/.

[177] Alireza Rahimpour, Ali Taalimi, Jiajia Luo, and Hairong Qi. "Distributed object recognition in smart camera networks." In: *2016 IEEE International Conference on Image Processing (ICIP)*. 2016.

[178] Roberto Marroquin, Julien Dubois, and Christophe Nicolle. "Ontology for a Panoptes building: Exploiting contextual information and a smart camera network." In: *Semantic Web* 9.6 (2018).

[179] Phoebus Chen, Parvez Ahammad, Colby Boyer, Shih-I Huang, Leon Lin, Edgar Lobaton, Marci Meingast, Songhwai Oh, Simon Wang, Posu Yan, et al. "CITRIC: A low-bandwidth wireless camera network platform." In: *2008 Second ACM/IEEE International Conference on Distributed Smart Cameras*. 2008.

"Computers are useless. They can only give you answers."

— Pablo Picasso

ACKNOWLEDGMENTS

The real intelligence comes from our ability to ask questions, whereas the artificial intelligence stems from the processing capability of computers, when it tries to answer some of those questions. My PhD was a very interesting quest to ensure that even the smallest computer can acquire some form of artificial intelligence. Countless people helped me in this entertaining and thought-provoking adventure. As I am about to finish this journey, I would like to express my appreciation and gratitude towards my supervisors, all colleagues, family and friends who supported me through the last few years.

Foremost, thanks to my husband, Sushant, for constantly encouraging me and listening to my ideas and my rants, while understanding none of it. I appreciate him celebrating every little win and making me laugh when I was in pits. I am also thankful to my daughter, Amaira, whose infinite love and constant need for attention kept me on my toes, though still motivated enough to keep working towards my publications.

My most sincere gratitude goes to my supervisor and mentor, Andy Pimentel, for taking me under his wing as a PhD student. He is the closest witness to my struggles and my victories. During the course of my PhD, he was always available to listen to me, to encourage me, to provide guidance when I felt lost, and most importantly, to add articles in my articles. I have learnt a lot from you Andy, and I hope to carry my future self with same values and ethics that you always lead with.

I am also thankful to Cees de Laat, for regularly spending time to advise me on professional as well as other important matters. A special thanks to Sebastian Altmeyer, who was my first supervisor and made it possible for me to begin my academic career in the SNE/PCS research group. A warm thanks to Simon Polstra for all the lively discussions and helping me with all the stuff I never wanted to deal with myself. A heartfelt appreciation to Paola Grosso for the endless conversations on all walks of life.

I would like to extend my gratitude to the members of the examination committee, prof. dr. ir. Clarisa Sánchez Gutiérrez, prof. dr. Judith Good, dr. Paola Grosso, prof. dr. Diana Marculescu, prof. dr. Patricia Lago and prof. dr. Marian Verhelst, for reading this thesis and providing their invaluable feedback.

Additionally, I would like to thank my colleagues in the SNE research group, for all the lunches, learnings and discussions. Specially to Hugo, Uraz, Xiaotian, Julius, Jun, Marius, Lukas, Benny, Clemens, Ana O., Ana V., Francesco, Benjamin, Ameneh, Anuj, Jelle, Ralph, Joe, Misha, Leonardo, Jamila, Milen, Marco, Ehsan, Daphne, Sudam, Pooya and Giulio.

I was also fortunate to work with so many wonderful people in the ALOHA project. A big thanks to Svetlana and Todor for their help in the project, and also for the shared time, discussions and musings during the project related travels. I will always fondly remember my interactions for ALOHA with Paolo, Gianfranco, Francesca, Daniela, Maura, Bernhard, Natalia, Giulio, Battista, David, Francesco, Adriano and Maya.

A lovely little note of thanks to my friends, Annelies, Bert, Chandni, Rohit, Jayati, Avanti, Amit, Ajinkya, Jyoti and Parul. Your friendship has rewarded me many times with love, support, conversations, food and amazing wine. I appreciate having you in my life and all the good times we spend together.

My family and my parents-in-law require a special mention for their role in always encouraging me and supporting me in everything I have aspired to do. It is only because of your unwavering belief in me that I could find the confidence to pursue the PhD dream. I will always aspire to make you proud mummy and papa. I will always aspire to tease you with my new degree Bhai. I will always aspire to be the awesome and brilliant sister to you Swati.

There were many people who supported me in the past few years and I might have missed a few names despite my best effort to list everyone. I apologise if your name should have been mentioned here, I am still grateful for your help and support.

SUMMARY

Generally, deep neural networks are executed on big servers on the cloud with availability of many GPUs and other computational resources. However, there has been a conspicuous rise of IoT networks with numerous connected devices and a demand for data processing closer to the data source. This has led to a strong interest in deployment of deep learning models at the edge. An edge refers to the small micro-processor based computer hardware with limited resources, and is usually placed near to a data sensor or a consumer.

Executing a neural network at the edge makes an advantageous premise and may provide better privacy, security and reliability. Deployment of neural networks at the edge is highly desirable, though challenging, for many applications. The main challenge arises from the fact that neural networks demand high computational capabilities from the underlying hardware, whereas an edge device has limited resource availability.

In this thesis, the focus is particularly on the neural architectures of Convolutional Neural Networks (CNNs) that can execute on the edge devices. The first part of the thesis presents *Evolutionary Piecemeal Training* (EPT), an evolutionary based algorithm to search for an efficient neural network architecture. This algorithm treats the Neural Architecture Search (NAS) as an optimization problem and provides a flexible methodology to consider a single objective or multiple objectives for the search.

The initial single objective EPT experiments considered only the accuracy maximization of the resulting neural network. To ensure their suitability to the edge devices, the model size was restricted through the constraints placed on the number of parameters of the neural networks. The next set of experiments extended the algorithm to consider two objectives, namely the accuracy maximization and the minimization of the number of parameters of the model.

In yet another set of experiments, also referred to as *hardware-aware EPT*, four objectives were considered for the search. As the name suggests, apart from the accuracy maximization, the rest of the objectives are specific to the execution of a neural network on a target hardware. The objectives

aimed towards CNN execution on the device are collectively referred to as ATME characteristics, which is short for Accuracy, Throughput, Memory and Energy characteristics. The *hardware-aware EPT* is able to derive various CNNs, which provide different trade-offs for the objectives in consideration through a pareto optimal set.

Furthermore, the second part of the thesis examines strategies and techniques to ensure adaptivity of the CNN-based application running on the edge. The first work presented in this direction is the Scenario Based Runtime Switching (SBRS) framework. SBRS proposes the concept of scenarios, where each scenario is associated with a unique CNN and represents an operation mode of the application. An operation mode reflects the immediate environment needs of the target device at different circumstances. An application in SBRS may switch from one scenario to another, therefore, allowing the application to adapt synchronously with the environmental changes.

Continuing with the theme of adaptivity, we proposed a framework to investigate the efficient sharing, exploitation and reusability of deployed CNNs in a distributed network. This is realized in the context of Knowledge Centric Networking (KCN) by considering neural networks as dynamic models. In terms of adaptivity, this framework aims to provide the support needed for maintenance and modification of existing and deployed CNNs at the edge.

To conclude, the work presented in this thesis demonstrates various strategies and methodologies focused on neural architectures. The aim is to improve the performance of a CNN-based application deployed on a resource-constrained edge device. In a nutshell, the key ideas explored in this thesis include searching for an efficient neural architecture, adaptive applications to allow run-time CNN switching and CNNs as dynamic entities in a distributed IoT network.

SAMENVATTING

Over het algemeen worden diepe neurale netwerken uitgevoerd op grote servers in de cloud met de beschikbaarheid van veel GPU's en andere computerbronnen. Er is echter een opvallende opkomst van IoT-netwerken met tal van aangesloten apparaten en een vraag naar gegevensverwerking dichter bij de gegevensbron. Dit heeft geleid tot een sterke interesse in de inzet van deep learning-modellen aan de *edge*. Een edge verwijst naar de relatief eenvoudige computerhardware op basis van een microprocessor met beperkte middelen en wordt meestal in de buurt van een gegevenssensor of een consument geplaatst.

Het uitvoeren van een neuraal netwerk aan de edge is voordelig aangezien het kan zorgen voor betere privacy, beveiliging en betrouwbaarheid. Hoewel uitdagend voor veel applicaties, is de inzet van neurale netwerken aan de edge voor veel toepassingen dus zeer wenselijk. De grootste uitdaging komt voort uit het feit dat neurale netwerken hoge rekencapaciteiten van de onderliggende hardware vereisen, terwijl een edge-apparaat een beperkte beschikbaarheid van middelen heeft.

In dit proefschrift ligt de focus op zogenaamde convolutionele neurale netwerken (CNN's) die kunnen worden uitgevoerd op edge-apparatuur. Het eerste deel van het proefschrift presenteert *Evolutionary Piecemeal Training* (EPT), een evolutionair gebaseerd algoritme om te zoeken naar een efficiënte neurale netwerkarchitectuur. Dit algoritme behandelt de Neural Architecture Search (NAS) als een optimalisatieprobleem en biedt een flexibele methodologie om een enkele doelstelling of meerdere doelstellingen voor de zoekopdracht in overweging te nemen.

Bij de eerste EPT-experimenten met één enkele doelstelling werd alleen de nauwkeurigheidsmaximalisatie van het resulterende neurale netwerk in overweging genomen. Om de geschiktheid voor de edge-apparatuur te garanderen, werd de modelgrootte van de CNN modellen beperkt door een maximum op het aantal parameters van de neurale netwerken. De volgende reeks experimenten breidde het algoritme uit om twee doelen in overweging te nemen, namelijk het maximaliseren van de nauwkeurigheid en het minimaliseren van het aantal parameters van het model.

In nog een andere reeks experimenten, ook wel *hardware-aware EPT* genoemd, werden vier doelen voor het zoeken overwogen. Zoals de naam al doet vermoeden, zijn de meeste van deze doelstellingen, afgezien van het maximaliseren van de nauwkeurigheid, specifiek voor de uitvoering van een neuraal netwerk op specifieke hardware. De doelstellingen gericht op CNN-uitvoering op het apparaat worden gezamenlijk ATME-kenmerken genoemd, wat een afkorting is voor nauwkeurigheid, doorvoer, geheugen en energiekenmerken. De *hardware-aware EPT* is in staat om verschillende CNN's af te leiden, die verschillende afwegingen bieden voor de beoogde doelstellingen via een pareto-optimale set.

Verder onderzoekt het tweede deel van het proefschrift strategieën en technieken om de adaptiviteit van een op CNN gebaseerde applicatie die aan de edge draait, te verzekeren. Het eerste werk dat in deze richting wordt gepresenteerd, is het Scenario Based Run-time Switching (SBRS) raamwerk. SBRS stelt het concept van scenario's voor, waarbij elk scenario is gekoppeld aan een unieke CNN en een bedrijfsmodus van de toepassing vertegenwoordigt. Een bedrijfsmodus weerspiegelt de onmiddellijke omgevingsbehoeften van het doelapparaat onder verschillende omstandigheden. Een toepassing in SBRS kan van het ene scenario naar het andere overschakelen, waardoor de toepassing zich synchroon aanpast aan de omgevingsveranderingen.

Voortbordurend op het thema van adaptiviteit, hebben we een raamwerk voorgesteld om het efficiënt delen, exploiteren en hergebruiken van ingezette CNN's in een gedistribueerd netwerk te onderzoeken. Dit wordt gerealiseerd in de context van Knowledge Centric Networking (KCN) door neurale netwerken als dynamische modellen te beschouwen. Op het gebied van adaptiviteit heeft dit raamwerk tot doel de ondersteuning te bieden die nodig is voor onderhoud en wijziging van bestaande en ingezette CNN's aan de edge.

Tot slot, het werk dat in dit proefschrift wordt gepresenteerd demonstreert verschillende strategieën en methodologieën gericht op neurale architecturen. Het doel is om de prestaties te verbeteren van een op CNN gebaseerde applicatie die wordt geïmplementeerd op een edge-apparaat met beperkte middelen. In een notendop, de belangrijkste ideeën die in dit proefschrift worden onderzocht, zijn onder meer het zoeken naar een efficiënte neurale architectuur, adaptieve toepassingen om runtime CNN-switching mogelijk te maken en CNN's als dynamische entiteiten in een gedistribueerd IoT-netwerk.