UNDERSTANDING NEURAL REUSE: A CASE STUDY ON IMPROVING ENERGY EFFICIENCY OF CONVOLUTIONAL NEURAL NETWORKS

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF SCIENCE AND ENGINEERING

2022

Merve Selçuk Şimşek

School of Computer Science

Contents

Word Count: 31120

List of Tables

List of Figures

Abstract

Understanding Neural Reuse:
A Case Study on Improving Energy Efficiency of
Convolutional Neural Networks
Merve Selçuk Şimşek
A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy, 2022

Artificial Neural Networks (ANNs) constitute a vital part in the Artificial Intelligence (AI) technology today. Neural networks run in basically every electrical machinery for the purposes of response-time efficiency, machine learning, or simulation. These networks have helped so many advancements in different parts of the areas over the years. However, even the simplest ANN is power-hungry and causes an overhead to the running platform; it is always expensive to benefit from a network. Moreover, separate tasks are handled by different networks in order to make the output produced by the network more accurate which results even more overhead.

Our main research question is "How can we make multiple ANNs more energy efficient without losing their effectiveness?". Accordingly, we showcase at least two different tasks can be handled by the same network which is more efficient when the two regular networks are combined. This idea is rooted and inspired by the *Neural Reuse* theory which indicates the same neural paths are used for more than one task in the brain, and these tasks do not need to be similar at all.

We merge two Convolutional Neural Networks (CNNs), one recognises sounds and the other recognises images, then compress them via quantisation and make them even lighter and faster as our case study. This research exploits and employs open-source software resources and pretrained CNNs. The research targets traditional hardware resources, such as a PC with a regular CPU or a mobile phone, and facilitates solutions to this extent. Our ultimate aim is to provide easily accessible, i.e., open-source, and reproducible *by everybody with a simple computer* software solutions while increasing energy efficiency in multiple-networked systems via this research.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

I remember one particular time when I came across an *Acknowledgement* part in a PhD dissertation, which went on and on about what a tough journey it was and how challenging it was, that ending with the individual to thank oneself. I found it unusual back then, during the infancy of my PhD journey. Well, youth. After so many ups and downs in a roller coaster like journey which is called a PhD in the literature of all candidates, a world-wide pandemic, losing two of my extended family members/loved ones, and waiting anxiously the quarantine result of my mother and sister during the final week of my submission deadline; I just have to start this part, before everything else, by patting myself on the back and saying "You survived. You did it. And you did it well.".

I would like to thank my main supervisor Mikel, Prof. Luján, for many things indeed. Thanks for being kind and understanding to me for the times I sounded hopeless, and felt like the *Atlas* regarding the enormous project I pursued from the very beginning. A strong thanks for trusting me and my abilities; sending me to the outer world to represent our team, to make presentations, to connect with people, and providing an environment for networking. I'd like to thank you for encouraging me into pursuing a PhD right in the beginning and being one of the best pep talk givers I knew back then. Thank you for pointing, perhaps sometimes too many, research directions at me Mikel; yet enabling an environment where I needed to be the soul captain of my own ship facing the tsunamis, that is this PhD. Thank you for this opportunity and for your belief in me that I would make it through regardless.

A big thanks go to my viva examiners Dr. David Halliday and Dr. Riza Batista-Navarro at this point. I've never thought the viva day would be a day that I would cherish regardless the outcome considering the anxiety the day carries. Thank you both for your politeness which relieved my stress on this crucial day. I also appreciate your detail-oriented, insightful comments, and valuable advice which improved my thesis.

Next, I'd like to thank some more academics who were a part of my journey by contributing to my field of knowledge and shaping my road-map in such sense. Thank you Prof. Ross King for introducing the world of *C. Elegans* to me; Prof. Steve Furber, my co-supervisor, for recommending me some of the books that now take place in my bibliography, and inviting me to research on *ants*; Prof. Barbara Webb and Dr. Jörg Conradt for giving me feedback on my research when we met at scientific meetings, and indeed encouraging me by acknowledging that my idea was "good" and worth following for. Thanks to you and my never-ending interest of researching the animal brains, my research took a long way from investigating a brain atlas for *frogs* to this far.

I'd like to thank Angela Saini, Prof. Brené Brown, and Prof. Cecilia Laschi with two of whom I had the chance to meet personally. Thank you for all your encouraging words and support which gave me "more power" indeed in this journey.

I'd also like to thank my colleagues Gabriel Fonseca-Guerra, Swapnil Gaikwad, and Guillermo Callaghan, who should all have the *Dr.* title before their names by now; for reading my reports, giving me feedback, and for the meaningful discussions along the way. Even though you were oblivious to my field of research at times, I am indeed thankful that you've always found it "cool".

I am proud to say that this research stands on the shoulders of giants. As being someone who was never shy of reinventing the wheel once upon a time, this is perhaps the most valuable lesson that I have taught myself during my PhD studies. As always being an open-source champion, now more than ever, I appreciate the open-source community and everybody who is generous enough to share their work publicly -and in a human-readable form!- on online resources. I'd like to especially thank the *larq* team, and the creator of *SpKeras*, Dengyu Wu, in this sense, who were kind enough to answer my countless questions, and hence contributed to my insight in the field of energy-efficient convolutional neural networks.

I thank all the members of my family with their never-ending belief in me in anything I do. Special thanks to my loving sister Mina, for the various song lists she prepared to keep me concentrated on my work, and all the "good vibes" she sent me from roughly four thousand km. away in various forms really to keep me as "zen" as possible during this trying period. Additionally, I thank my friends who supported me throughout the way and believed the outcome of this thesis would be positive no matter what happened, especially Gözde Kutayer-Bilgin and Dr. Mustafa Çavuş.

Lastly, yet the most, I am thankful for my husband, my partner in life, my colleague, and my best friend, dear Seçkin. Thank you for being an amazing cook; standing by me; cheering me up, bringing joy to my life; and thank you for *fixing* me, regardless the subject, throughout this crazy journey. Without you, this journey wouldn't be as bearable as it was, or, to put it simply, wouldn't conclude happily.

This thesis is dedicated to my beloved grandpa Şeref, who lost his life during the recent pandemic while the world lost a true gentleman. He is and will always be missed.

Son olarak, *in the name of Jäger...* Bu dönemde tanıdığım herkese ve yaşanan acı-tatlı, büyümeme katkısı olan, her şeye teşekkürler. Şerefe!

Glossary

AI	Artificial Intelligence. 18, 22, 23, 68, 107, 127, 128
ANN	Artificial Neural Network. 18, 19, 21, 22, 24–26, 28, 31–36, 38–41, 43, 44, 49, 51–54, 56, 58, 61–65, 68, 74, 76, 78–83, 86–88, 90, 94–96, 101, 104, 105, 107, 108, 111, 117, 120, 123, 126–130
API	Application Programming Interface. 73, 81–83, 95, 96
BERT	Bidirectional Encoder Representations from Transformers. 19, 21
BN	Batch Normalisation. 46, 88, 90, 92–94, 98, 102
BNN	Binarised Neural Network. 22, 25, 26, 31, 49, 50, 52, 76, 81, 89, 90, 92, 97, 98, 103, 108, 110–112, 114, 117, 118, 123, 124, 126
BP	Back-Propagation. 31, 36–38, 40–42, 70, 72
CNN	Convolutional Neural Network. 18, 19, 22–27, 31, 40–44, 46–49, 52, 63, 64, 67, 70, 75, 78, 81, 82, 86–89, 92–98, 102–106, 108, 110–121, 123–129
CPU	Central Processing Unit. 42, 50, 105, 108–110, 129
CTRNN	Continuous Time Recurrent Neural Network. 62, 65
DARTS	Differential Architecture Search. 87
DNN	Deep Neural Network. 43, 44, 48
FC	Fully-Connected. 39, 42, 43, 50, 88, 89, 92, 93, 102
FLOPs	FLOating-Point operations. 89, 101, 102, 107, 108, 126, 127, 129
fMRI	functional Magnetic Resonance Imaging. 58
FP	Full bit Precision-Floating Point. 48, 49, 74, 75, 78, 81, 90, 95–98, 108, 110–120, 123, 124, 126–128
GPU	Graphics Processing Unit. 40, 44, 74, 129
HBP	Human Brain Project. 51

ILSVRC	ImageNet Large Scale Visual Recognition Challenge. 43, 44, 46, 47, 106
KB	Kilobyte. 116
KiB	Kilo Binary Byte. 108, 116
MACs	Multiply and ACcumulate operations. 50, 107, 108, 111, 112, 123, 126, 127, 129
MB	Megabyte. 116
MiB	Mega Binary Byte. 108, 116, 121
ML	Machine Learning. 22, 24, 41, 44, 70, 81, 104, 105, 107, 127, 128
MLP	Multi-Layer Perceptron. 31, 33, 39–41
MNIST	Modified National Institute of Standards and Technology. 82, 83, 85–87, 93, 103, 110, 115, 117
MSE	Mean Squared Error. 107
NAS	Neural Architecture Search. 19, 86, 87
NIST	National Institute of Standards and Technology. 82
NLP	Natural Language Processing. 68, 129
QNN	Quantised Neural Network. 22, 25, 26, 31, 49, 52, 81, 92, 95, 96, 103, 108, 112–116, 118, 120, 121, 123, 124, 126, 128
ReLU	REctified Linear Unit. 44, 88, 90, 92, 95, 96, 114
SNN	Spiking Neural Network. 22, 25, 26, 31, 32, 51–53, 61, 62, 65, 67, 75, 78–81, 88, 95, 96, 103, 108, 118–120, 123, 124, 126–128
SotA	State of the Art. 23, 61, 64, 79, 83, 85–87, 95, 117, 129
STDP	Spike Time Dependant Plasticity. 32, 62
TF	TensorFlow. 73, 76–78, 81–83, 93, 95, 96, 102, 105, 106, 108, 109, 114, 116
TL	Transfer Learning. 21, 22, 24, 124, 126

Chapter 1

Introduction

Energy can neither be created nor destroyed; it can only be transformed from one form to another.

ÉMILIE DU CHÂTELET, PRINCIPIA-*Commentary*, 1749

Inspired by nature, by the biological neural networks, Artificial Neural Networks (ANNs) has been on stage (Rosenblatt, 1958) for more than half a century. The simulations of biological neural networks are important to comprehend *where the intelligence originates*, *how to fight with some mental diseases*, and *to see the effects of drugs in the brain*. Their emulations are also important to aid us, human beings, in daily life as Artificial Intelligence (AI)[1] that able to hear, speak, write, see, and move (Anantrasirichai and Bull, 2022).

The first ANN (Hay et al., 1960) was a visual deciphering emulation. Understanding visual cues and deciphering them computationally have always been and are still of interest to the AI community. Therefore, the rise, hence the common use, of the ANNs started with the Convolutional Neural Networks (CNNs) which were formed initially to process visual data, i.e., images and videos. This rise led to *challenges* specifically created for intelligent visual processing (Russakovsky et al., 2015), various datasets and benchmarks, a diverse range of ANN types and architectures, overall, ANNs becoming successful deciphers, based on metrics, that compartmentalised in various places, from mobile devices to TV's, in our everyday lives. In time, the situation of ANNs created a *Jevons Paradox* (Giampietro and Mayumi, 2018), meaning the more they were made available, reachable, and "efficient to use" for various tasks; the more they became in

[1]We would like to highlight the relationship between the fundamental terms that we use in the thesis before moving further. From left to right the terms gets more specific, and the terms on the right are sub-terms of their preceding term: $AI \rightarrow MachineLearning(ML) \rightarrow ANNs \rightarrow CNNs$.

demand. Their rise happened because of their being data-hungry given the demanded task repertoire; and combining it with the demand towards ANNs eventually led them to become power-hungry.

A recent study showcased that training *Google*'s Bidirectional Encoder Representations from Transformers (BERT) has almost the same amount of CO_2 emission (lbs) as a trans American flight (Strubell et al., 2020). Moreover, according to the same study, training a Neural Architecture Search (NAS) network has $\approx$5 times more CO_2 emission than a car does throughout her life-time. Such results are displayed in Table 1.1.

Table 1.1: CO_2 equivalent emissions (lbs) of consumption vs. training ANN models (Strubell et al., 2020).

		CO_2e[2](lbs)
Consumption	Air travel, *1 passenger, NY ⇔ SF*	1,984
	Car, *avg incl. fuel, 1 lifetime*	126,000
Model	BERT	1,438
	NAS	626,155

Considering the crises that going on in our world today (Kennedy et al., 2021), keeping the greenhouse gas, e.g., CO_2, emissions low has the uttermost value now more than ever (Yue and Gao, 2018; Chen, 2021). One way to carbon neutrality, meaning to achieve a balance between emitting and absorbing the carbon, is "reducing energy consumption and increasing energy use efficiency" (Chen, 2021). We conclude that as the recent dramatic findings prove (Strubell et al., 2020; Patterson et al., 2021), ANNs, despite their current high-value in our lives, should be no exception to that.

This research investigates methods to make the ANNs, specifically CNNs, more energy efficient. With the increased types of neural networks and their increased use cases, especially on mobile platforms, e.g., autonomous cars, mobile robots, and phones, not one, but at least two neural networks are in use today. Therefore, although we also showcase how to make singular CNNs more energy efficient via our experiments, our hypothesis is on **making multiple CNNs more energy efficient**. The "more" here means, obtaining better metrics results, concerning the energy, compared to their separate forms combined. Biological neural networks are the roots of our hypothesis which is explained in the next Section, Section 1.1. Afterwards, we move on to elaborate our research motivation in Section 1.2; outline the research questions and

[2]CO_2e means carbon dioxide equivalent emissions that includes the primary greenhouse gas, i.e., carbon dioxide, along with other greenhouse gases, e.g., methane, nitrous oxide.

objectives of this thesis in Section 1.3; explain thesis scope in Section 1.4; discuss our contribution to the literature in Section 1.5; and conclude this chapter by offering a summary to the thesis structure in Section 1.6.

1.1 Nature's Solution to Preserving Energy in the Brain

As a chordate, a *sea squirt* is born with a nerve cord and a primitive nervous system. Larval sea squirt, which resembles a frog tadpole, can sense the light and gravity, freely move and search for food, all thanks to her primitive "brain" (Smith-Ferguson and Beekman, 2020). Prior to stepping into adulthood, the sea squirt finds a safe haven for herself, perhaps a shadow of a rock, and lands there for good. Adulthood means metamorphosis for her which causes a series of physical changes in her body (Karaiskou et al., 2015). The most dramatic change that occurs to the sea squirt is that her nervous system disintegrates from her body and is gradually absorbed at this stage. After this point, without a *"brain"* in her body, her journey turns into waiting for the prey to come to herself. The adult squirt feeds herself with this way, without spending any extra *energy*, until the end of her life.

A brain full of neurons as we know of is proved to be non-vital to live a life and survive in animal kingdom, yet the brain emerged for a greater purpose than keeping organisms just alive. Animals that are known to have no neural cells at all, for example *sponges*, are also suspected to have them once upon a time, yet lost them by evolution (Ryan and Chiodin, 2015) simply because their existence caused a budget that they could not maintain. It is not the existence of the brain that matters, nor the size of the brain, but its fitness to its function (Czekanski-Moir and Rundell, 2020). The brain is necessary for a better adaptation of an organism while regulating energy consumption for its cause at the same time (Sterling and Laughlin, 2015). A better adaptation increases the chance of survival and reproduction of the organism. Moreover, less energy consumption in its body, in other words, saving the withhold energy, enables the adaptation process to be maintained.

The human brain constitutes 2% of the body weight, yet consumes 20% of the body's overall energy (Attwell and Laughlin, 2001). Keeping a "brain" and and maintaining its functions are expensive for the host; and the reason of this phenomenon, especially regarding human brain as being the most cognitively capable in all animals, has always captured interest of the researchers (Pulido and Ryan, 2021). Despite the brain might seem as "power-hungry" by looking at the aforementioned numbers, the expense of such energy in return of brain's endless function repertoire remains indispensable, and even may seem little when dug deeper.

The more advanced the technology gets, the deeper gets our knowledge in neuroscience. It was believed earlier that the human brain is partially different than primate brain; for instance, it weighs heavier and have more brain regions and neurons than its counterparts. However, this knowledge now broadens thanks to the new imaging technology: it is now known that the human brain is not only partially, but fundamentally different. This information shows itself in *energy preserving* within the human brain which is greater in humans even compared to our closest living relatives based on evolution (Beaulieu-Laroche et al., 2021).

There are several approaches aiming to explain how the brain utilises and preserves its energy. One prominent approach aiming to explain this phenomenon, *the free energy principle*, takes the brain as a whole where neurons in different brain regions (Strotzer, 2009) may or may not be connected. Nonetheless, according to the *free energy principle*, brain always chooses the most optimal path which would consume the lowest possible energy[3] in order to realise any functionality (Friston, 2010).

Another approach, *neural reuse*, offers an explanation to the phenomenon with a *pre* and a *post* evolution lookout. Neural reuse theorises that paths needed to realise new functionalities in the brain are overridden onto already existent paths, rather than creating new paths, in order to save energy (Anderson, 2010, 2015). The neural reuse and *the free energy principle* theories do not clash, but complement each other in our opinion which inspires and constitutes the root of this thesis. We elaborate this topic in Chapter 3.

1.2 Motivation and Hypothesis

Although the ANNs have become costly in time, their benefits can not be overlooked. Despite its enormous training cost as shown in Table 1.1, during the recent pandemic, i.e., *COVID-19*, BERT has helped many scientists and researchers world-wide by powering the COVID-19 Research Explorer[4]. Therefore, the solution should be addressed to the **question** of "How can we make ANNs more energy efficient without losing their effectiveness?".

With the increased ANN usage, so was born network-based *Transfer Learning (TL)*

[3]The energy optimisation is put forward via *maximum entropy* approach in the paper.

[4]`https://covid19-research-explorer.appspot.com/biomedexplorer/`

(Tan et al., 2018). TL is a promising technique as it provides a shortcut for the ones who would like to build their ANNs upon the existent, i.e., trained, ones. The *transfer* here can be obtained based on a part of the trained network, e.g., via some of its layers, or can be obtained by including the whole trained network into another, new, network. However, TL comes with a limitation and a cost. The limitation is that for a successful "transfer", the network source domains, i.e., datasets, should be related. To illustrate, if one domain comprises image data, the other one should do as well. Moreover, TL requires extra training, ergo it is costly. The trained network, which will be addressed as *pretrained network* throughout the thesis, that is used as a base, should be trained even more, so that it would fit the new domain of hers.

This research investigates methods to make multiple CNNs energy efficient by spending as little cost as possible with the consideration of performance/efficiency trade-off. We emulate the neural reuse theory onto pretrained CNNs in this thesis, also benefit from innately low-cost, energy efficient ANNs, such as Quantised Neural Networks (QNNs), Binarised Neural Networks (BNNs), and Spiking Neural Networks (SNNs). There are various online resources[5,6,7] that supply pretrained ANNs today. Our hypothesis suggests that, via such resources, we can benefit from ANNs more than TL offers. In the light of our research question, we offer a **hypothesis**: "Multiple CNNs that are combined via neural reuse will preserve more energy than their counterparts when converted into their innately energy efficient relatives".

Considering the rising cost of the CNNs especially after the challenges, as mentioned in the Introduction, that were specifically organised for them, initiatives have taken place to extend the use of low-power, on budget ANNs (Hu et al., 2021). Such initiatives are generally called as "Green AI" today (Schwartz et al., 2020). All our experiment efforts within this research involve techniques that align with Green AI.

There were also other initiatives to fix various problems in the domain of Machine Learning (ML), specifically regarding ANNs. We list some of the addressed problems in the literature along with the correspondent research below.

1. The weaknesses of the current ANN models, e.g., *"brittleness"*, being data-hungry, presuming a stable world (Marcus, 2018; D'Amour et al., 2020).

[5]TensorFlow (TF) - Hub: `https://tfhub.dev/`
[6]PyTorch - Model Zoo: `https://pytorch.org/serve/model_zoo.html`
[7]The Model Zoo *for multiple platforms*: `https://modelzoo.co/`

2. Focusing too much onto certain metric results and benchmarks; lack of practical, real-world, solutions (Wagstaff, 2012; Marcus, 2018; Lipton and Steinhardt, 2019; Sodhani et al., 2020).

3. Underrepresented data and lack of diversity in datasets (Shankar et al., 2017).

4. Problems regarding openness, availability, and reproducibility (Dacrema et al., 2019; Sodhani et al., 2020).

As our research concerns energy efficiency in CNNs, we only focus on items 2 and 4 in the above list. Aiming to reflect both the Green AI principles and item 2 above, we use two separate metrics set to evaluate our hypothesis, and seven main metrics in total as we aim to deliver this work as transparent as possible. We explain our metrics in Chapter 6 - Section 6.1. Considering our work is upon increasing the energy efficiency of the neural networks, we believe we address a real-world problem.

Dacrema et al. state the importance of reproducibility in their work as $^{11}/_{18}$ State of the Art (SotA) studies in *top-n recommendation systems* that they examined failed to be reproduced. Sodhani et al. highlight the openness and sharing both the literary and the *code* works of the research. We consider both statements regarding item 4, and reflect them onto our research which are addressed in Chapter 4 - Section 4.1.2.4 and Chapter 5 - Section 5.1.

1.3 Research Questions and Objectives

In the prior section, Section 1.2, we discussed our motivation, mentioned our main research question, and stated the hypothesis of this thesis. In this section, we offer a collective overview to the first steps of our scientific method: Research questions, hypothesis, and objectives.

The hypothesis of this research is: Multiple CNNs that are combined via neural reuse will preserve more energy than their counterparts when converted into their innately energy efficient relatives. We list below the research questions that formed our hypothesis.

RQ1. What are the efficient ways to work with multiple CNNs on conventional hardware resources?

RQ2. How can we make multiple CNNs more energy efficient without losing their effectiveness?

RQ3. How do we measure energy efficiency in comparison to others without requiring specialised devices?

The objectives to achieve by investigating the answers to the above research questions are listed below. Such objectives aid us to test and prove the merits of the hypothesis.

- The objective to achieve by answering RQ1 is to survey the literature involving CNNs that run on conventional hardware and the techniques that are used to make multiple of them run efficiently.
- RQ2 is also the main research question of our research. The objective by researching the answer to RQ2 will generate our methodology to offer a solution to the research problem mentioned in the Introduction.
- The objective regarding the final research question, RQ3, is to devise our evaluation criteria by surveying the techniques that were used beforehand and narrowing them down to the ones that would fit to our research.

1.4 Thesis Scope

This thesis focuses on feed-forward pretrained CNNs that process static input data on mainframe[8]. The aim within this thesis is conducting an extensive energy efficiency study on multiple CNNs that are merged and combined in the same body of neural network, hence become multi-functional networks. Multi-functional networks gather a functional repertoire in which each are distinctive from each other in the same network body. For instance, the same CNNs can recognise images, and also classify instrumental audio. This very example is the final product of this thesis.

Our hypothesis concerns making multiple CNNs more energy efficient. Therefore, although there are research that target making singular CNNs more energy efficient, e.g., *pruning the network*, we consider them as beyond the thesis scope and will not elaborate such methods. *TL* (Tan et al., 2018), *multi-task learning* (Zhang and Yang, 2018), and *shared representation learning* are three major fields of ML in ANNs that approximate to our study; and since we mention "multi-functionality" in the previous

[8]Mainframe is used here with the definition "the Central Processing Unit (CPU) and primary memory of a computer" in the *Oxford's English dictionaries*. Throughout the thesis **conventional** and **traditional** terms will also be used regarding the ANNs which point to ANNs that run on mainframe.

paragraph, these approaches may be confused with ours. However, there are significant differences between these techniques and ours. Firstly, all three of them require training while we utilise pretrained neural networks as they are. Secondly, the tasks that the networks will excel need to be similar in all of them. Otherwise, the training would not result satisfactorily (Standley et al., 2020). This second reason definitely excludes training an image classifier network with audio data later on, as we do in this research.

The works that resemble ours usually use the terms such as *merging* (Wu et al., 2019) and *zipping* (He et al., 2018) neural networks in the literature. Our work is also built upon a *merger* research named *NeuralMerger* (Chou et al., 2018). The difference of our work and theirs is that we benefit from different types of neural networks in our research that are known for their efficiency, such as QNNs, BNNs, and SNNs. We apply conversions between such networks which enriches our research and improves our final results.

The reason of naming this thesis as "Understanding Neural Reuse" is two fold. Firstly, because of the neural reuse theory, as this research is inspired by the theory and aims to emulate it by using the same body of network to realise two distinct functions. Secondly, as an analogy, because we *reuse* the pretrained neural networks repetitively: to merge them with another network, and to convert them into three different types of ANNs.

1.5 Novelty and Contributions

The **novelty** of this research lies in our overarching neural reuse emulation that we achieve via pretrained conventional CNNs, and the results we obtain from such emulation. To elaborate, inspired by the neural reuse theory, we combine an algorithmic approach with three different types of energy-efficient ANNs to realise this objective. This section summarises the following contributions of this thesis to the existent literature:

- Based on our literature review, we believe that this is the first time the neural reuse theory, elaborated in Chapter 3, is applied onto conventional pretrained ANNs. Being a natural way to create energy efficient pathways in the living organism's brain, it is important for such theory to be understood and emulated via computational resources; and we make it possible via the most commonly used ANNs, i.e., CNNs. We hope that this research will pave the way for others as well considering the energy efficiency is more vital than ever today.

- Our work concerns multiple neural networks. We, specifically, focus on making not one, but two CNNs energy efficient at the same time as we explain in Chapter 4. We prove in Chapter 6 by the results of our experiments that without hurting the accuracy of multiple models, a smaller network size and faster inference time is possible.

- We first merge multiple networks into a single network in order to decrease the overall network size while providing multi-functionality to the merged network. Thereafter, we convert the output network into a more energy efficient model which further decreases the model size. Both of these steps provide a compaction to a multi-functional network and the resulting model can fit into edge devices where the memory of the system is very limited.

- We conduct various experiments over a set of different metrics on QNNs, BNNs, and SNN models. This is the first research within our knowledge to utilise two different network generations: second generation of ANNs and the third generation of ANNs which is explained in Chapter 2, in order to quantise, binarise, and also convert into a spiking format multiple neural networks at the same time. We detail our experiments in Chapter 5.

1.6 Thesis Structure

The outline of this thesis is as follows:

Chapter 2 provides background information on ANNs, starting with how the ANNs emerged conceptually. Explanations and background information on CNNs are given next, followed by special types of ANNs such as quantised and binarised neural networks. This chapter finishes with explanations on SNNs. All these types of networks are covered in the background since it is essential to better understand the study and the outcome of this thesis.

Chapter 3 presents the neural reuse theory starting with the biological aspects of how neural reuse occur in nature. It involves explanations on how this biological phenomenon can be translated into a computational paradigm and why it is beneficial to use such approach in neural networks.

Chapter 4 describes our approach of providing multi-functionality to CNNs as well as solutions to decrease the model sizes of neural networks while providing much needed computational efficiency. We present how the multi-functionality is enabled in CNNs which also enables model compaction. We also describe how further optimisations can be achieved from the computational perspective by converting the multi-functional model into different types of neural networks.

In Chapter 5 we explain the experimental setup, discussing the details of our experiments and giving information on the datasets. Neural network architectures and the training details we have used in this thesis to support our hypothesis are also elaborated in this chapter.

Chapter 6 provides information on our evaluation metrics and the experimental results. We show how our approach performs especially from model size and inference time perspectives.

The conclusions on the work presented in this thesis and directions for future research are given in Chapter 7.

Chapter 2

Background I: Artificial Neural Networks

The energy itself starts within the atomic level in nature. Considering our work relies on the principle of conserving energy, we believe one should start from the very beginning, from the "atomic" phase of the concerning topic, that is ANNs in this thesis.

In this Chapter, we provide a technical background of the ANNs. Our background review starts from the origins of feed-forward ANNs. Because our research concerns energy usage, we examine ANNs from all compatible aspects regarding the energy. Therefore, different generations, types, and architectures of ANNs which concern our hypothesis are explained within the Chapter.

To put it simply, ANNs are algorithms to solve computational problems. By reviewing their background, we also put forward and highlight what causes overhead in these algorithms, what makes them power-hungry, ergo, turns them into environmentally costly in this Chapter.

2.1 Generations of Artificial Neural Networks

The history of ANNs starts with single artificial neuron emulations inspired by biological neurons (McCulloch and Pitts, 1943). Several artificial neuron prototypes were put forward in time which led to *generations* of ANNs to come about. There are three generations of ANNs. The generations are decided based on how neurons in the ANNs process input and output signals. Note that, artificial neurons have also been addressed as *neural*, *computational* or *processing* units in the ANN literature. The all three generations are illustrated by providing numerical details in a diagram in Figure 2.1.

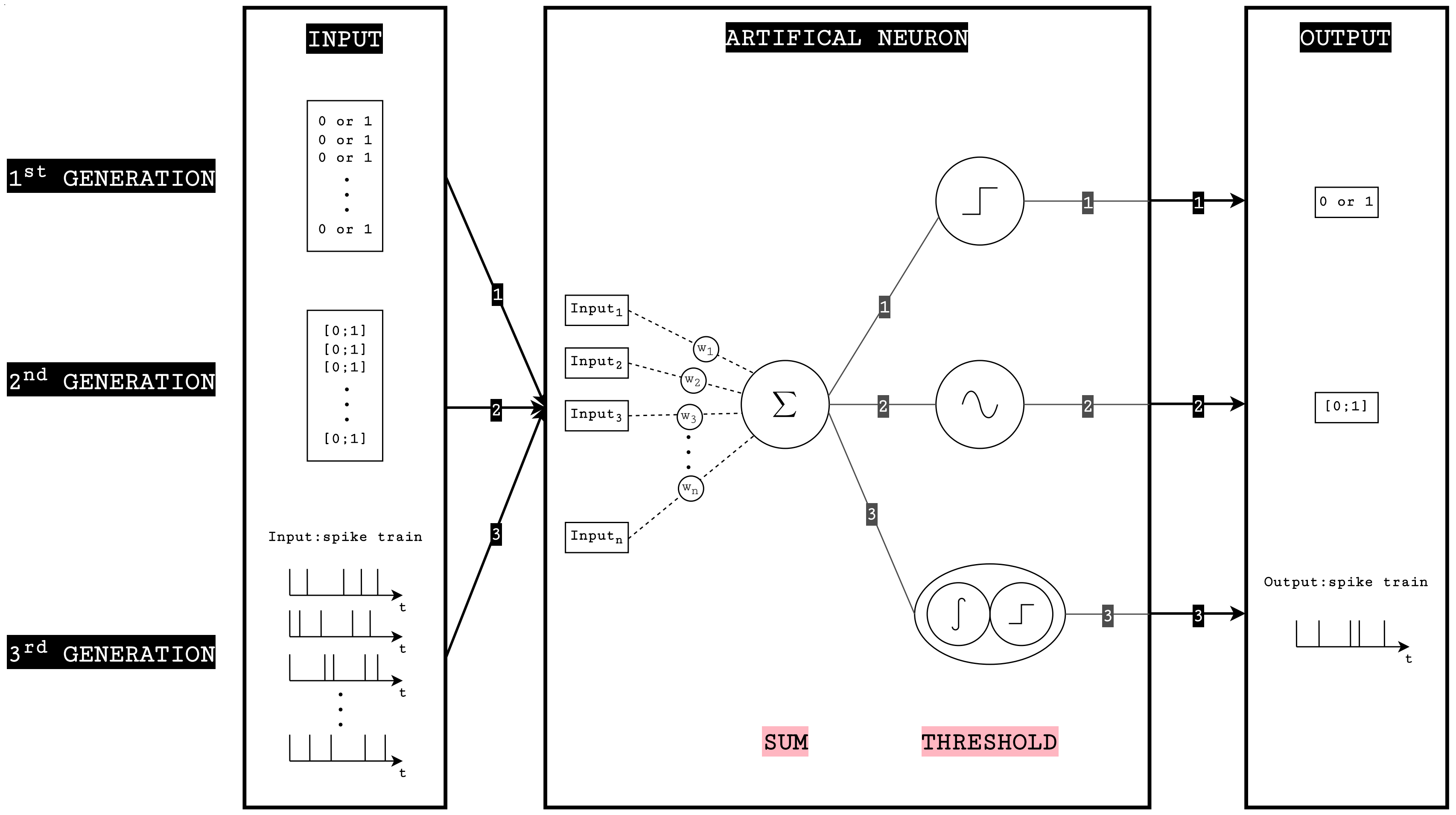


Figure 2.1: Generations of artificial neurons[1].

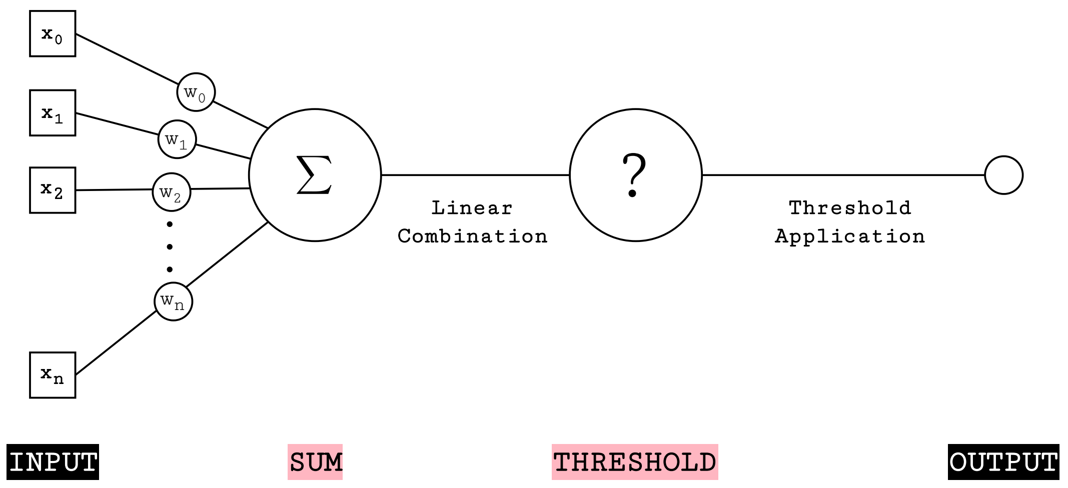


Figure 2.2: Structure of an artificial neuron[2].

Before moving on to explaining the generations, we would like to portray the structure of an artificial neuron via Figure 2.2. Any artificial neuron, inspired by a biological neuron, goes through two steps: combining its weighted inputs and applying a threshold onto them which yields an output at the end. In Figure 2.2, variable x represents the input values ranging in an index of 0 to n, and variable w represents the weights of each input with the same index range as x, from 0 to n. Firstly, the inputs and their correspondent weight values are put into a *linear combination* method, also called as *linear unit* or *sum unit*, i.e., *sum* in Figure 2.2. Thereafter, a threshold is applied onto the linear combination. Finally, the output is obtained. The two operations that are shown in the middle of the Figure 2.2, *sum* and applying a *threshold* are still common today in all artificial neurons. Every artificial neuron receives an *input*, calculates a *linear combination*, applies a *threshold* onto the combination, and produces an *output*. The aforementioned steps are also illustrated in Figure 2.1 which displays the steps that the three generations of artificial neurons follow in detail.

Because the threshold function varies and is the key to differentiate the generations of artificial neurons, it is represented with "?" in Figure 2.2. In Figure 2.1, the three different approaches to threshold function, also called as *threshold unit*, are showcased via labelled arrows each corresponding to the generation of a neuron. Different input and output types are also labelled in Figure 2.1. Another difference between the generic neuron figure, i.e., Figure 2.2, and the generational figure, i.e., Figure 2.1, is the *Input* labels. In Figure 2.1, they are located in the "Artificial Neuron" part and sorted with the

[1]The figure is adapted from "Learning in Spiking Neural Networks" by Davies (2012, p. 24).
[2]The figure is adapted from "Machine Learning" by Mitchell (1997, p. 96).

name *Input* ranging in an index of 1 to n. However, in Figure 2.2, both the input labels, i.e., x, and the index ranges of inputs and weights, i.e., starting from 0 instead of 1, differ. The reason for that is mathematical equations that will clarify the function of an artificial neuron in the following sections are based on the variables in the Figure 2.2.

The 1st generation involve the neural networks that make use of *McCulloch-Pitts* neurons. These neurons had fixed thresholds and only worked with *binary* values, i.e., 0 and 1. Therefore, the first generation ANNs are also called as *gated neural networks*. Such neural networks were utilised to emulate biologically inspired simple experiments as they allowed association between the network layers. However, they lacked the “learning” element. The *Perceptron* that is elaborated in Section 2.2 and some of the Multi-Layer Perceptrons (MLPs) in Section 2.2.2 are among the first generation ANNs in the literature. In Figure 2.1, the arrows labelled as 1 refer to the route that the neurons of the 1st generation follows.

The 2nd generation ANNs uses artificial neurons that transform input signals non-linearly, hence outputs them as continuous values. In other words, ANNs moved from linear to non-linear transformation functions, ergo, binary outputs turned into continues outputs within this era. This era is believed to begun in 1985 when the Back-Propagation (BP) algorithm was applied to ANNs[3] (Rumelhart et al., 1985). BP allowed ANNs to “learn” new tasks. The BP algorithm is still the backbone of ANNs in learning today. CNNs in Section 2.3, QNNs in Section 2.4, and BNNs in Section 2.4.1 are 2nd generation ANNs that will be discussed in the thesis. In Figure 2.1, the behaviour of 2nd generation neurons are represented with the arrows labelled as 2.

The 3rd generation ANNs are called as Spiking Neural Networks (SNNs), and as the name suggests they comprise of spiking neurons. These neurons work based on several parameters all of which are inspired by biological spiking neurons. Features of the spiking neuron, for instance, its activation potential, i.e., spiking, or *firing*, threshold, and how long the spike would last, depend on the values of the parameters. Considering the activation potential of the neuron, on which one might argue that the most important feature of such neuron, spiking neurons are also called as *Integrate and Fire* neurons (Maass, 1997). Since the main paradigm of SNNs is to simulate and approximate to the biological neural networks, neuron models have the significant importance here. Therefore, an SNN may include different types of neurons in different layers of the network based on their tasks. Hodgkin and Huxley, *Leaky Integrate and*

[3]The idea of a BP algorithm dates back to 1960’s, firstly published in 1970. Later, its use on ANNs were proposed by Werbos. However, the first BP learning application on ANNs was in 1985, thus we review it that way.

Fire (LIF) (Dayan and Abbott, 2001), and Izhikevich neuron models are among the most commonly used spiking neuron types. SNNs have a special feature named Spike Time Dependant Plasticity (STDP) which allows them to learn new tasks (Dayan and Abbott, 2001). The 3$^{\text{rd}}$ generation ANNs resemble the 1$^{\text{st}}$ generation from the acts of neurons and efficient use of the hardware resources points of view (Ghosh-Dastidar and Adeli, 2009). In Figure 2.1, the arrows labelled as 3 indicate to the route that the 3$^{\text{rd}}$ generation neurons follow. SNNs will be further discussed in Section 2.5.

* * *

The 2$^{\text{nd}}$ and 3$^{\text{rd}}$ generations of ANNs are in use today, and accordingly, different platforms and software resources are utilised for their applications. The platform and software resources of the 2$^{\text{nd}}$ generation are discussed along with our own choice in Chapter 5 - Section 5.1. Such resources for the 3$^{\text{rd}}$ generation are described in Section 2.5.

The single artificial neuron experiments, or neuron types as mentioned in the preceding paragraphs, is beyond the thesis scope. We focus on artificial neural systems consisting of several neurons separated by layers, i.e., ANNs, in the following sections.

2.2 Perceptrons

“...if one understood the code or wiring diagram of the nervous system, one should, in principle, be able to discover exactly what an organism remembers by reconstructing the original sensory patterns from the memory traces which they have left, much as we might develop a photographic negative, or translate the pattern of electrical charges in the memory of a digital computer.” Rosenblatt (1958)

The *Perceptron* which was introduced as a “hypothetical nervous system” (Rosenblatt, 1958) was the first computational form of a neural network. The creator of *Perceptron*, Rosenblatt, aimed to emulate and hopefully decipher a peripheral sensory organ via this work. The sensory organ was human eye, thus it was also called as “photo-perceptron” in the 1958 paper. It is important to emphasise that the first ever ANN initiative targeted to mimic “vision” which is still crucial for machine learning today and is the fundamental experimental subject of this thesis.

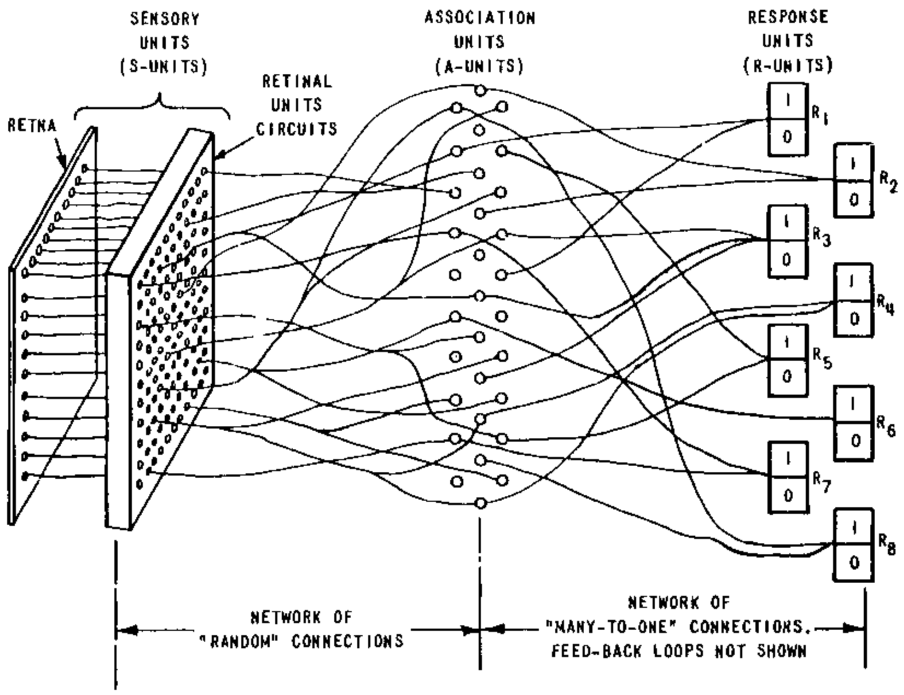

SENSORY UNITS (S-UNITS)
ASSOCIATION UNITS (A-UNITS)
RESPONSE UNITS (R-UNITS)
RETNA
RETINAL UNITS CIRCUITS
R_1
R_2
R_3
R_4
R_5
R_6
R_7
R_8
NETWORK OF "RANDOM" CONNECTIONS
NETWORK OF "MANY-TO-ONE" CONNECTIONS, FEED-BACK LOOPS NOT SHOWN

Figure 2.3: Structure of the first ANN, the analog machine *Mark I Perceptron* by Hay et al. (1960, p. 4).

2.3 is the original illustration of the *Mark I Perceptron* by Hay et al. (1960, p. 4). The *Mark I* machine consisted of three main components: A vision sensor (*S-Units*), an input layer (*A-Units*), and an output layer (*R-Units*). This threesome structure is also illustrated in Figure 2.7a. The network comprising *Mark I* had overall 16 analog neurons. Because of the analog structure of the machine, the voltage capacity of every unit, e.g., [0, 100] for *A-Units*, and the power consumption of every step was clearly observable. The machine was able to "associate" as an ANN despite of the limited setup thanks to the back and forth communication between the A and R Units.

Due to its association ability, Perceptron and Perceptron-like machines are also called as "pattern regularity detectors", yet not as *recognisers* caused by their lack of *learning* element. *Perceptron* paved the way and inspired many for more advanced Perceptron designs which would later be called generally as MLPs or multi-layered networks.

2.2.1 The “Learning” Element

Rosenblatt anticipated the *wiring diagram* to be the key element into deciphering a biological neural network. The *wiring diagram*, also known as connectome in a neural network maintained its significance to decipher, and mystery until the very first biological connectome was revealed. It is of a nematode, *C.Elegans*, a transparent microscopic worm whose body comprises 302 neurons (White et al., 1986; Larson et al., 2018). The problem then emerged that even if we knew the exact connectome of a neural network, the puzzle still wouldn’t be complete, and the network still wouldn’t be in action unless the weights of the connectome are known (Rakowski and Karbowski, 2017; Cohen and Denham, 2019). The weight solving is still burdensome on computational systems today whether it is a simulation of a living being (Sarma et al., 2018) or an ANN learning a new task hence needing the exact weights for its connectome.

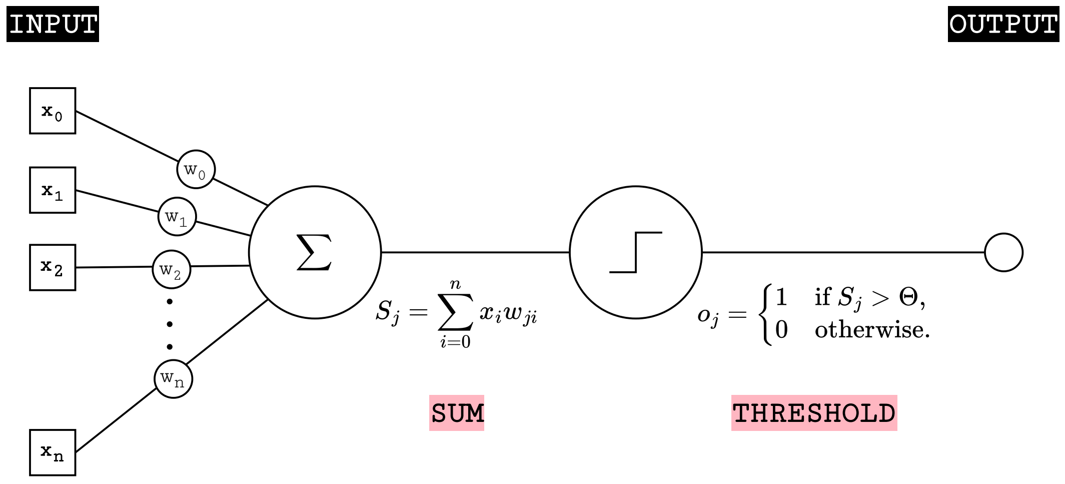

INPUT
OUTPUT
x_0
x_1
x_2
x_n
w_0
w_1
w_2
w_n
Σ
$S_j = \sum_{i=0}^{n} x_i w_{ji}$
$o_j = \begin{cases} 1 & \text{if } S_j > \Theta, \\ 0 & \text{otherwise.} \end{cases}$
SUM
THRESHOLD

Figure 2.4: The function of a *Perceptron* neuron.

The neurons used in the *Perceptron* were McCulloch and Pitts neurons and they responded to a linear threshold function to yield an output. Figure 2.4 is the updated version of Figure 2.2 via mathematical equations (Estebon, 1997) that define the function of a *Perceptron* neuron. The equations in the Figure 2.4 is also shown below respectively in Equation 2.1 and 2.2.

$$S_j = \sum_{i=0}^{n} x_i w_{ji} \tag{2.1}$$

$$o_j = \begin{cases} 1 & \text{if } S_j > \Theta, \\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

Equation 2.1 is a linear input function, a weighted sum of the input set where each item is symbolised as x and indexed via i. Equation 2.2 consists of a case which acts like a logic gate that outputs a binary result based on a threshold, Θ. The output range is represented via j in both equations. The result of the *sum* unit, Equation 2.1, S_j, is fed into the threshold unit, Equation 2.2, which yields the predicted output of the artificial neural unit that is symbolised as o_j.

Next to a *linear unit* and a *threshold unit*, Perceptron also had a convergence procedure, addressed as "error correction training technique" by Hay et al., to train, hence improve the performance of the network. The error correction formula is shown in Equation 2.3 (Estebon, 1997; Mitchell, 1997). In the below equation, Equation 2.3, C is a constant representing the *learning rate* of the neural network assigned by the user; the definition of x, w, and o symbols remain the same as the previous equations; and t represents the "target" output, the desired and/or real output of the function.

$$w_{ji_{new}} = w_{ji_{old}} + C(t_j - o_j)x_i \tag{2.3}$$

The training technique in the Perceptron was based on comparing the predicted output, o_j, and the target output, t_j. When the two outputs did not agree, an update to the weight of the relevant input was made based on the difference in the outputs multiplied by the learning rate constant, C in Equation 2.3. Although the procedure aided to increase the performance of the network, it was still limited as a "learning" method due to the ideal circumstances it required (Rumelhart et al., 1986; Estebon, 1997; Mitchell, 1997). Firstly, the training data of the network needed to be linearly separable. Secondly, the C value needed to be sufficiently small, ideally close to 0. Otherwise, the convergence was not guaranteed.

2.2.1.1 Introducing Back-Propagation Learning

The Perceptrons were found inauspicious back then (Minsky and Papert, 1969) due to their limitations. Moreover, the criticism over the Perceptrons caused a cut in the funding of the related work and put an hold on the work over ANNs until the application of *Back-Propagation (BP)*[4] learning algorithm onto ANNs (Rumelhart et al., 1985)

[4]The term "Back-Propagation" is derived from *backward error propagation*. It is commonly referred as Back-Propagation or Back-Prop in the literature. We will refer it as BP in short in the thesis.

which showcased that they were able to learn in a supervised way. With the BP, the 2nd generation of ANNs also begun as the BP was designed to work with differential, also called as nonlinear, threshold units instead of the linear units. Figure 2.5 is the updated version of Figure 2.2 via mathematical equations that define the function of a 2nd generation artificial neuron (Rumelhart et al., 1986; Mitchell, 1997).

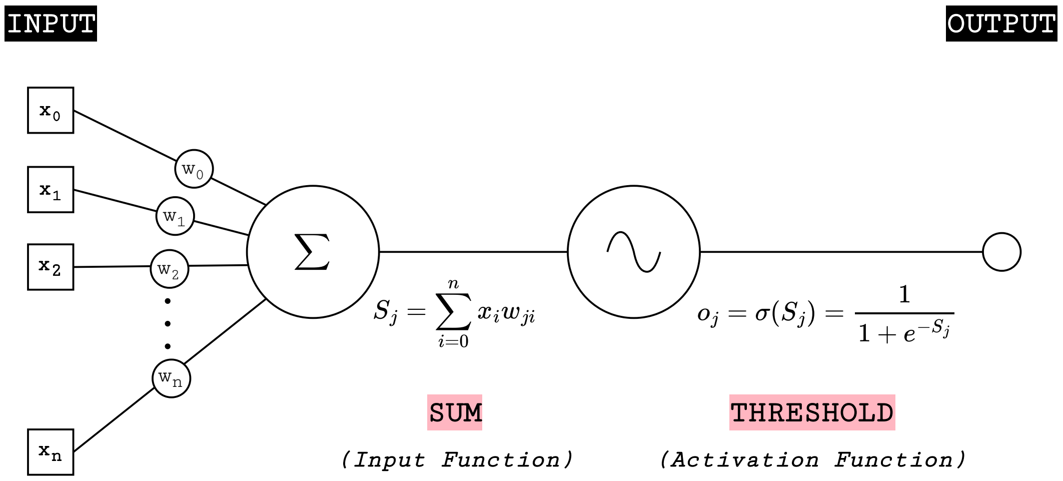


Figure 2.5: The function of a 2nd generation artificial neuron.

The modern terminology to address the linear function, labelled as *sum* in Figure 2.5, is *input function*; and it is *activation function* for the *threshold* function in Figure 2.5. Various activation function types are available to use today. However, in the beginning it was the *Sigmoid* function. Therefore, the threshold function we display in Figure 2.5 is also the sigmoid. The differential equation belonging to sigmoid (Rumelhart et al., 1986; Mitchell, 1997) in the Figure 2.5 is also shown in Equation 2.4.

$$o_j = \sigma(S_j) = \frac{1}{1+e^{-S_j}} \tag{2.4}$$

Although the training technique of Perceptron was not sufficient to be called entirely as a learning technique, the BP was built on it and used the same fundamental principles in terms of comparing the two outputs, measuring their difference, and updating the weights accordingly. BP triggers learning by updating the weights of neurons by "*backward propagating*" the errors caused by the mismatch between the predicted output and the target output values of the network. The BP mechanism is illustrated in Figure 2.6.

What makes BP algorithm progressive compared to the prior approach is its capability to work on linearly non-separable training data as well, and it does it via *delta*

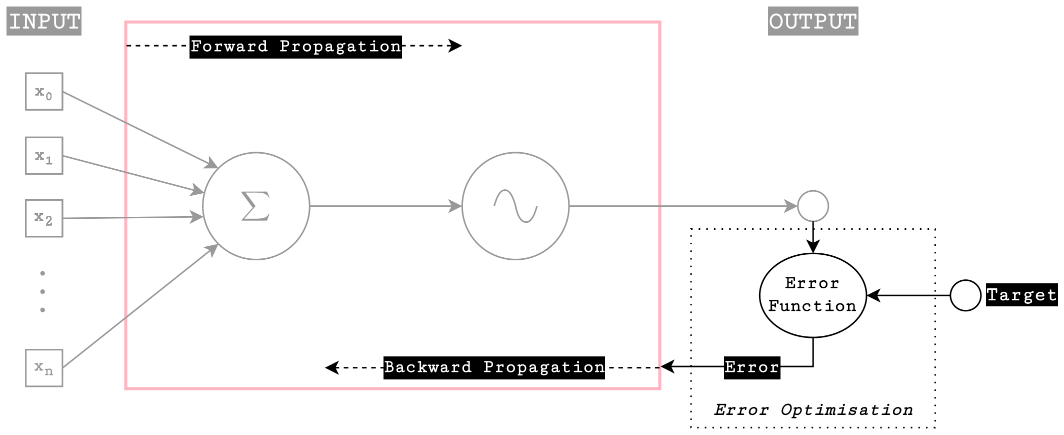


Figure 2.6: The mechanism of Back-Propagation learning.

rule. The delta rule, for the linearly non-separable data, involves a search of *gradient descent* to find the best candidate of a weight value to minimise the error. The "descent" here refers to its top-to-bottom search style to find the steepest point in the search space. In the previous paragraph, we have mentioned that the fundamentals of the both approaches remained the same. The learning mechanism of Perceptron rather approaches simplistically to updating the weights. In the second part of the Equation 2.3, on the right of $+$, we see the error calculation formula of the system. The error calculation rule[5] for a single neural unit chosen for the BP algorithm (Rumelhart et al., 1985) is shown as follows:

$$E(t,o) = \frac{1}{2}\sum_j (t_j - o_j)^2 \tag{2.5}$$

BP aims to find the most optimal value for the weight update which would lower the error via gradient descent, yet it might not entirely fix or *correct* the error. However, BP optimises the error. Therefore, we name the error checking unit as *error optimisation* in Figure 2.6 instead of error correction. The optimisation is achieved via the gradient descent rule that mathematically involves partial derivatives. The initial partial derivative of the gradient descent (Rumelhart et al., 1986) is as follows:

$$\frac{\partial E}{\partial w_{ji}} \tag{2.6}$$

[5]The chosen error rule here is a modified version of Mean Squared Error (MSE). We also use MSE to evaluate our approach whose formula is shown in Chapter 6 - Section 6.1.1.

In Equation 2.6, E refers to the error formula in Equation 2.5. The overall formula in Equation 2.6 targets to minimise or eliminate the error caused by the value of the weight, w_{ji}. When chain rule is applied onto the partial derivative in Equation 2.6, the ultimate result would be

$$\frac{\partial E}{\partial w_{ji}} = -(t_j - o_j)\sigma'(S_j)x_i \tag{2.7}$$

In Equation 2.7, σ' refers to the derivative of the activation function of the neuron. The activation function, i.e., Sigmoid, was illustrated in Figure 2.5, and its formula was also showcased in Equation 2.4. S_j refers to the input function of the neuron who was also was illustrated in Figure 2.5 and whose formula was shown in Equation 2.1. Finally, the x_i refers to the input value. After computing the partial derivatives, the formula for the delta rule is obtained as follows

$$\Delta w_{ji} = \alpha(t_j - o_j)\sigma'(S_j)x_i \tag{2.8}$$

The delta, i.e., Δ, rule specifies how the regarding weight, i.e., w_{ji}, would be updated. Similar to the C in Equation 2.3, α in Equation 2.8 also represents the learning rate and is chosen as a small constant. The Δ value is added onto the specific weight to modify it. As a final comparison, we add the delta rule of Perceptron below, by exchanging C with α this time, which was designed for the linearly separable data unlike the BP.

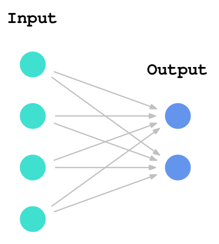
$$\Delta w_{ji} = \alpha(t_j - o_j)x_i \tag{2.9}$$

When all the required weights are updated in a network via BP, the network is accepted as "converged", and learnt the given task. The illustrations, equations, and discussion in this section are over a single artificial neural unit[6], yet the BP is designed to enable learning for any scale of neural networks which we would discuss in the following sections. To consider the mechanism with multiple neurons and layers visually, simply the pink box in Figure 2.6 can be filled with the pink coloured nodes in 2.7b. With the addition of BP onto ANNs, their neuron and layer counts increased which aided the networks to learn more diverse tasks. This is also the start point where the ANNs begun to be computationally expensive.

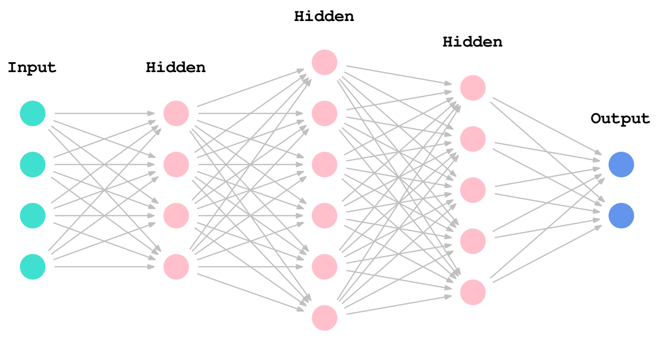
[6]In multi-layered ANNs having more than one hidden layer, on which we discuss starting with Section 2.2.2, BP algorithm applies two different approaches depending on a neuron being in a hidden layer or being an output layer neuron. Because the BP algorithm is out of the thesis scope, we will not further elaborate BP and will leave it as general background information here.

2.2.2 Multi-Layer Perceptrons

The Multi-Layer Perceptrons (MLPs) were the first formed feed-forward ANNs. They are able to process any input type as long as the input is vectorised such as text, image, and sound data. An MLP consists of dense, i.e., fully-connected, layers that are feed-forward from input to output. MLPs are especially applied to classification and regression problems.



(a) Perceptron



(b) Multi-Layer Perceptron

Figure 2.7: *Perceptron* illustrations. **(2.7a)** illustrates the original *Perceptron, Mark I,* where the input layer was directly connected to the output layer without having any hidden layers in between. **(2.7b)** exemplifies an MLP which typically has one or more hidden layers that are illustrated as pink coloured nodes in the graphics.

An MLP is an ANN in which all the neural units, i.e., neurons, in a layer are fully connected to the units of the following layer. Because of their fully-connected nature, along with *Fully-Connected (FC)*, MLPs are also called as *dense* networks. An MLP includes at least one hidden layer as shown in Figure 2.7b. The first Perceptron model did not include any hidden layers. In the Perceptron, the output of vision sensor was fed into the input layer which corresponds to the first layer in Figure 2.7a. Furthermore, the input layer was directly connected to the output layer, which corresponds to the final layer in Figure 2.7a, in the original Perceptron. The difference between the Perceptron and the MLPs is displayed in Figure 2.7.

The first MLP was *Cognitron* which was an extended version of Perceptron aiming to emulate the self-organising scheme and plasticity of the human brain (Fukushima, 1975). Cognitron used analog neurons that have the excitation and inhibition features as in Perceptron, inspired by the biological neurons. The Cognitron took place prior to the invention of BP, hence it is one of the 1st generation ANNs. Afterwards, with the applications of BP on ANNs, conserving the energy in neural networks has become more and more difficult. We overview what made the 1st generation more energy-efficient below.

- **Computing style:** Analogue computing that allows manual configuration of the variables where necessary.
- **Metric measurement:** Transparent, more accurate measurement of metrics. For instance, the overall consumed energy could be calculated via voltage readings from the machines.
- **Neuron types:** Simplistically designed, yet biologically inspired, energy-efficient neurons.
- **Learning algorithm:** Simplistic error correction techniques that allows association between the layers.

NETtalk, a 2nd generation MLP, was the first ANN to which the BP learning technique applied (Sejnowski and Rosenberg, 1986). It had one hidden, overall three layers; its task was converting texts in English into speech. NETtalk reached a 95% best guess rate[7] after 12-hours long training with the BP (Estebon, 1997). The highest best guess rate achieved via BP learning on NETtalk, reported by Sejnowski and Rosenberg, was 98%; whereas without the BP learning, the best guess rate decreased down to 77% which showcases clearly the importance of the BP learning in ANN success rate. Among the later, yet prominent, examples of MLPs are a smell recogniser, the *Artificial Nose*, network (Santos et al., 1998); and two of the first Graphics Processing Unit (GPU) applications of ANNs (Oh and Jung, 2004; Steinkraus et al., 2005).

MLPs were initially utilised for simulating biological systems where association mattered (Fukushima, 1975; Santos et al., 1998). Later on, they were utilised for prototyping and used as proof-of-concept, for instance the leading GPU applied MLPs (Oh and Jung, 2004; Steinkraus et al., 2005). Today, MLPs are still in use for prototyping. However, we mostly see them as part of other ANN types, e.g., CNNs, especially as constituting their final layers. MLPs that are used in such positions are called with layer names that are *Dense* or *Fully-Connected*.

[7]The NETtalk paper defines its own metrics which are "perfect match" and the "best guess". Best guess refers to the closest output to the original output (see Sejnowski and Rosenberg, 1986, p. 665, §Performance).

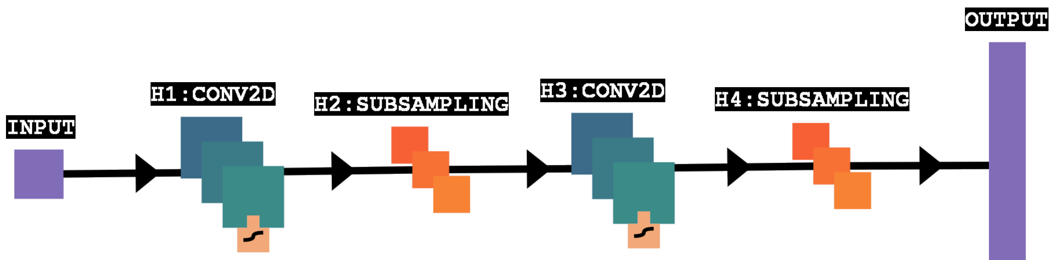
2.3 Convolutional Neural Networks

Inspired by the mammalian visual system, Fukushima invented *Neocognitron* to solve the visual pattern recognition problem via unsupervised learning with ANNs. Neocognitron, like the *Perceptron* machine, was built to emulate the visual processing between the eye and the brain. The difference between the two aforementioned touch-stone machines was the level of detail in Neocognitron. Different biological neuron groups, based on their tasks in visual processing, were emulated. These groups were divided into layers in the corresponding neural network, and specific filters were applied onto those layers to get the desired output. Applying the filters were called as "invariant shifting" in the system which then addressed commonly, yet is addressed as the **convolution** operator today based on the correspondent *Math* function. Neocognitron is a 1^{st} generation ANN and the first Convolutional Neural Network (CNN) in history.

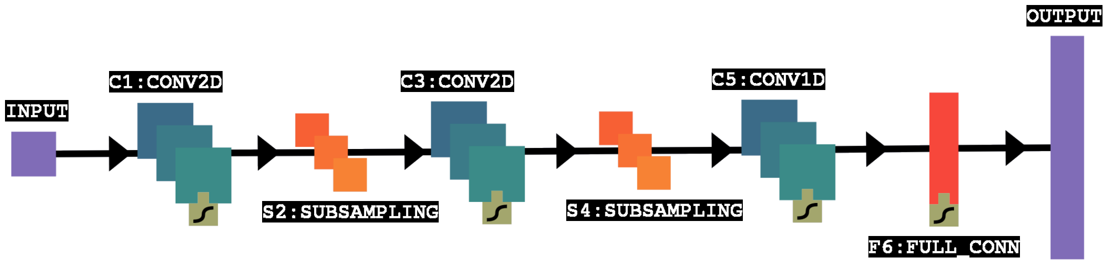
Addressing with the today's terminology, each convolved network layer generates **feature maps**. The feature maps represent the detected patterns, i.e., *features*, in the input, hence are crucial for the network's outcome. Therefore, the features located in each layer's maps, i.e., local feature maps, should be globally reachable and recognisable by the network. This is achieved via *weight-sharing* in a CNN and it was firstly applied on Neocognitron. The "weight-sharing" technique of Neocognitron was one of the inspirations leading to the BP algorithm (Rumelhart et al., 1985). It also inspired the first BP applied CNN[8] (LeCun et al., 1989). An important point here is that, the reason of moving to CNNs from the available early ML techniques was not only because they were biologically inspired solutions to the visual pattern recognition problem, but also because the earlier ANN solution, which were MLPs, was not a good fit to the problem as they "generate too many parameters to generalise accurately" (LeCun et al., 1989, 1990).

The architectural scheme of CNNs that still in application today initially appeared in LeNet-1 architecture (LeCun et al., 1990). The LeNet-1 comprises two sets of $Convolution \rightarrow Subsampling$ layers following each other as displayed in Figure 2.8a; these sets constitute the signature layering of CNNs. The *H* codes in Figure 2.8a represent *Hidden layer* as in the correspondent paper (LeCun et al., 1990). The training time for LeNet-1 was reported as three days back then (LeCun et al., 1995).

[8]This early CNN by LeCun et al. only utilises Convolutional and Fully-Connected layers and does not include Subsampling, i.e., Pooling, operation between the layers as later the modern CNN architectures adapted.



(a) LeNet-1



(b) LeNet-5

Figure 2.8: Architectures of the LeNet-1 and LeNet-5 CNNs. **(2.8a)** illustrates the architecture of the LeNet-1(LeCun et al., 1990, p. 401), the first official CNN. **(2.8b)** illustrates the LeNet-5 architecture (LeCun et al., 1998, p. 7), the ancestor of modern CNNs.

Later on, in LeNet-5, FC layers were also added to the LeNet-1 architecture as final layers of the network to improve the error rate (LeCun et al., 1995, 1998). LeNet-5 architecture is shown in Figure 2.8b. The task that were taught to the LeNet networks was handwritten digit recognition to be used on the postal codes by BP via the data supplied by US Postal Services. The training time for LeNet-5 was reported as two weeks in 1995, and as two to three Central Processing Unit (CPU) days three years later by LeCun et al. (1998). Although Fukushima did not use BP, his works (Fukushima, 1980; Fukushima and Miyake, 1982) aiming to simulate how mammalian visual system processes patterns inspired LeNet architectures in terms of:

– *The convolution operator*, to obtain feature maps that detect different patterns,
– *Weight sharing*, between the feature maps,
– *Convolutional and subsampling layers*, that follow one another.

Convolutional layers are to detect the features in the input; and **subsampling** layers are to combine the representative features together and minimise the count by eliminating some of them (Lecun et al., 2015). Note that subsampling was also addressed as *averaging* during the early papers. Nowadays it is mostly referred as **pooling**.

The modern CNNs have a few more add-ons onto the above list, and these are applying a non-linear function, i.e., activation function, right after the convolutional layer and finishing the CNN with FC layers since they excel in classification (Lecun et al., 2015; Anantrasirichai and Bull, 2022). Although the main application area of CNNs have been "image recognition", they also have been proved to be effective on auditory and language perception as well (LeCun et al., 2010b).

2.3.1 Convolutional Neural Network Architectures

Starting with the LeNet-5, CNN architectures begun to thrive. In the papers where LeNet-1 was summarised (LeCun et al., 1995) and LeNet-5 was introduced (LeCun et al., 1998), Lecun mentions that *"the true potential of ANNs will come true when **more data** is present"*. We have seen the realisation of it, especially with the availability of *ImageNet* (Russakovsky et al., 2015), the biggest image dataset until recently (Sun et al., 2017). For reasons such as achieving higher accuracy in benchmarks, predominantly in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), various CNN architectures were invented.

We overview CNN architectures as *deeper* and *deeper and wider* in the next section. Both of the aforementioned comparatives are based on LeNet-5 here. Neither "deep" nor "wide" is defined by specific number of layers in the ANN literature, ergo we do not quantify them in the thesis. The start point of both architecture types are Deep Neural Networks (DNNs). DNNs, that is discussed in Section 2.3.1.1, are defined as trainable ANNs with free parameters and multiple hidden layers (Anantrasirichai and Bull, 2022). After the success of the newly introduced *deep* ANNs, i.e., DNNs, in terms of high accuracy values obtained, the *wide* DNN concept appeared in the literature to reach even higher accuracy levels, and we discuss it in Part 2.3.1.1. Our overview is over *deep* and *wide* CNN architectures separately to provide more insight, yet the both concepts are regarded as, and under the umbrella of, DNNs in the ANN literature.

2.3.1.1 Deeper Architectures

After LeNet-5 (LeCun et al., 1998), the first renowned CNN became *AlexNet* (Krizhevsky et al., 2012) as being the winner of ILSVRC in 2012[9]. AlexNet was heavily influenced by LeNet-5 by having 5 convolutional, overall 8 layers stacked in a similar order. It became the pioneer of the modern CNNs by including REctified Linear Unit (ReLU) as an activation function and adding *Dropout* layers to the architecture to avoid overfitting of the data (Russakovsky et al., 2015). Moreover, it was trained via GPU. All of these features of AlexNet paved the way for many more CNN architectures to appear.

ZFNet (Zeiler and Fergus, 2014), the winner of ILSVRC13; then *VGG-16* (Simonyan and Zisserman, 2015), runner-up of ILSVRC14 followed the footsteps of AlexNet and put forward deeper CNNs, i.e., having more stacked layers. The deeper an architecture becomes based on convolution operations, the more computationally expensive it gets. However, such architectures resulted in lower error rates on ImageNet compared to their counterparts, hence similar architectures have continued to dominate the CNNs literature.

Deeper and Wider Architectures The "wide" CNN architectures started with the Inception-V1 network, also known as *GoogLeNet*, which won the ILSVRC in 2014 (Szegedy et al., 2015). GoogLeNet is a DNN like the previous winners of the challenge. However, it is not only deep but also architecturally wide, which made it unique back then. GoogLeNet has 22 deep layers, yet overall 58 layers when including the wide layers. Such wide layers are called as *Inception modules* in the paper and they consist of several convolutional, i.e., 1×1, 3×3, and 5×5, layers along with a maximum pooling layer. The inception modules work as parallel filters in the model that process a given image input at different scales. The 1×1 convolutions are especially planted with the aim of reducing the weights and, overall, reducing the underlying operations in the model.

[9]Although ILSVRC started firstly in 2010, the first two winners of the challenge presented models containing other ML techniques than ANNs.

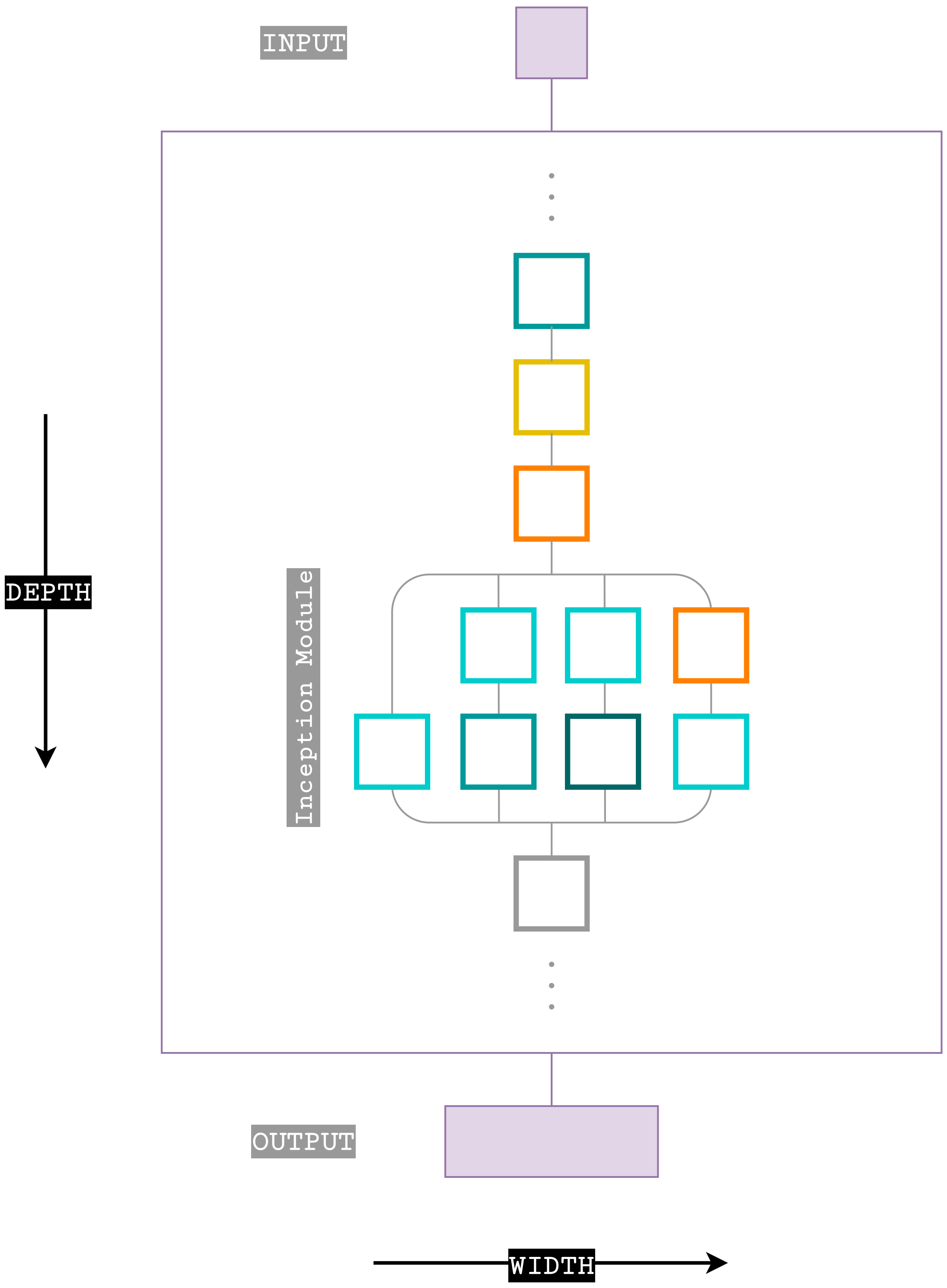


Figure 2.9: GoogLeNet partial illustration.

Figure 2.9 illustrates a part of GoogLeNet including its first inception module. The layers of the GoogLeNet architecture are illustrated as square boxes in the graphics and the starting layer in the graphics is the fifth layer, i.e., a 3×3 convolutional layer, in the original architecture. The colours of the layers are kept the same as in the other graphics of CNN architectures in the thesis, e.g., Figure 2.8 in this Chapter. The details of the partial GoogLeNet illustration in Figure 2.9 are listed below.

– The green/blue boxes that are in three different tones indicate convolutional layers. The lightest blue, i.e., turquoise coloured, boxes in the graphics are 1×1 convolution(operation)s; one tone darker of them, teal coloured boxes, are 3×3 convolutions; and the darkest coloured of them is a 5×5 convolution that is seen as only one in the graphics placed in the inception module.

– The orange boxes indicate subsampling layers where the chosen subsampling method is maximum pooling in GoogLeNet. We discuss subsampling including maximum pooling in Chapter 5 - Section 5.3.1.1.

– The mustard colour box indicates a normalisation layer. The normalisation method preferred for GoogLeNet was *Local Response Normalisation (LRN)*. LRN was used in and firstly introduced by AlexNet (Krizhevsky et al., 2012). We discuss layer normalisation in Chapter 5 - Section 5.3.1.1 over an up-to-date technique that is Batch Normalisation (BN), and do not provide details on LRN. However, an important difference that we would like to mention between LRN and BN is while BN layers are trainable, LRN layers are not, which affect the performance of a CNN.

GoogLeNet reduced the Top-5 error rate ≈50% less than the winner of the previous winner, ZFNet. After such success of GoogLeNet in 2014, *ResNet* (He et al., 2016), *SENet* (Hu et al., 2020), and *PNASNet-5* (Liu et al., 2018) followed the same approach in building “deep and wide” CNN architectures and won the ILSVRC in 2015, 2017 and 2018 respectively. These architectures continued to lower the error rate, yet another big decrease in error rates, like the GoogLeNet achieved, did not occur again.

GoogLeNet was using inception modules, as seen in Figure 2.9, to encapsulate the wide branching in the architecture with the aim of reducing computation and size of the model. The CNN architectures mentioned in the previous paragraph applied similar techniques, such as the “Shortcut modules” in ResNet, also known as residual blocks; “Squeeze and Excitation (SE) blocks” in SENet; and “cells” in PNASNet-5. The deep and wide architectures, and mix and match type of approaches still in demand in the image classification literature. Currently[10] the Top-1 accuracy holder in ImageNet is *CoAtNet-7* (Dai et al., 2021) which is a mixed architecture including CNNs and *attention* layers that constitute *Transformer* type of networks[11].

* * *

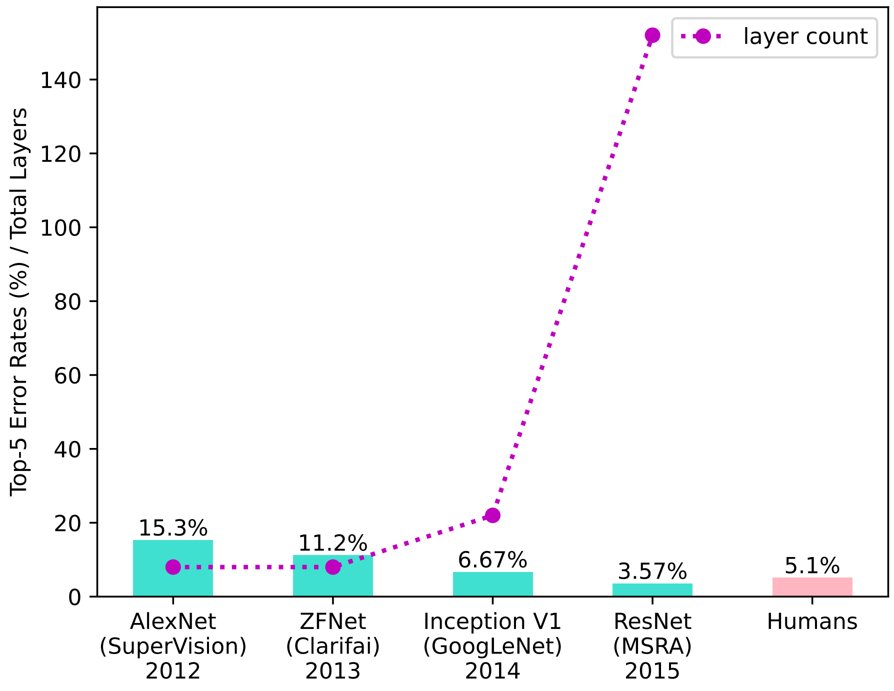


Figure 2.10: ILSVRC winner models between 2012 and 2015 vs. humans in image classification.

Figure 2.10 showcases the accuracy vs. layer counts of the CNNs within the ILSVRC. The original LeNet-5 is a 7-layer CNN. AlexNet and ZFNet have 8; GoogLeNet has 22; and ResNet, which outperforms humans based on error rate in image recognition, has 152 layers.

These architectures proved themselves regarding accuracy in image recognition as shown in Figure 2.10. However, firstly their getting deeper, then wider, overall bigger resulted in the energy draining problem of the CNNs.

[10]Lastly checked on 15/12/2021 via `https://paperswithcode.com/sota/image-classification-on-imagenet`.

[11]Transformers are known to excel in the *Natural Language Processing* field today. We do not delve into the Transformers in this thesis as it is out of our scope. However, more details can be found in "Attention is all you need" by Vaswani et al..

2.4 Quantised Neural Networks

The rising popularity of DNN models require efficient and accurate inference schemes, even on mobile and low-power edge devices. Given the computational complexity of the *Full bit Precision-Floating Point (FP)* operations on real hardware, using a quantised model significantly increases the efficiency. Figure 2.11 illustrates the relation between CNNs, which was the topic of the previous section, i.e., Section 2.3, and Quantised Neural Networks (QNNs) in terms of the thesis scope.

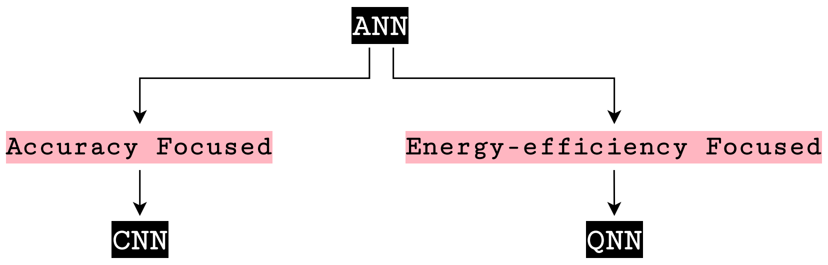


Figure 2.11: The relation between CNNs and QNNs.

Approaches in this field roughly fall into two categories. The first category, exemplified by MobileNet (Howard et al., 2017), SqueezeNet (Iandola et al., 2016), ShuffleNet (Ma et al., 2018), and DenseNet (Iandola et al., 2014), designs novel network architectures that exploit computation/memory efficient operations. The second category quantises the weights and/or activations of a CNN from 32 bit floating point into lower bit-depth representations. This methodology, embraced by approaches such as Ternary weight networks (Li et al., 2016), Binary Neural Networks (Lin et al., 2017), XNOR-net (Rastegari et al., 2016), and many more approaches have shown that increased efficiency can be achieved without sacrificing much from the accuracy of the models. In (Jacob et al., 2018) authors provide a quantisation scheme where floating-point arithmetic is used during training; and integer-only arithmetic is used for inference in order to reach a higher level of correspondence, and thus, improves the latency-vs-accuracy trade-offs. The aforementioned work is especially important since the authors used real mobile hardware to show the efficiency of their approach. The category that we focus in this thesis is the second category.

2.4.1 A Special Case: Binarised Neural Networks

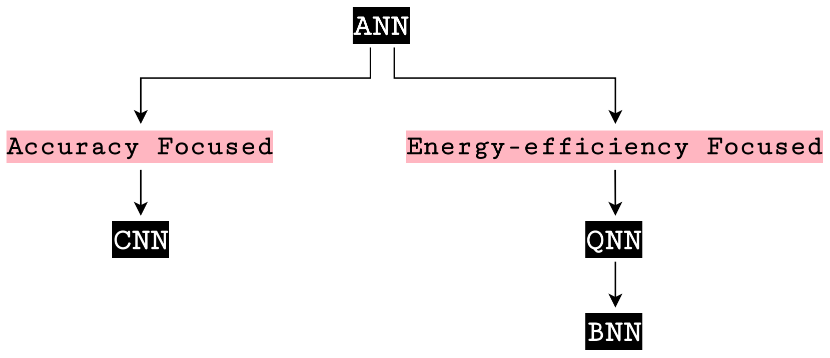


Figure 2.12: The relations between CNNs, QNNs, and BNNs.

Binarisation is a special case of quantisation in neural networks where some or all the arithmetic involved in computing the outputs are constrained to single-bit values. It is also called as *extreme quantisation* because of that (Gholami et al., 2022). In Figure 2.12, we illustrate the relations between CNNs, QNNs, and BNNs. Because BNNs are a subgroup of QNNs, they are categorised under QNNs in the *(Energy-efficiency Focused)* EEF-ANN group in the Figure.

Table 2.1: A comparison of CNNs, QNNs, and BNNs in terms of precision of neural unit weights.

ANN Type	Precision of Weights			
	Float32	Float16	Integer	Binary Only
CNNs	✓			
QNNs		✓	✓	
BNNs				✓

In Table 2.1, a comparison of CNNs, QNNs, and BNNs are displayed over their use of weight precision in artificial neural units. The weights of conventional ANNs are addressed with the term *full bit precision* in the QNNs literature, compared to the QNNs that innately have lower bit precision (Gholami et al., 2022). Therefore, we also address the weights of CNNs accordingly, with the abbreviation FP. The full bit precision regards floating point data in this concept, i.e., data type *float*, *FP32* in computational environment; whereas lower bit precision regards *Float16*, and the fixed integer values that may be varied from *int32* to *int4* depending on the quantisation degree which will be discussed in the following chapters.

Umuroglu et al. define the full binarisation of a neural network as the network having binary input activations, binary synapse weights, and binary output activations. In other cases where one or two layers are binary, it is defined as a partially binarised network. BNNs enable great benefits for decreasing the computational complexity while reducing the model size for the following reasons.

– Binary weights and activations reduce memory usage and model size up to 32 times compared to single precision floating point models.

– If the weights are binarised, Multiply and ACcumulate operations (MACs) can be replaced by simple accumulation operations removing the computational complexity of multiplication.

– If both weights and activations are binarised, MACs can be replaced by bitwise XNOR and bitcount operations which can be executed faster on CPUs compared to floating point operations (Courbariaux et al., 2016).

These outcomes not only increase the performance and power efficiency of the neural networks, but also help decrease the model sizes. However, the main setback of binarisation is the accuracy aspect (Rastegari et al., 2016). Despite the fact that the binarisation of the weights/activations into 1-bit values significantly affects the accuracy of a model, further research is ongoing in order to minimise the loss of accuracy. Kim and Smaragdis consider a predetermined amount of synapses having zero weight and other synapses as a weight of one, where they report 98.7% accuracy with FC layers on the MNIST dataset using XNOR and bitcount operations only.

Lin et al. propose multiple binary weight bases/activations scheme where the authors show that using five binary activations can diminish the reduction in accuracy and achieves prediction accuracy comparable to its full-precision version on ImageNet. Although binarisation enables lots of advantages and the research in the literature try to cope with the accuracy loss of the neural networks, all these proposed methods require the retraining of the existing models. In case a pretrained model's weights and/or activations are replaced with binarised weights/activations, the accuracy drop is unavertable[12].

[12]We also attempted an experiment regarding the subject that discussed in Chapter 5 - Section 5.5.2.1

2.5 Spiking Neural Networks

The first generation and the third generation neural networks resemble each other in terms of ideology and technique. Networks in both aforementioned generations are more biologically inspired and more plausible as simulations. The "time" factor in simulations is mentioned as a must even though at some places is not applied to the network. Connections are called as excitatory and inhibitory, as in the mammalians, that the former excites the regarding neuron causing a higher voltage, and the latter inhibits the neuron's state by causing a decrease in its voltage levels. Due to artificial neuron-wise similarities between the 1st and 3rd generations of ANNs, some studies regard the Perceptron (Hay et al., 1960) as the first example of neuromorphic computing (Davies et al., 2021) which is the general term that is used to represent biologically inspired systems and tools.

The origins of emulating a spiking neuron as in the mammalian brain with the aim of deciphering and understanding led to creating an artificial spiking neuron which is traced back to early 1900's (Maass, 1997). It started with the singular neuron models, as mentioned earlier in the 3rd generation part of the Section 2.1, then turned into networks of spiking neurons, i.e., SNNs. SNNs has been and is still in use to mostly simulate or emulate the mammalian brain which can be used to test the effect of drugs or trace the origins of neural diseases, e.g., Parkinson's disease, computationally (Davies, 2012). A recent cortical simulation consisting of $77,000$ neurons exemplify the current state of SNNs (Rhodes et al., 2020).

Different hardware and software-based resources have been launched to simulate SNNs over the years especially with the start of Human Brain Project (HBP) (Markram et al., 2011; Amunts et al., 2016). *BrainScaleS* (Schemmel et al., 2010, 2022) is an analog and *SpiNNaker* (Furber et al., 2014; Mayr et al., 2019) is a digital neuromorphic computing platforms developed under the HBP. *PyNN* is a high-level software library, an interface encapsulating many software libraries, that developed within the HBP, under the same roof (Davison et al., 2009). Moreover, *Loihi* by Intel (Davies et al., 2021) and *TrueNorth* by IBM (Akopyan et al., 2015) are the industrial neuromorphic processors to simulate SNNs. SNNs work with continuous data forms. In order to supply such data to the SNNs, either the discrete forms are translated into continuous forms (Liu et al., 2016; Garcia et al., 2017), or fed directly from other neuromorphic devices. One of such devices is event-based cameras that produce the visual output as continuous, spatio-temporal data rather than frames (Brandli et al., 2014). Such neuromorphic partnerships generate highly efficient systems and are especially benefited in the field of robotics (Amir et al., 2017; Milde et al., 2017; Basu et al., 2018).

During the recent years, building conversion techniques, from the pretrained ANNs into SNNs[13], to enable running SNNs on conventional hardware have gained momentum (Yamazaki et al., 2022). It is also possible to train SNNs from scratch to run on conventional machines with[14,15,16] or without[17] the help of software libraries. All of these systems that running SNNs on conventional hardware imitate some of the fundamental aspects of their neuromorphic counterparts, such as time-steps. Time-steps can be assigned as a constant in the beginning to set *spike-frequency* (Wu et al., 2022); or can be measured within the system, then are utilised based on observance in terms of system optimisation (Stöckl and Maass, 2021), i.e., *spike-timing*. The concept of time-steps is crucial for a spiking neuron as it specifies how frequently it might spike given the sufficient conditions are met for the occasion. We further discuss about the SNNs that are converted to run on conventional machines in Chapter 4 - Section 4.2.3 as a part of our thesis scope.

2.6 Summary

In this chapter, we offer an overview to ANN generations and models that are used in this research. We examine the topic of ANNs by making a timeline and following the crucial events in their history. Such crucial events are not only relevant to the success of ANNs, but also relevant to what went wrong in the sense of how and when they turned into power hungry.

Table 2.2: Feature summary of the ANNs utilised in this thesis that run on conventional hardware resources.

Generation	ANN Type	Section	Precision of Weights				Activation Type	
			Float32	Float16	Integer	Binary Only	Rate-based	Spike-based
2nd	CNNs	2.3	✓				✓	
	QNNs	2.4		✓	✓		✓	
	BNNs	2.4.1				✓	✓	
3rd	SNNs	2.5	✓				✓	✓

[13]SNN-ToolBox: https://github.com/NeuromorphicProcessorProject/snn_toolbox
[14]BindsNET: https://github.com/BindsNET/bindsnet
[15]Norse: https://github.com/norse/norse
[16]SpikingJelly: https://github.com/fangwei123456/spikingjelly
[17]https://github.com/Shikhargupta/Spiking-Neural-Network

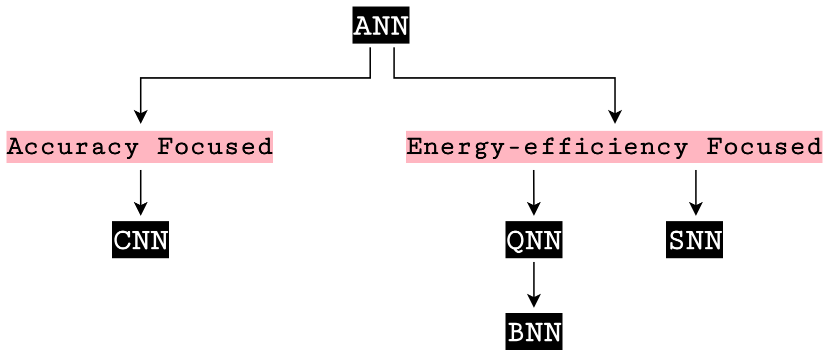


Figure 2.13: The relations between the ANNs that are mentioned throughout the thesis in terms of energy-efficiency.

We illustrate the relations between the aforementioned ANNs in this chapter, which will also be mentioned throughout the thesis, in Figure 2.13. We find it appropriate to classify them as *accuracy* and *energy-efficiency focused* within the thesis scope. The sections where the ANNs are introduced in this chapter are displayed in Table 2.2 along with their identifying features that contribute to the hypothesis of this thesis. For example, the column “Activation Type” in the Table showcases clearly what makes SNNs the 3rd generation, and how they are different than the rest of the neural networks. Moreover, the column “Precision of Weights” identifies the difference in the data types and precision of the neural units of the regarding networks. In Figure 2.14, we display the relations between all the aforementioned ANN types in this chapter over the hardware resources on which they were built to run in the first place. We discussed the term neuromorphic computing in Section 2.5 that is also where the similarities between the SNNs and the Perceptron were highlighted.

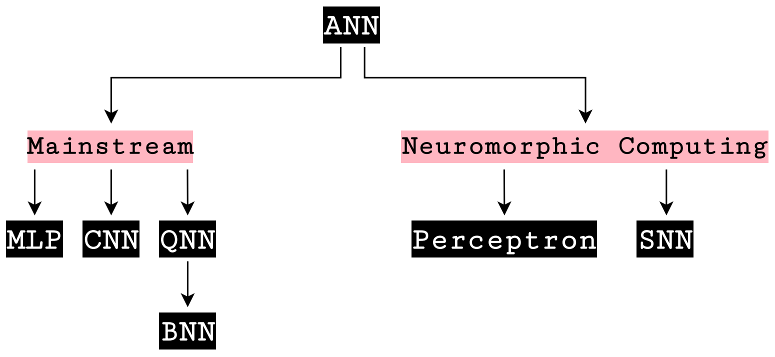


Figure 2.14: The relations between the ANNs that are mentioned in this chapter in terms of computing resources.

In the next chapter, Chapter 3, we proceed to covering biological foundations of the neural networks. As we summarise and discuss in this chapter, inspirations from nature have fed the researchers throughout the history of ANNs. The hypothesis in this thesis also roots from the nature. Such roots, called the neural reuse theory, will be elaborated in the next chapter along with the related work of this research.

Chapter 3

Background II: Neural Reuse

There is no machine like brain. It is soft, more energy-efficient than any human-made machine, yet capable of putting forward tasks with an endless repertoire. It is no surprise that scientists have always been curious about the mystery behind it.

A brain consists of a connectome, i.e., a series of connections between neurons. It is the connectome that defines an organism; its behavioural patterns, abilities, and disabilities. We may think of connectome as a map or a graph, as seen in Figure 3.1, to put it in an analogy; and the analogies are the causes of what we have reached today to put forward mechanisms of virtual brains that emulate the real brains.

"Nature vs. Nurture" concept have long been discussed in the literature to solve the mystery behind the connectome. Experiments and observations indicate that even twins might have entirely different connectome in their brains suggesting that the so far evidence might point to "nurture" rather than "nature". Aside from the possibilities, for a certainty, the connectome and all the different connections in the brain are there to fulfil a functionality.

Neurons are divided into groups based on their shape, e.g., pyramidal neurons; evolution, i.e., reptilian brain, limbic brain, and neocortex; functionality, i.e., sensory, motor, and interneurons; and brain region, e.g., the primary visual cortex (V1) or *Brodmann* Area-17 (Strotzer, 2009) in the brain. The same shaped neurons might be serving in different brain regions, as in the case of pyramidal neurons. Therefore, brain regions are divided based on their functionality, as in the case of *V1*. In this chapter, the neurons we mention will be based on their brain region.

A randomly generated graph to resemble a connectome is showcased in Figure 3.1. The darker a node, the more connection it has in the graph. Based on that, the mahogany coloured node at the centre of the graph has apparently the most connections;

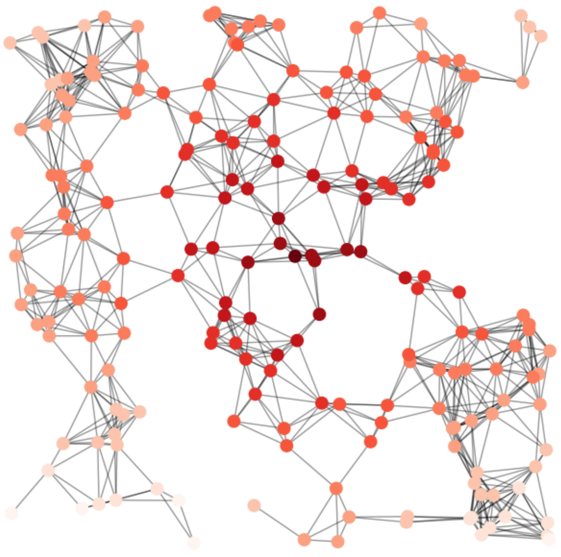


Figure 3.1: A connectome resemblance.

in a sense, it is a “hub”. Consider that the graph in Figure 3.1 includes neuron groups which serve to different purposes, ergo considered as belonging to separate brain regions. In that case, they would be connected to neurons which are not considered belonging to the same brain region. This means that there are interconnections between different brain regions. This also means that the neurons enabling the interconnection are used for more than one purpose, to fulfil different functionalities. In other words, firstly, they are used to serve their original function, and then re-used to serve another function. Moreover, this duality does not harm or change their original function. This is called **neural reuse** (Anderson, 2010, 2015).

In this section, we delve into the neural reuse theory and explain it by giving examples from the biological neural networks. Additionally, the ANN applications inspired by neural reuse are presented in this chapter as related work, and as the technical background information of this thesis.

3.1 Neural Reuse: The Theory

Reuse by name suggests a change in the original use. When the change occurs, the original use-case still remains, thus change here causes the *reuse*. When applied into the nervous systems, the reuse concept becomes “neural reuse ”. *Neural reuse*, the theory, suggests that the majority of the connections in a connectome are formed over already existent ones through their reuse in different functions, mainly due to preserve energy (Anderson, 2010). Same structures are used to realise sometimes even seemingly unrelated functions via different type of connections.

The neurons which are used for multiple purposes in a nervous system may be the output of two types of formations according to the *neural reuse* theory (Anderson, 2010): reuse in evolution and reuse during early development. These two possible formations are discussed as two following scenarios:

Evolutionary Neural Reuse Neural reuse may be the result of millions of years long evolution, and/or metamorphosis, in the first formation scenario. The result not only affects neural integrity, but also may affect physical appearance in this scenario. For instance, by the time a caterpillar turns into a butterfly after cocooning, both its bodily structure, hence its integral organs, and its nervous system go through change. The change belonging to the integral organs is obvious, thus it is commonly addressed as remodelling of the body. The knowledge that regarding the nervous system change is not recycling of the underlying system, but reuse, is the evidence of research findings pointing out remaining memory of the butterfly belonging to its caterpillar days (Blackiston et al., 2008).

Developmental Neural Reuse Second scenario involves a change that does not affect physical appearance from inside out, but essential for the system itself. A new inner functionality is necessary for the system in this scenario. Therefore, neural structures, either a whole brain region or a specific group of neurons, are introduced a brand new identity which produces the desired functionality in order to fulfil the need. However, when being introduced a new identity, their previous identity still remains. This phenomenon, using existent structures for a new functionality rather than generating new ones, occurs to preserve energy (Anderson, 2010). This, *multi-functionality with energy efficiency*, is the backbone idea of the thesis where our focus is to apply developmental phase neural reuse computationally.

This theory were firstly rooted in and came to life after observing thousands of functional Magnetic Resonance Imaging (fMRI) scans of the human brain. Such scans were examined for diversity analysis and functional connectivity analysis before the theory put forward. Although, the birth place of the theory is humans, many evidence regarding neural reuse in other living beings has been put forward ever since. Neural reuse does not solely concern the human brain or, in general, spiking neural systems, but all nervous systems from the invertebrates to mammals whomever is applicable.

We discuss neural reuse further by providing examples in the following sections in this chapter. In Section 3.2, neural reuse is exemplified by biological findings. In Section 3.3, ANNs that are inspired by and adapted from the theory are discussed which comprises the related work of this research.

3.2 Neural Reuse in Biological Neural Networks

Neural reuse is exemplified mostly with mammalian brain, specifically human brain, with the aid of neuroimaging techniques, such as fMRI, in the theorised paper (Anderson, 2010). It is the language usage and the ability to communicate of humans influenced the author the most, and how language usage is connected to multiple areas in the human brain such as motor-control and recognition. Furthermore, this was not the only study showcasing "handy" language connections as language tasks being the most scattered in the human brain (Anderson, 2015; Ardila et al., 2016; Ziegler et al., 2018; Pulvermüller, 2018; Rajalingham et al., 2020; Phillips and Pylkkänen, 2021). In a recent study which was conducted to shed a light to the brain while a programmer codes, participants are observed using the same brain areas with speech comprehension when they are coding which may sound unusual (Siegmund et al., 2020). Some studies point out that as being a recently evolved feature of the brain compared to the rest of them, "language" was facilitated by the features already available in the brain, and is the perfect candidate to exploit neural reuse (Zerilli, 2019).

Enculturation in numerals are considered as an example of neural reuse (Jones, 2020). Emotions felt via different senses such as to see and to read are the result of neural reuse based on human observations (Ziegler et al., 2018). Other emotions which may be tightly coupled with some illnesses, such as Obsessive-Compulsive Disorder (OCD), are linked to neural reuse by clinical studies (Viol et al., 2019).

There are also other studies investigating local and global interactions within and between human brain regions. Although the point at which we stand today is by far the furthest in the scientific history thanks to the technological advances, we are still far away to reach concrete results over these neuroscientific theories. However, the latest scientific findings also indicate that especially behaviours including motor functions, visual activities, and active attention are widespread connected in the brain (Garcia et al., 2020).

V1 was believed to be the start point of the all visual focused processing in the brain as being the hub of the visual centre (Grill-Spector and Malach, 2004). However, clinical studies targeting people with cortical blindness, which is caused by brain damage, showcased that despite the damage in V1, obstacle avoidance is still achieved up to a certain degree by these people (Striemer et al., 2009; Ross et al., 2018). The visually guided spatial co-ordination area, dorsal stream, was seen as active during the obstacle avoidance experiments in their brain images. This phenomenon later got pointed to the re-organisation, neuroplasticity, of the cross-modal perceptional brain paths by another study (Voss, 2019). However, it still remains unclear whether it was always the case of neural reuse between the two relevant areas of the brain, or solely plasticity coming at a later age.

3.2.1 Neural Reuse in Invertebrates

Although the examples of this phenomenon is relatively easier to measure and observe on the more advanced animals, e.g., humans, neural reuse is thought to be benefited more by the biologically less advanced. With the help of advancing imaging techniques, such as electron microscopy (Lizbinski and Jeanne, 2020), the majority of the specimens of neural reuse in animal kingdom are observed in the invertebrate, especially in insect nervous systems (Niven and Chittka, 2010).

The body of *C. Elegans* consist of only 302 neurons. A specific example of neural reuse for their case is how they move in their habitat. There are nine neurons responsible for the worm's movement, and the direction of the movement is decided by the nine neurons and what kind of input they receive before the action (Rakowski and Karbowski, 2017; Cohen and Denham, 2019).

Another example is the determining mechanism of movement in cockroaches. Cockroaches use all their legs during walking under normal circumstances. However, they are observed as not using all of their legs while running away from a threat even though the neurons controlling the movement of all their legs belong to the same small-world network (Patanè et al., 2018).

Insect *mushroom bodies* are best known as to be the learning centres in insect brain which serve mainly to *store and recall* (Sterling and Laughlin, 2015) and also decision making (Heisenberg, 2003). Therefore, they are considered as the functional equivalent of hippocampus and amygdala in mammalian brain which are believed to be responsible for the same tasks. However, along with the aforementioned tasks, mushroom bodies are also in charge of many other activities, such as sleep control, regulating hunger and motor learning (Patanè et al., 2018). Some of the mushroom body functions are still yet to be understood, but there is enough evidence to dispute years long belief of insects limited capability. Due to wide task repertoire of mushroom bodies while comprising only a few thousand neurons, they are seen as a neural reuse paradigm (Arena et al., 2013; Vogt et al., 2014).

3.2.2 A Special Case: Synaptic Plasticity

Synaptic plasticity deserves a highlight in this section as it, although in such close proximity, slightly differs from neural reuse. Both neural reuse and synaptic plasticity are considered as the result of neuroplasticity. However, in case of synaptic plasticity, the original function of the neuron, if there was any in the first place, is entirely lost. Synaptic plasticity applies an "erase and write" approach onto neurons when the organism is learning, thus requiring to obtain new functionalities. It has still utmost importance for the living beings, hence we also give examples of such type of plasticity to draw the line between them.

It has been believed that a perfectly structured brain, as we know of, is necessary to accomplish complex functions in life for so long. One exception has appeared in the recent years: a rat who is capable of functioning as her counterparts, yet with an unusual brain structure in terms of size and shape. The rat has *hydrocephaly* disease which results *cerebrospinal fluids* to dominate the entire brain, hence leaving her with a shrunken, miniature version of her brain which is located next to her brain stem. Having observed her miniature, yet structurally whole brain, the researchers run many tests to compare her behavioural scheme to other rats. Nevertheless, none[1] stands out pointing to diminish her capabilities. This research conducted by Ferris et al. exhibits the importance and power of neuroplasticity in regard to survival in nature.

The invertebrates have long been thought to be born with innate, rigid connections in their nervous systems. However, supported by the advancing technologies, studies

[1]The researches in the study note to observing some behaviours in the rat, such as staying for too long in a corner, which may indicate high anxiety levels. They do not conclude whether it is related to the miniature brain of hers or not.

are appearing more and more, especially in the recent years, proving the plasticity in their neural networks as well. Such features are more observant in insects. Because of the harsh living conditions in their habitat, their ability to learn where their nest is located in the desert is discovered in desert ants (Rössler, 2019). Bees are especially known for their intelligence as being quick learners showcased on reinforcement learning experiments (Tedjakumala and Giurfa, 2013; Peng and Chittka, 2017). Recent studies showcase bees learning to pull a string, which is considered to require high intelligence in animal kingdom, and even teaching other bees to do the same as being social creatures, in order to reach a reward, sugar water in the experiment (Alem et al., 2016). All these are thanks to the synaptic plasticity in their mushroom bodies, the learning centres, as introduced in the previous section, Section 3.2.1.

3.3 Neural Reuse in Artificial Neural Networks

The more ANNs become commonly used, the more research is done to increase their efficiency, allowing them to be used on a diverse range of hardware resources. Therefore, as an efficiency aid to reduce memory requirements as in biological neural networks, different types of reuse in ANNs emerged in literature in recent years. The neural reuse includes an active search for neural collaborations which may show themselves as partnerships and/or alternatives (Anderson, 2015). Proving that, we come across various different types of computational neural reuse studies in the literature and reach a decision to review them under two main categories: neural reuse in elastic systems and neural reuse in rigid systems.

While *elastic systems* in Section 3.3.1 follow the "evolutionary" route, the *rigid systems* in Section 3.3.2 follow the during "development" route (see Section 3.1 for route details) in neural reuse. We discuss the SotA research results in both parts in this section.

3.3.1 Neural Reuse in Elastic Systems

The related work in this section is built upon Anderson's neural reuse theory (Anderson, 2010). Being directly inspired by the biological neural networks, elastic systems make use of neuroplasticity. To provide such neuroplasticity, they utilise dynamical systems. The dynamical systems that are utilised by the work in this section are SNNs that involve STDP, *reservoir computing*, and feed-forward ANNs on which *evolutionary algorithms* are applied.

Arena et al. conduct research to emulate the mechanisms in insect mushroom bodies via SNNs. The SNN in the aforementioned research comprises two layers and overall nine neurons that connected to each other via STDP which is the learning method in SNNs. The resulting SNN is multi-functional and among the behavioural repertoire of the neural network are conditioning, attention modelling, and memory consolidation.

The reservoir computing, or *reservoir computer*, is an umbrella term and a framework for dynamical systems that allow flexibility in neural model design by providing a "reservoir" that contains non-spiking neurons (Tanaka et al., 2019)[2]. The framework relies on recurrent connections and is ideal to emulate the neural network of small species, such as *C. Elegans*, or the retina of larger living beings where the neurons are non-spiking (Grosu, 2020). Unlike the feed-forward ANNs, and like the SNNs, reservoir computers process sequential data (Tanaka et al., 2019). Continuous Time Recurrent Neural Networks (CTRNNs) which are especially used in the robotics today (Hasani, 2020) to realise realistic and energy-efficient neural modelling are among the sub-types of reservoir computing. The main principle of reservoir computers is to enable self-sustainability in a learning system even, and especially, in chaotic systems as in nature (Flynn et al., 2021).

Unlike their previous work (Setzler and Izquierdo, 2017; Candadai and Izquierdo, 2018), in which CTRNNs were used to realise multiple functionalities via neural reuse, Benson et al. use a feed-forward ANN to put forward three different functionalities by using the same network. They apply evolutionary algorithms to evolve the network to perform multiple tasks. Such tasks are inverted pendulum, cart-pole balancing, and single-legged walking. The network consists of two hidden layers and overall 14 neurons.

* * *

Notice that in the works of Arena et al. and Benson et al., the regarding multi-functional neural networks are small considering their overall layers and neuron counts. They are indeed smaller than even the first ever created ANN, Mark I as introduced in Chapter 2 - Section 2.2. Keeping that in mind, it is important to mention here that the objective of the above aforementioned research is two-fold. Firstly, it is to understand

[2]We do not mention this topic in Chapter 2 where we explain the ANNs background because the thesis specifically concerns the feed-forward ANNs.

how neural reuse works in nature by emulating the biological neural networks into appropriate computational platforms. Secondly, it is to preserve the energy within the targeted system, such as robots, in which the battery power is crucial. Finally, as mentioned in the introduction of this section, Section 3.3, they follow the "evolutionary" route in neural reuse by making their model multi-functional from the very beginning.

3.3.2 Neural Reuse in Rigid Systems

The conventional ANNs lose their elasticity once their training process is complete. They would need to be re-trained to regain elasticity which is energy consuming. Therefore, we call them as "rigid" within this section. The systems that we mention in this section are all feed-forward CNNs. We explain the reuse solutions for rigid single networks in Section 3.3.2.1 and for multiple networks in Section 3.3.2.2. The research in this section aims to put forward more energy efficient CNN solutions to facilitate their use on computationally limited resources, such as embedded platforms and mobile devices.

3.3.2.1 Neural reuse on a Single Convolutional Neural Network

Kopuklu et al. offer to use the same layer blocks, where possible in the architecture of a model, instead of generating new ones. This idea roots in the fact that CNNs are built by the same repetitive layers architecturally. This approach, named as *LruNet* in the paper, is applied prior to training the model. Obtaining reduction in memory access cost is aimed via LruNet. Solely accuracy results are presented in the evaluation part of LruNet, thus we are unable to report the reduced memory access cost values.

Ning and Shen apply vector-reuse between the layers in CNNs during the inference stage. Such reuse, named as *deep reuse* in the paper, is achieved via finding the similar neuron-vectors searching from neuron to neuron, and assigning them where applicable. Five stages, among which are clustering and similarity distance measuring, are gone through in the deep reuse algorithm to find vector similarities. Up to $2x$ speed-up is achieved during the inference stage via deep reuse.

3.3.2.2 Neural reuse on Multiple Convolutional Neural Networks

This section focuses on neural reuse in multiple neural networks by "merging pre-trained CNNs". The merge especially fits into the neural reuse during "development" in biological neural networks as each network already has a functionality that is going to remain after the merge as well. The below research utilises *weight sharing* among the layers for an efficient merge.

NeuralMerger by Chou et al. is the first research in this field and a pioneer to offer to merge different inputted CNNs which is still technically the only one to do so up to this date. We explain the details of this research in Chapter 4 - Section 4.1.2 as it constitutes the base of our work.

Wu et al. merge two MobileNet's (Howard et al., 2017) that were trained on separate, solely image, datasets. Their merged model reaches $\approx 1.3x$ compression and $\approx 1.5x$ speedup during inference.

Currently the SotA in this category is *Multi-Task Zipping (MTZ)* (He et al., 2021)[3]. MTZ is built upon the work of He et al. (2018) which was able to merge two networks at a time, and is able to merge up to nine CNNs via layer-wise weight sharing. It has layer coverings for convolutional, fully connected and residual layers; also, for now, empirically supports different inputted models, for example audio and image, as reported. MTZ results with a 0.46% error rate increase on average of total nine models and a $5x$ in total model size decrease.

3.4 Discussion: The Link Between The Terminologies in this Chapter

We mentioned the two possible formations in theory regarding neural reuse earlier in Section 3.1 which are "evolutionary" and "developmental" neural reuse. Furthermore, we reviewed the computational applications related to the neural reuse theory under two categories as "elastic" and "rigid" systems in Section 3.3. The former terminology is used in the form put forward by Anderson, yet the latter terminology is our preference based on the state, i.e., being elastic or rigid, of a system that the neural reuse becomes active.

We mentioned that our research ideologically follows the developmental route of neural reuse in Section 3.1. Moreover, we reviewed our related work in Section 3.3.2.2 under "rigid" systems.

The ANNs that are reviewed as *rigid* systems in this research are conventional, feed-forward CNNs that were trained once, then became stiff, hence the name "rigid". Such systems were later worked on by applying algorithmic manoeuvres either

[3]The date of publication for the work MTZ by He et al. is 02/11/2021. The regarding SotA results are lastly checked on 18/12/2021 via *Google Scholar*.

to make them work more efficiently (Section 3.3.2.1), and/or to introduce them more functionalities (Section 3.3.2.2). Because such systems align with our research goal, we review our related work in Section 3.3.2.2 under “rigid” systems. On the other hand, we review mixed types of approaches including ANNs, e.g., SNNs; and reservoir computing paradigm, e.g., CTRNNs, that emulating neural reuse computationally in Section 3.3.1 as *elastic* systems. These approaches, by being directly inspired by the neural reuse theorem, utilise neuroplasticity and either apply evolutionary algorithms or exploit synaptic plasticity to introduce neural reuse to their systems, hence the name “elastic”. They follow a top-to-bottom approach in the way they apply neuroplasticity where the neural reuse is set to grow in the very beginning, hence present all the time within the system. This approach aligns with the evolutionary route mentioned in Section 3.1.

Considering our review material and the terminology in this chapter, the elastic systems fit the evolutionary route, and the rigid systems fit the developmental route. However, this description is only relevant for the discussed approaches within the thesis.

3.5 Summary

In this chapter, we have presented the theoretical background of neural reuse theory and provided examples of the theory both from biological and artificial neural networks. Regarding the examples of neural reuse on computational systems, Section 3.3.2.2 comprises the related work of this thesis.

As we have discussed in Section 3.2.1, neural reuse in insects increase their general ability by enabling multiple tasks with limited neurons and neural paths. From a computational perspective, this can translate into computational devices that have limited memory and computational capacity such as edge devices, which with neural reuse, can operate multiple tasks without requiring more memory space. In addition to this, the limited computational capacity in edge devices require higher energy efficiency since these devices generally operate on battery power. In the next chapter, Chapter 4, we present our approach to provide multi-functionality to convolutional neural networks and offer solutions to decrease the memory and computational costs.

Chapter 4

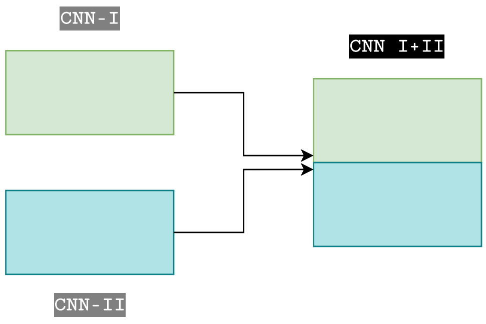
Achieving Energy Efficiency in Multi-functional Neural Networks

The Truth: It's more about the algorithm than the hardware.

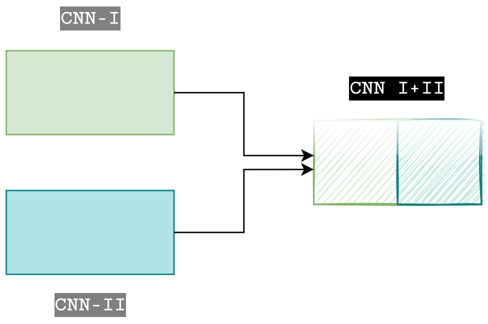
(HOROWITZ, 2014)

We defined the neural reuse theory, provided examples from the biological neural networks, and discussed our literature survey including computational forms of neural reuse in the previous chapter, Chapter 3. In this chapter we explain how "developmental" route neural reuse, explained in the previous chapter, can be achieved in the form of multi-functional neural networks. Neural reuse in simple terms means parts of a, or a whole, neural network being reused for a different purpose, such as for or within another network. We can establish neural reuse as multi-functional neural network where instead of two different networks doing inference on two different jobs, the networks can have some common parts that they share and the shared parts are reused essentially for different networks. Such reuse provides compactness as well as efficiency which we aim to show in this work.

We illustrate two scenarios regarding neural reuse in Figure 4.1. On one hand, the first scenario, Figure 4.1a, showcases a straightforward approach in combining two neural networks: They are stored and utilised together without any intervention. Therefore, we see that their actual sizes are combined without any compression which is as expected. On the other hand, Figure 4.1b showcases an ideal neural reuse scenario in case of two neural networks: They are merged, combined in a way to utilise their shared, i.e., similar, parts in one body. Therefore, their size together is less than the total combined[1] and such combination remain compact without hurting the memory. In other words, let the model size in physical memory be a for CNN-I, and b for CNN-II in Figure 4.1: while the total output size, the size of CNN I+II, would be $a+b$ in Figure 4.1a; it would be $< a+b$ in Figure 4.1b.



(a) Without Reuse



(b) With Reuse

Figure 4.1: The illustration of two CNNs merged with, and without neural reuse. **(4.1a)** Illustrates two CNNs merged without neural reuse. **(4.1b)** Illustrates two CNNs merged with neural reuse.

We have described how we interpreted the developmental route and devised to emulate neural reuse in our research in the preceding paragraphs. Our experiments, that we discuss in-depth in Chapter 5 and 6, prove that solely combining neural networks for multi-functionality does not yield sufficiently satisfactory outcomes as we would address as neural reuse emulation. Therefore, we propose a two-step approach **(1)** to provide multi-functionality and **(2)** to decrease the energy costs of CNNs as displayed in Figure 4.2. In the first step, we use a research tool named *NeuralMerger* which can merge two pretrained networks even if they are essentially different typed classification models, i.e., sound and image data. Merging two different models and reusing both partially is how neural reuse works in nature as discussed in Chapter 3. NeuralMerger is not only able to merge two different networks, but also decreases the model size thanks to the reuse. In the second step, we convert the multi-functional network into different neural network types using binarisation, quantisation, or conversion into SNNs in order to provide further efficiency. Binarisation and quantisation are used to decrease the model sizes and computational complexity while SNNs are inherently energy efficient.

[1]The illustration depicts an ideal scenario where the two CNNs are combined in ½ ratio. That would not be the case for every application since the outcome would be proportional to the similarities between the networks. However, the point of neural reuse is that the final size of the combined networks to remain less than the overall combination of both sizes.

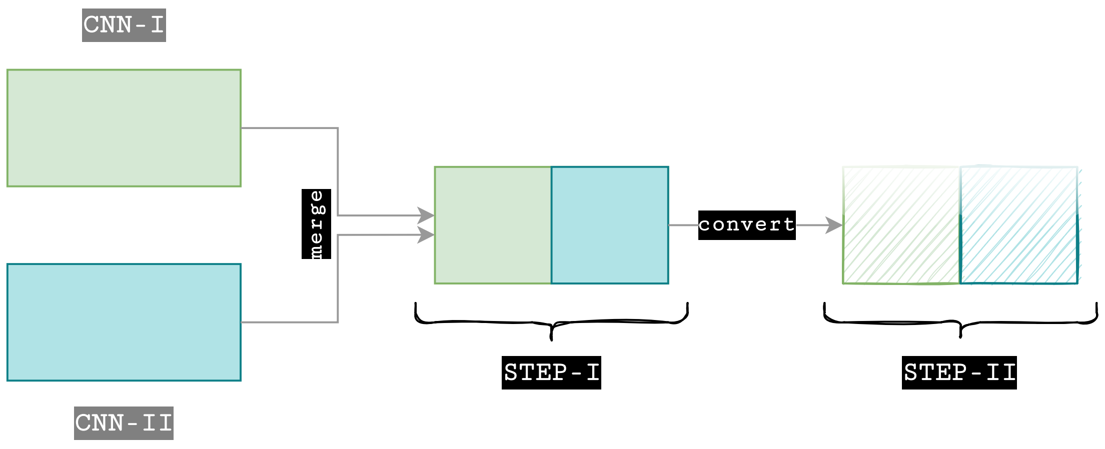


Figure 4.2: The two algorithmic steps of our approach.

This chapter consists of three sections: The first two sections involve the first, Section 4.1, and second, Section 4.2, steps of our approach consecutively, and the final section, Section 4.3, summarises the chapter.

In Section 4.1, we outline the literature survey regarding multi-functional ANNs. The survey also involves introducing the first step of our approach that is realised via *NeuralMerger* by Chou et al. in Section 4.1.2.1. In Section 4.2, we elaborate the second part of our approach that consists of various conversion experiments which is the technical part of our novel proposal.

4.1 Achieving Multi-functionality in Neural Networks

The recent developments in neural networks enabled a wide range of AI applications including computer vision, speech recognition, Natural Language Processing (NLP) etc. To handle various tasks, the usual way is to design different network models and train these models with corresponding datasets. Well-trained networks perform quite well for a specific purpose, but in more practical AI applications, it is common to use multiple tasks simultaneously. Having multiple models for different but specific tasks result in a race for the computational resources on the system between models. Therefore, integrating multiple network models to use the system resources efficiently becomes an important problem for future AI applications.

Sharing resources on a system is indeed a difficult optimisation problem, but latest research shows that there are ways to combine multiple neural networks into a single model in order to eliminate resource sharing difficulties in the system level altogether.

The main approaches to achieve multi-functionality in neural networks can be divided into two groups; training a new network from scratch that is able to do multiple tasks which we explain in Section 4.1.1, and merging multiple pretrained neural networks into a single network which we explain in Section 4.1.2.

4.1.1 Training Neural Networks for Multi-functionality

There are different approaches in order to achieve multiple tasks simultaneously via a single neural network. A typical way is to increase the output nodes of a network and train it from scratch. Kaiser et al. propose an approach that allows image, sound, and text inputs with different dimensions and then convert them into a unified representation. Their multi-model architecture with building blocks enable to create a deep learning model that is able to learn a number of large-scale tasks from multiple domains.

The goal of Aytar et al. is to create representations that are aligned across modality. For example, considering the sentence ”she jumped into the pool.”, the concept of this sentence can also be represented as the image of a pool or the sound of splashing. The authors use such representations to create a deep convolutional neural network that accepts sound, sentence and image inputs and the model is then trained from scratch. Their experiments show that the new model achieves higher retrieval and classification performance for challenging in-the-wild situations.

Although it is useful to create multi-functional neural networks using a learn-them-all approach, learning multiple tasks from scratch requires a cumbersome training effort and intensive inference computation. Additionally, in a learn-them-all approach, it is difficult to choose a network architecture that is suitable for learning all the tasks well in advance. Since many well-trained models are available now, merging pretrained models to provide multi-functionality is another valid option.

4.1.2 Merging Pretrained Neural Networks

There are a few studies focusing on merging well-trained network models into a single model. We have briefly discussed how the neural reuse approaches have been implemented by merging pretrained networks in the literature in Chapter 3 - Section 3.3.2.2. A systematic approach is proposed by Chou et al. for merging and compressing multiple CNN models where the input types of the models are allowed to be different. This work and the tool that is the outcome of this research, *NeuralMerger*, has been used in this thesis in order to merge multiple pretrained network models into a single model, thus providing multi-functionality to a network while compressing the multi-model approach. We base our first step onto NeuralMerger because (1) it holds the best metric results in terms of accuracy and model size compression, (2) it enables working with different data types at the same time, (3) the inter-network quantisation it achieves via *k*-means clustering which is the key to how NeuralMerger merges pretrained models. The third item basically offers an ML solution to an ML problem. An illustration of the main mechanism of NeuralMerger is displayed in Figure 4.3.

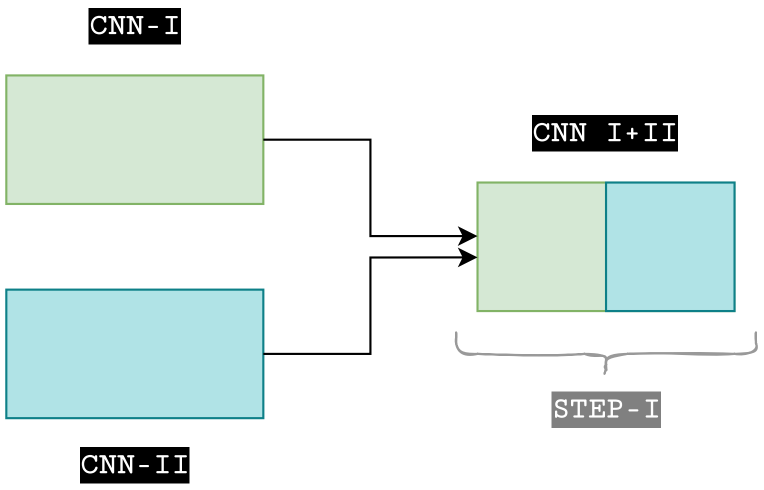


Figure 4.3: An illustration of the main scheme of *NeuralMerger*.

In this section, we explain the internals of NeuralMerger and provide detailed explanations on how two CNNs are merged. NeuralMerger follows a three step procedure in order to merge two networks. Assume two models, *A* and *B* are being merged, where c_A and c_B represent *convolutional* layers, followed by *fully connected* layers represented by f_A and f_B. In the first step, NeuralMerger merges the convolutional layers, in the second step, the fully connected layers are merged, and in the final step newly merged network is fine-tuned via BP. These steps are explained in detail below.

4.1.2.1 NeuralMerger: Merging Convolutional Layers

NeuralMerger is able to merge layers of the same type be it convolutional or fully connected layers from two different networks. The principle is that, given two layers, one in model *A* and one in model *B*, finding a set of *codewords* that represent the weights of the layers from both networks with small quantisation errors.

Assume that two convolutional layers, one from model A and one from model B are to be merged. The layer of model A has the input size $\mathrm{N_A} \times \mathrm{M_A} \times \mathrm{d_A}$, where $\mathrm{N_A} \times \mathrm{M_A}$ corresponds to the spatial size, and $\mathrm{d_A}$ is the number of channels. The input is convolved with $\mathrm{p_A}$ convolution kernels where the size of each kernel is $\mathrm{n_A} \times \mathrm{m_A} \times \mathrm{d_A}$. The same notations also apply to model B.

The aim of the NeuralMerger is to jointly encode the convolutional coefficients in order to achieve a decrease in the number of joint coefficients compared to the total number of separate coefficients, i.e., convolutional kernels of model A is denoted as p_{A} and model B as p_{B}, the number of codewords the NeuralMerger aims to achieve is shown as $p < p_{\mathrm{A}} + p_{\mathrm{B}}$. However, jointly encoding the convolutional coefficients is difficult if the kernel dimensions are different between the models. To address this issue, NeuralMerger unifies the different sized convolutional kernels by first decomposing each convolutional kernel into 1×1 kernels.

Given an ordinary convolution operation, the convolution is generally applied as a 2D sliding window operation followed by a channel-wise summation operation along the depth direction. NeuralMerger decomposes a convolution operation sized $n \times m \times d$, into convolution of multiple $1 \times 1 \times d$ operations and combine them with shift operations. Spatial decomposition of the convolutional kernels results in for each model as $C = nm$ convolutions of size $1 \times 1 \times d$. The kernels for each model can then be unified into 1×1 in the spatial domain regardless of the differences between the convolutional kernels sizes of model A and model B.

After decomposing the convolutional kernels in the spatial directions, NeuralMerger separates the kernels along the depth direction due to the possible mismatch in depth channel differences between the models. The separation is done by simply separating $1 \times 1 \times d$ kernels into non-overlapping $1 \times 1 \times r$ kernels along the depth direction. In case the depth d is not divisible by the value r, the last segment of the separated kernel is padded with zeros.

After the convolutional kernels are decomposed into smaller units, NeuralMerger runs a k-means clustering algorithm with various initials and select the results yielding the least representation error to produce the codewords. From these codewords, codebooks are created for each segment. The convolutions are pre-computed on the codebook, and a lookup table is built to index the results. All these steps are shown in Figure 4.4.

4.1.2.2 NeuralMerger: Merging Fully-Connected Layers

Unlike the convolutional layers which have sliding window operations, the operations in fully-connected layers are all summation based. Therefore, given the two weight matrices of models A and B, the weight vectors in each layer are simply divided into length-r segments along the row direction. The dim-r weight vectors in the same segment are then clustered using a k-means algorithm and C codewords are found. The codebook for the fully-connected layers is thus built this way, similar to the convolutional layers.

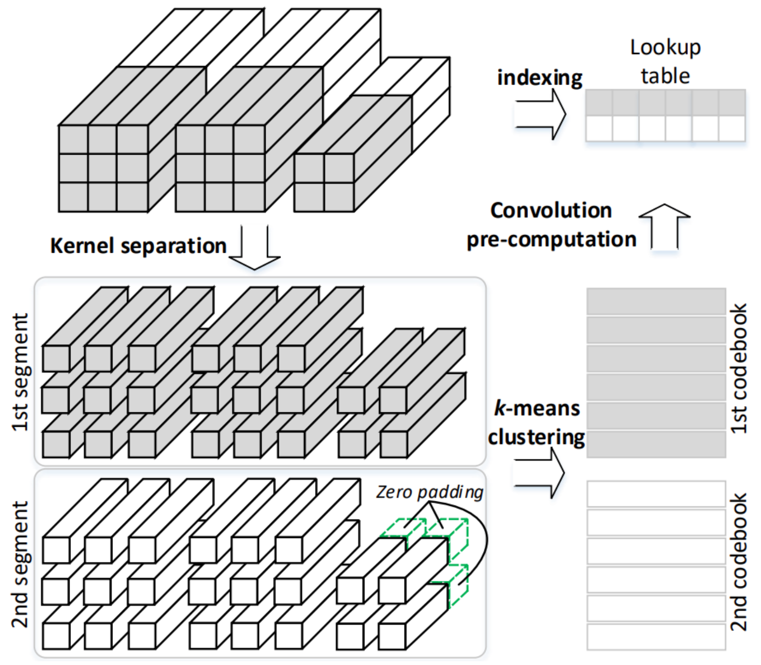


Figure 4.4: Illustration[2] of merging *convolutional* layers of two models going through three main steps: (1) Align - different sized layers and filters (2) Encode - coefficients $\rightarrow$ k-means clustering (3) Unify - codebook $\rightarrow$ lookup table.

4.1.2.3 NeuralMerger: Fine-tuning Phase

After the construction of the codebooks for the convolutional and fully connected layers, the model becomes ready to be fine-tuned from the training data through end-to-end BP. In NeuralMerger, all layers in the codebooks are tunable which is an advantage compared to single-model compression approaches (Wu et al., 2016).

[2]The Illustration is extracted from the work of (Chou et al., 2018, p. 2052).

For the fine-tuning phase, NeuralMerger uses the combination of two error terms. The first one is the classification, or regression, loss which is utilised in the original models A and B. The second one is the output mismatch error which is calculated for each layer. When applying the input, x_1, to the model A or B, the output of each layer in the merged model should be close to the output of the associated layer, L_1; and the norm is used to measure this error.

4.1.2.4 Our Utilisation of NeuralMerger

We use NeuralMerger by Chou et al. as a tool during the first step of our approach illustrated in Figure 4.2. The source code[3] of the NeuralMerger is publicly available. However, we have needed to make changes in the code in terms of software engineering to make it work with the current technology and fit into our code.

The code was written to work with a specific programming language, *Python 2.7*, and a main software library, *TensorFlow (TF) 1.4.0*, versions that reached their end of lives and are not supported anymore[4,5]. Therefore, we update all the required software packages and modify the code[6] to fit the newest possible versions and release it as open-source.

NeuralMerger was written in native TF which is currently the low-level Application Programming Interface (API) of the software library. Because such merge mechanism requires intricate code manoeuvres, we leave the main API level as it was. Nonetheless, we introduce saving the weights and/or the models into NeuralMerger. Such save mechanism is possible now either with low-level or the high-level API of TF. For the low-level API, we enable saving via the *checkpoints* of the TF; for the high-level API, *Keras*, the save is available after the model is trained via NeuralMerger. With this way, the models can be combined and used within Keras as well. This is how we obtain the singular neural network experiment results that are discussed in Chapter 6.

[3]NeuralMerger: `https://github.com/ivclab/NeuralMerger/tree/master/Fine-tuning`
[4]Python versions: `https://devguide.python.org/versions/`
[5]TensorFlow versions: `https://www.tensorflow.org/versions`
[6]Our modified NeuralMerger repository: `https://github.com/mervess/neural-reuse/tree/main/fine-tuning`

4.2 Converting Merged Networks to Provide Energy Efficiency

NeuralMerger utilises a joint encoding scheme for weight sharing which can be understood as a inter-network quantisation. Using a shifting and summation scheme, due to the decomposed kernels, in addition to the weight sharing decreases both the size of the model and the computational complexity, since shifting a summation on 1×1 kernels are faster than multiply accumulate computations. However, the merged model still uses FP floating-point numbers for the computations which are more expensive compared to integer values. These computations are especially difficult on edge devices where the computational capability is limited.

In order to decrease the computational cost of inference in merged models, we propose applying conversion mechanisms to the merged network to enable further efficiency both computationally and memory-wise.

4.2.1 Overhead on the Traditional Hardware

ANNs are mostly trained on high throughput devices such as large servers and GPUs due to the requirement of a large number of FP computations. As a result, it is challenging to run neural networks on low-power devices and research efforts are invested from both academia and industry to port neural networks into low-power devices i.e., phones, watches etc. While some research focus on running neural networks faster on these devices (Vanhoucke et al., 2011; Gong et al., 2014; Han et al., 2015; Romero et al., 2015) other efforts focus on either reducing the model sizes or reducing the computational complexity while trying to retain the accuracy of the models (Han et al., 2015; Yu et al., 2019). Both aspects of research aim to increase the energy efficiency of neural networks, but on some edge devices where the memory and computational power are limited, decreasing model size and computational complexity takes precedence to even execute a neural network model in the first place.

Table 4.1 shows the cost of computations and memory operations in energy. While Han et al. did not specify how these energy calculations were made, they have referenced to [*ref: Energy table for 45nm process, Stanford VLSI wiki.*] which is not an open source wiki. Therefore, we do not exactly know how these numbers were measured, but given the fact that these numbers are reported for 45nm technology, we assume each operation is schematically implemented via 45nm technology on Very

Table 4.1: Energy costs of floating point computations vs. integer computations (Han et al., 2015).

Operation (32 bit)	Energy (pJ)	Relative Cost
int ADD	0.1	1
float ADD	0.9	9
Register file access	1	10
int MULT	3.1	31
float MULT	3.7	37
SRAM cache access	5	50
DRAM memory access	**640**	**6400**

Large-Scale Integration (VLSI) design tools from *Cadence* or *Synopsys* in order to achieve the energy results per operation. Overall, the raw energy numbers are irrelevant since 45nm technology is old. However, the important part of the Table 4.1 is that it displays the relative cost of each operation which is independent of the technology used.

Computationally, doing any FP operation costs more energy compared to the same length integer operation as shown in the Table 4.1. Moreover, the most expensive operations are memory operations, especially accessing off-chip memory. Therefore, in order to achieve more energy efficient execution of neural network models, decreasing the computational complexity of operations by replacing more expensive FP operations with integer ones, also decreasing the memory usage increases the energy efficiency of the system.

The two main methods for decreasing both the computational complexity and the model size of neural networks that is related to the work in this thesis are quantisation and binarisation of the CNNs which we discuss next. Another method to achieve energy efficiency is to use an SNN model which is inherently energy efficient. Since conversion into SNNs is an orthogonal optimisation to merging two networks, we also discuss this option later in this chapter. We illustrate the aforementioned methods that constitute our second step in Figure 4.5.

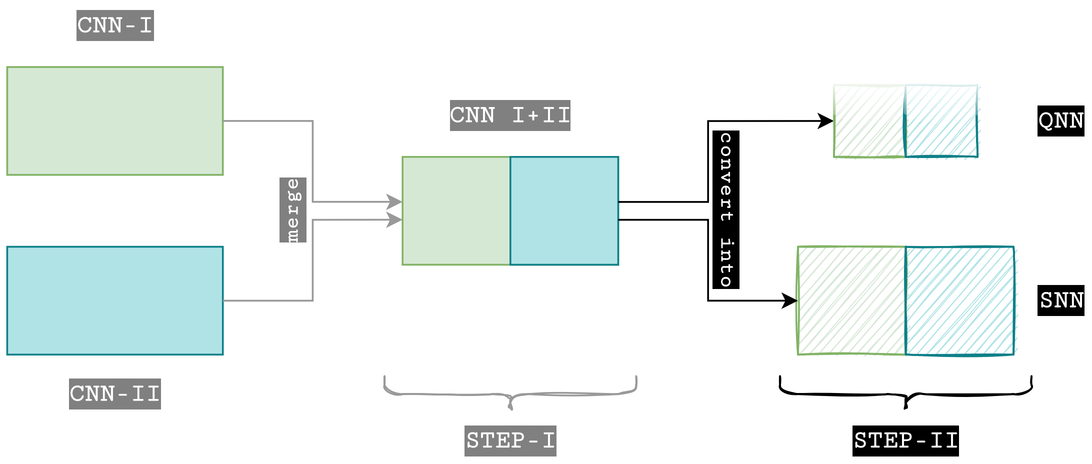


Figure 4.5: Step-II of our approach.

4.2.2 Quantisation of a Merged Neural Network

Although our initial intention was to convert the merged model into a BNN, our experiments have showed that converting a pretrained ANN directly into a BNN results in a monumental accuracy drop and an increase in model loss. Therefore, when it comes to BNNs, training from scratch is proved to be necessary. However, unlike the binarisation operation, a higher degree of quantisation in neural networks is possible without training the ANN. We choose the furthest possible degree in quantisation that does not require a training among the degrees of quantisation.

In the first step of our approach we used NeuralMerger to merge multiple networks which also does a cross-network quantisation on weights that helps reduce the storage requirements. Since our method already has this feature, we do not focus on approaches that quantise only the weights (Gong et al., 2014; Chen et al., 2015; Han et al., 2016; Zhou et al., 2017) which are less related with computational efficiency, instead we use the methodology proposed by Jacob et al. in order to reduce the computational complexity.

Jacob et al. propose a quantisation scheme that permits efficient implementation of all arithmetic computations using only integer arithmetic operations on quantised values. Since the performance and energy efficiency of integer only calculations is far superior than of floating point operations, this type of optimisation fits very well into what we aim to achieve in this thesis. We prefer to use *TF-Lite*[7] for our quantisation part of this thesis due to the following reasons: (1) it is a well-known and

[7]https://www.tensorflow.org/lite

mature infrastructure for machine learning, targeting low-power devices, (2) the work of Jacob et al. is implemented into TF-Lite, which is what we refer to for our quantisation methodology, (3) easy to use and open source characteristics of TF-Lite makes it widely used in research enabling further technological developments.

The quantisation scheme is an affine mapping of integers q, to real numbers r of the form

$$r = S(q - Z) \tag{4.1}$$

for the constants S and Z as shown in Equation 4.1. For 8-bit quantisation, q is quantised as an 8-bit integer, the constant S (for scale) is an arbitrary positive real number and the constant Z (for zero point) is of the same type as quantised value; and represents the quantised zero-point value which corresponds to the real value zero. Having a static quantised zero-point value allows the representation of real value zero which enables more efficient implementation of neural network operators since these often require zero-padding of arrays around the boundaries.

Jacob et al. also show that this quantisation can be applied to matrix multiplication, resulting in an integer-arithmetic-only matrix multiplication where the only non-integer values are S values which can be calculated offline. The steps that the quantisation scheme follows is listed below in which the second step involves applying Equation 4.1 to the weights of the pretrained network.

1. Create a training graph of the model using floating-point values.
2. Using proposed TF *fake quantisation* operations, downcast the tensors into fewer bit representations.
3. Simulation of the quantised model.
4. Optimisation of the inference graph for running in a low bit inference engine.
5. Run inference using the quantised inference graph.

The approach is already contributed to the TF-Lite, thus it is easy to follow these steps and use this quantisation method in any model.

As discussed in Section 4.1.2, our approach first merges multiple networks into a single network, capable of multi-functionality. In the next step of our approach, we use this merged network and quantise it using the approach explained above. One of the main differences is that the merged network uses a look-up table for shared weights. In order to also quantise the look-up table, we have used the TF's checkpointing capability to get the shared weights and also quantise these values that are going to be used during inference stage. We discuss the results of these experiments in Chapter 6.

4.2.3 Converting a Conventional Neural Network into a Spiking Neural Network

As the "popularity" of SNNs rises, so do the energy-efficiency measurements of different hardware resources and discussions accordingly. A recent study showcased that running an SNN simulation on digital neuromorphic hardware consumes $\approx$2 times less energy than running an equivalent ANN on the same system (Davidson and Furber, 2021). Another study comparing analog vs. digital neuromorphic hardware resources concluded that the analog consumes 20 times less energy than their digital counterparts along with requiring 5 times less memory space (Joubert et al., 2012).

Although SNNs are known to work most efficiently on neuromorphic hardware (Pfeiffer and Pfeil, 2018), they are also harder to work on and train on such resources. As alternatives, there are software libraries to simulate and run SNNs on conventional hardware, yet they require from scratch training (Hazan et al., 2018). However, the highest accuracy results that can compete with the CNNs have been obtained via the conversion of pretrained networks so far. Because we aim to exploit pretrained ANNs which are built to run on conventional machines, and additionally, considering the aforementioned hardships on neuromorphic hardware, our choice is to benefit from ANN-SNN conversion algorithms. Note that, unlike the case of quantisation, after converting FP-CNNs into SNNs, the weights of the neural network remain the same.

The two most common *neural coding* methods to convert a conventional ANN into an SNN are by using either a *firing rate based* or a *spike timing based* method (Brette, 2015; Guo et al., 2021). Such methods are also called as *rate-based / rate coding* and *spike-based / temporal coding* methods in the literature. In the former, the information is conveyed via spike rates; and in the latter, it is conveyed via the timing of spikes in the neural network.

Rate coding focuses on the cumulative rates coming from the spikes. It offers a collective approach by diverting decisions, needed for the result of a given task, towards the highest spike rates received from the neurons. Some rate coding approaches make use of *thresholding* as an interrupt mechanism for collecting the rates (Wu et al., 2022).

Moreover, weight normalisation is applied to the model to reduce the loss after the rate coding based conversions (Li and Furber, 2021; Wu et al., 2022). Rate coding has been widely applied due to its simply intelligible mechanism in the literature. Nonetheless, because of its "collective" point of view, rate coding is also known for its applications having latency, and slow processing times (Guo et al., 2021).

Temporal coding focuses on the timing of spikes. SNN simulations on neuromorphic platforms include built-in temporal schemes, yet conventional platforms, without running a specialised software library such as *PyNN* (Davison et al., 2009) as mentioned in Chapter 2 - Section 2.5, do not have such schemes. When applying the conversion via temporal coding, while some of these methods pivot on the *first spike time*[8] (Guo et al., 2021), others generate a *timescale* on the conventional machine, and follow through based on time steps (Stöckl and Maass, 2021). Temporal coding conversion technique reduces the spikes of the SNN since it focuses on not the overall rates, but the timing of the spikes. Therefore, it also reduces the latency of the conversion (Guo et al., 2021).

Although rate-based conversions are more common in the literature, spike-based conversions are considered as better fitted to the neuromorphic computing, hence more suitable for the overall nature of SNNs (Davidson and Furber, 2021). Note that both techniques are stated to root in human brain and utilised by the different regions of the brain in literature (Brette, 2015; Guo et al., 2021). Depending on the structure of the neural model, a temporal coded model can also be translated into a rate coded model (Brette, 2015).

Our aim is to minimise the energy consumption by applying a conversion into SNNs which are innately more energy efficient than traditional ANNs. Therefore, we experiment with both rate and temporal coding based conversion methods, and observe the trade-off between the metrics after conversion in order to reach a decision. We choose representative SotA research of each coding technique for our experiments which will be detailed in Chapter 5 - Section 5.5.1. After this point, the two methods will be addressed as *rate coding* and *temporal coding* methods in the following chapters of the thesis.

[8]Temporal coding via *first spike time* is also addressed as *Time-to-First-Spike coding* in literature.

4.3 Summary

In this chapter we introduced our approach to provide both energy efficiency and multi-functionality to ANNs. Our approach is inspired by the developmental route of the neural reuse theory discussed in Chapter 3. We show the internal workings of the NeuralMerger tool that we use to provide multi-functionality to ANNs by merging two networks into a single network, followed by the conversion of the merged model into a quantised model and into SNN in order to provide energy efficiency. We illustrate our algorithmic approach summarising the aforementioned technical steps in Figure 4.6 below.

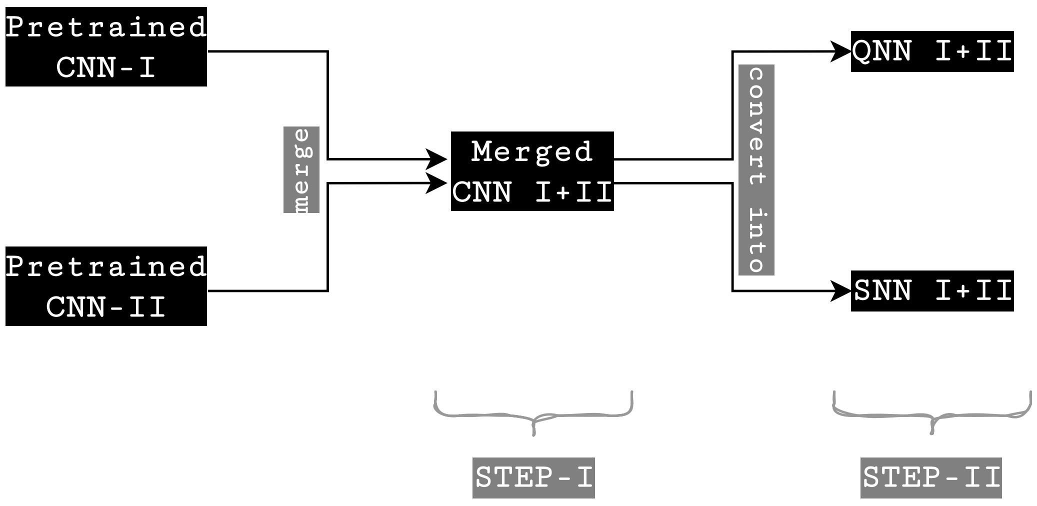


Figure 4.6: The proposed algorithmic approach.

NeuralMerger enables the multi-functional aspect as well as providing compaction of models when merged. The quantisation method is an orthogonal optimisation step to provide computational efficiency to an already compacted merged network. Additionally, we show how the conversion into SNNs happen, since SNNs are innately energy efficient compared to ANNs. In the following chapters we show our experimental setup, in Chapter 5, and the results of our approach, in Chapter 6.

Chapter 5

Experimental Setup

In this chapter, we explain the experimental setup we did to understand the effect of neural reuse on CNNs. We pay attention to the energy consumption between different types of CNNs which have the same origin. FP-CNNs and SNNs; then, QNNs and BNNs are used for our experiments. After explaining the nature of our experiments in this chapter, we discuss the results of the experiments and evaluate them in the next chapter, Chapter 6.

5.1 Hardware and Software Environment

All the experiments mentioned in this thesis are run on a 64-bit, *Linux* (Ubuntu 16.04) machine that has 4 `Intel(R) Core(TM) i7-7600U CPU @2.80GHz`. Every experiment is run solely and when the machine is in the idle state for a better comparison.

All the ANN experiments are run via programming language `Python`, *version 3.7.10*, with the ML algorithms library `TF` (Abadi et al., 2016a,b). TF-1.14.0[1] is used in the neural reuse focused experiments where low-level code manoeuvrability is needed. The high-level API of TF, `Keras` on TF-2.5.0[2] is preferred for the rest of the experiments for the purposes of readability and maintainability in the codes. We use the Python library `Larq`[3] (Geiger and Team, 2020) which was built upon Keras for the experiments that include BNNs. The TF module `TF-Lite`[4] is used for the other quantisation experiments.

[1]TF-1.14.0 API: `https://github.com/tensorflow/docs/tree/r1.14/site/en/api_docs`
[2]Keras API: `https://www.tensorflow.org/versions/r2.5/api_docs/python/tf/keras`
[3]Larq: `https://docs.larq.dev/larq/`
[4]TF-Lite: `https://www.tensorflow.org/versions/r2.5/api_docs/python/tf/lite`

We choose TF as the main software library in this research because of its versatility enabling the codes to be employed not only on the traditional hardware, but also on the edge devices. All the codes that are used to run the experiments are publicly available at `GitHub` [5].

5.2 Datasets

Modified National Institute of Standards and Technology (MNIST) (LeCun et al., 2010a), Fashion-MNIST (Xiao et al., 2017), and Sound20 (Chou et al., 2018) datasets are used in this research. The purpose of *Sound20* is to recognise different types of sounds and the rest of the datasets are image classification datasets. The details of the datasets and why they are chosen for this research are explained in this section.

All the datasets that are used in the thesis are publicly available. The image classification datasets can be obtained on the website of TF[6] and Sound20 dataset is available at `GitHub`[7].

5.2.1 MNIST

The first ever image recognition dataset, MNIST, emerged at the same time with the ancestor of the CNNs today, LeNet-5 (LeCun et al., 1998). MNIST is the modified version of National Institute of Standards and Technology (NIST) dataset, and it was born as a result of the *handwritten digit recognition research in the domain of post-codes* by Lecun et al. The dataset has been in active use by the computer vision and ANN community since 2010 (LeCun et al., 2010a) and the most commonly-used image recognition dataset of all times to this date. We prefer to use MNIST in our initial experiment (see Section 5.3.1) in terms of intelligibility in the proof-of-concept.

The MNIST dataset that is used in this research is a part of the TF's high-level API `Keras`[8], version *3.0.1*. The dataset comprises of 60,000 training and 10,000 test data, overall 70,000 images, all of which are 28×28 in *width*×*height*, encoded in grayscale format, handwritten digits. The dataset contains 10 categories/classes that are labelled from 0 to 9 to classify the handwritten digits. A sample of the images from the dataset is shown in Figure 5.1.

[5]Our main repository: `https://github.com/mervess/networks-on-board`

[6]TF-Datasets: `https://www.tensorflow.org/datasets/`

[7]Sound20: `https://github.com/ivclab/Sound20`

[8]MNIST: `https://www.tensorflow.org/versions/r2.5/api_docs/python/tf/keras/datasets/mnist`

Figure 5.1: A snapshot from the MNIST dataset.

5.2.2 Fashion-MNIST

Considering the over-use of the MNIST dataset throughout the years, and especially the near the top accuracy rates, i.e., 99%, that are being received via the dataset, a refreshing, yet an MNIST-like dataset, Fashion-MNIST was generated (Xiao et al., 2017). Fashion-MNIST has proved itself as a challenge to be mastered on by the resulted accuracy and error rates so far which is shown in Table 5.4. Moreover, compared to other SotA image classification datasets, e.g., *ImageNet* (Russakovsky et al., 2015), it is lighter on memory. Combining the two aforementioned reasons makes Fashion-MNIST dataset the perfect candidate for the energy concerned ANN tasks. Therefore, we run our main neural reuse experiment (see Section 5.3.2) using the Fashion-MNIST data.

The Fashion-MNIST dataset that is used in this research is a part of the TF's high-level API `Keras`[9], version *3.0.1*. The dataset, since inspired by and based on it, has technically the same features with the MNIST set. The only difference between the MNIST and the Fashion-MNIST is that the Fashion-MNIST consists of fashion items

[9]Fashion-MNIST: `https://www.tensorflow.org/versions/r2.5/api_docs/python/tf/keras/datasets/fashion_mnist`

instead of handwritten digits as images. The dataset contains 10 classes each labelled in a different fashion item category. The labels are listed in Table 5.1.

Table 5.1: Image classification labels and their description in the dataset Fashion-MNIST.

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

5.2.3 Sound20

What makes neural reuse interesting in terms of energy efficiency is that it is able to re-use neurons which originally respond to, or decipher from different peripheral sensory organs (see Chapter 3 for the details). Images correspond to the input of visual sensory organ, the eye, for us, human beings. Along with the image classification datasets used in the research which satisfies the *“visual sensory info”*, for the proof-of-concept purposes in neural reuse, a dataset representing a different input type was also necessary. The different input type is “sound” here which corresponds to the input of auditory sensory organ, the ear, in humans. We choose *Sound20* over other publicly available audio datasets in order to benefit from the pretrained weights of the LeNet model in the work of Chou et al..

The Sound20 dataset (Chou et al., 2018) is the product of two separate studies and consists of a mixture of instrumental audio and animal sounds (Hao et al., 2012). Labels 1-12 correspond to instrumental audio, and labels 13-19 correspond to animal sounds which overall constitute 20 sound classes listed as labels along with their descriptions in Table 5.2.

Table 5.2: Sound classification labels and their description in the dataset Sound20.

Sound20			
Instrumental Audio		**Animal Sounds**	
Label	Description	Label	Description
0	Drum_FloorTom	13	Bufo_Alvarius (a type of toads)
1	Drum_HiHat	14	Bufo_Canorus (a type of toads)
2	Drum_Kick	15	Pseudacris_Crucifer (a type of frogs)
3	Drum_MidTom	16	Allonemobius_Allardi (a type of crickets)
4	Drum_Ride	17	Anaxipha_Exigua (a type of crickets)
5	Drum_Rim	18	Amblycorypha_Carinata (a type of katydid)
6	Drum_SmallTom	19	Belocephalus_Sabalis (a type of katydid)
7	Drum_Snare		
8	Guitar_3rd_Fret		
9	Guitar_9th_Fret		
10	Guitar_Chord1		
11	Guitar_Chord2		
12	Guitar_7th_Fret		

5.2.4 Comparison of the Datasets, State-of-the-Art, and Discussion

We provide comparative information on datasets and discuss the SotA results in this section. Table 5.3 shows the overall numerical information of the datasets and Table 5.4 shows the SotA of the image datasets as they are commonly used in the literature.

Table 5.3: Numerical summary of the datasets that are utilised in this research.

Dataset	Data Count				Data Shape	Number of Classes	Dataset Size
	Total	Train	Test	Validation			
MNIST	70,000	60,000	10,000	-	$28 \times 28 \times 1$	10	21 MB
Fashion-MNIST	70,000	60,000	10,000	-	$28 \times 28 \times 1$	10	36.42 MB
Sound20	23,612	16,636	3,727	3,249	$32 \times 32 \times 1$	20	94.04 MB

The values in the "Data Shape" column on Table 5.3 hold different meanings for the image datasets, MNIST and Fashion-MNIST, and the sound dataset, Sound20. On one hand, regarding the image datasets, the common shape, 28×28×1, corresponds to *width*×*height*×*channel*. Because all the images in the datasets are in grayscale format, the *channel* value is 1 for them. On the other hand, the sound data in Sound20 are represented by a spectrogram to be processed by ANNs. Therefore, regarding the sound dataset, the data shape 32×32×1, corresponds to $x \times y \times null$. In a sound spectrogram, x axis refers to time, and the y axis refers to frequency of the sound. Because the sound data on a spectrogram is inherently two dimensional and does not involve a *channel* as image data does, we have addressed it as *null* recently. However, in the case of Sound20, the sound data is also assigned with a *channel* to match and to be processed with the image data at the same time. All the data used in this research are stored in `numpy` arrays in the Python environment.

Classical CNNs, that includes repetitive $Conv + Pooling$ layer blocks, yet with different arrangements, i.e., filter sizes may differ, stayed at the top of the SotA for a long time. However, with more data on the table and the emergence of more competent databases, these traditional approaches have begun to come short. This topic is also discussed in Chapter 2 - Section 2.3.1.1. We present the SotA in Table 5.4. Both *HVC* and *DARTS+Attention* in the Table approach completely differently to image classification, and the recommended ANN architectures are accordingly different.

Table 5.4: SotA in image classification datasets used in this thesis[10].

Methods		Dataset	Parameters	Top-1 Accuracy (%)	Error Rate (%)
HVC	(Byerly et al., 2021)	MNIST	1.5 M	99.87	0.13
DARTS+Attention	(Tanveer et al., 2020)	Fashion-MNIST	3.2 M	96.91	3.09

Byerly et al. use a **capsule design network** in which they benefit from **vector multiplications** instead of using conventional matrix multiplications in convolutional layers. Via benefiting from vectors, they are able to do element-wise multiplications which have less overhead computationally, compared to matrix multiplications. They name their method *Homogeneous Vector Capsules (HVC)*. Overall, their ANN architecture achieves the highest accuracy, *99.87%*, and the lowest percentage error rate, *0.13* on the MNIST dataset in the literature to this date.

Tanveer et al. employ **NAS** to decide on their network architecture and embeds ***Attention*** modules into the architecture to increase the overall accuracy of the network. NAS approaches in general are known to be computationally expensive since they generate a neural network from scratch without any supervision. We showcased the cost of

[10]The regarding SotA results are lastly checked on 18/10/2021. Below is our resources to check the leaderboards.
MNIST: `https://paperswithcode.com/sota/image-classification-on-mnist`
Fashion-MNIST: `https://paperswithcode.com/sota/image-classification-on-fashion-mnist`

a NAS approach earlier in Chapter 1, in Table 1.1. In order to increase the performance of the architecture search, and thus minimise the computational overhead, the authors prefer Differential Architecture Search (DARTS) which is a stochastic form of the NAS. What makes DARTS faster, yet eventually producing inferior results in accuracy compared to other stochastic methods, is that it benefits from *approximate computing*. Attention modules are introduced at this point as fixed computing modules in order to avoid the eventual accuracy loss. Overall, their ANN architecture achieves the highest accuracy, *96.91%*, and the lowest percentage error rate, *3.09* on the Fashion-MNIST dataset in the literature to this date.

Note that, as our research is not "accuracy" or benchmark focused, such SotA results based on accuracy are not among our related work. By demonstrating the SotA results regarding the used datasets in this research, we would like to give an integrated overview and offer clearer comparison between different approaches.

5.3 Architectures of the Neural Models

In the previous section, Section 5.2, we provide information over the datasets that are used in this research. In this section, we explain and discuss the CNNs that are applied onto the datasets during the experiments.

5.3.1 Singular Neural Network Experiment: MNIST on LeNet-5

Initial energy-based experiments start with MNIST on LeNet-5. Our reason to choose both LeNet-5 and the MNIST dataset is their simplicity and intelligibility, hence their power to stress the points in proof-of-concept.

We mention two LeNet architectures, LeNet 1 and 5, in Section 2.3. Note that, we only talk about LeNet-5 in this chapter since it has been the most recent version of the LeNet family. Therefore, we call LeNet-5 as only LeNet after this point for the simplicity in reading.

5.3.1.1 Architecture of the Full-Precision Model

We use almost the identical architecture of the original LeNet that was shown earlier in Figure 2.8b. Our modified LeNet architecture is shown in Figure 5.2.

Although, we preserve LeNet in our experiments as much as we can despite the technological differences between the times, our model has a few differences compared

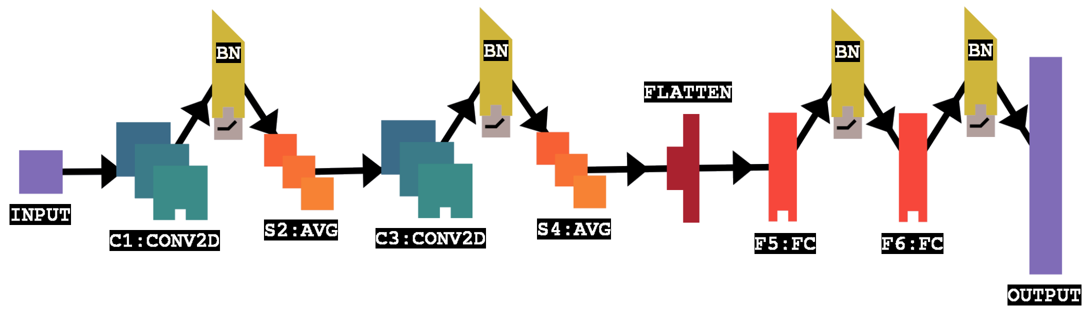


Figure 5.2: LeNet architecture that we use for the “singular” experiments.

to the original version of the model. These differences are listed below.

BN. BN technique (Ioffe and Szegedy, 2015) is relatively new compared to the rest of the advances in CNNs. The BN layer normalises the values in the previous layer in a model which leads to acceleration in training and increase in accuracy. Therefore, BN is usually added after the operation-heavy layers, e.g., a convolutional or an FC layer, in ANNs. For the aforementioned reasons, we update our LeNet with BN layers which are placed after each convolutional and FC layers in the model as seen in Figure 5.2.

Activation function. Activation function *tanh* is used right after every convolutional layer until F6 in the original LeNet (LeCun et al., 1998, p. 8). However, we prefer *ReLU* over *tanh* in all the experiments we run in this research because of the reasons below.

– It is computationally cheaper, yet more effective than *tanh*.

– Its algorithmic structure makes it more suitable to be used in SNNs that run on conventional hardware resources. Therefore, some ANN to SNN conversion tools explicitly states that they only work with ReLU (see Section 5.5.1).

ReLU is displayed as the little puzzle piece in grey colour that is directly attached to the BN layers in Figure 5.2.

Subsampling. We use *Average Pooling*, denoted as *AvgPool* and illustrated as *AVG* in Figure 5.2, as the subsampling method in LeNet. *AvgPool* calculates the average value of a given matrix and with this way makes a subsample of the input matrix. A complex subsampling method involving more calculations were used in the LeNet (LeCun et al., 1998, p. 7, §LeNet-5). The most commonly used subsampling methods today are *AvgPool* and *Maximum Pooling*. We choose to use *AvgPool* for the reasons below.

– Its functionality is closer to the subsampling function that was used in the original LeNet.
– Our FLOating-Point operations (FLOPs) tests showed that using *AvgPool* or *MaxPool* has no difference in computational complexity.
– *AvgPool* is proved to result more robustly when applied to even the most challenging datasets.

Layer C5. C5 is the final convolutional layer in the LeNet (see Figure 2.8b). However, it is different compared to prior convolutional layers in that it consists of 1×1 feature maps. In other words, under the right circumstances, it works as a FC layer (LeCun et al., 1998, p. 8). LeCun et al. explain the situation in the paper as, if an input larger than the size 32×32 is fed into the network, then such convolution operation would be necessary. We tested both a FC C5 and a convolutional C5 in our LeNet architecture, and observed that the overall complexity (in FLOPs) of the neural network remains the same, as the weight count of the layer does not change. However, we notice a minor accuracy drop[11] in the use case of a convolutional-C5. We keep the input size in that margin, 32×32 or smaller, in our experiments with LeNet, thus we choose to convert C5 into a FC layer. As illustrated in Figure 5.2, we, firstly, flatten the layer S4 to enable full connection. Thereafter, we replace the final convolutional layer, i.e., C5, in the original LeNet with a FC layer named F5.

After this point, the CNN models used in the experiments will be called with shortened names for the sake of intelligibility in the upcoming sections and chapters. The LeNet architecture described in this section will be addressed as *o*LeNet since it approximates to the original LeNet architecture.

5.3.1.2 Architecture of the Binarised Model

Although we intended to keep the *o*LeNet as it is because of the fundamental differences, described in Chapter 2 - Section 2.4.1, in training between the BNNs and

[11] A drop of 0.04%.

FP-ANN models, we had to make changes to the model to fit into the BNN structure. The differences between the neural network types are listed below.

– Quantising all the layers in a FP neural network results in dramatic drop in accuracy. Therefore, some of the layers are kept as they are depending on the design decision. Such layers might be the initial layer, final layer, and BN layers in a neural network.

– BNNs have to have BN layers to make them keep balanced.

– MaxPool works the best for them as they already consist of highly reduced values that are either -1 or $+1$.

– Activation functions do not work on them as they do in FP-ANNs. The activation function we used beforehand, ReLU, due to its no-negativity policy, does not help in BNNs in our case.

Because of the above reasons, during our binarised LeNet, which will be addressed as *b*LeNet from now on as being the binarised neural network in this research, implementation[12],

– We use MaxPool instead of AvgPool.

– We do not use any activation functions in the model.

– The neural network model is fully binarised except the BN layers.

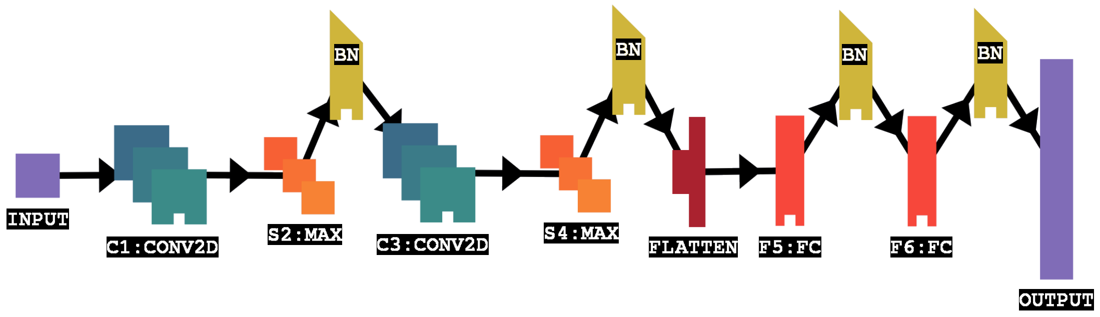
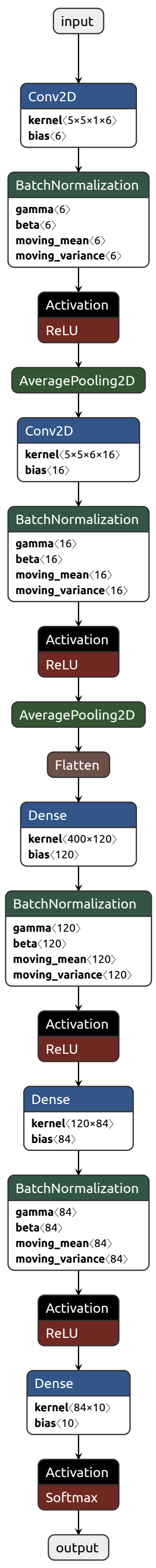


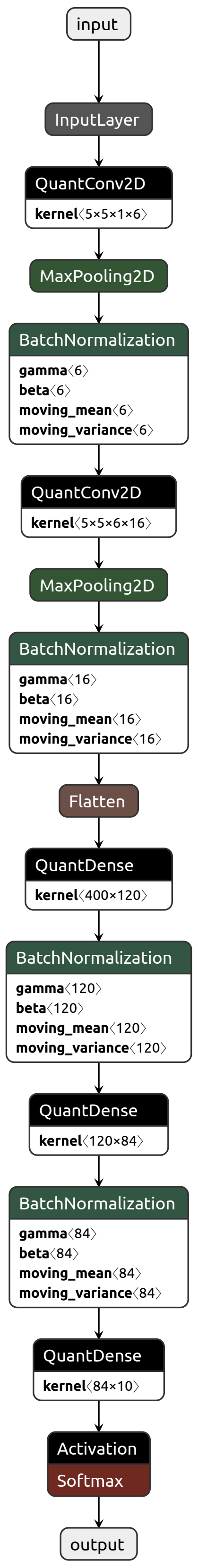
Figure 5.3: The BNN architecture of LeNet.

The *b*LeNet architecture adapted from *o*LeNet in light of the aforementioned differences is shown in Figure 5.3. Additionally, we present another form of *b*LeNet that not only showcases the layers of the architecture, but also showcases various numerical details of the model in Figure 5.4b. We also present the same form of *o*LeNet

[12]We tried several other combinations of the LeNet architecture before settling into the final version of the *b*LeNet. Because this version resulted with the highest accuracy and the lowest loss rates, we have kept it.



(a) *o*LeNet



(b) *b*LeNet

Figure 5.4: Full precision and binarised forms of the same CNN model, LeNet.

next to *b*LeNet for a better comparison in Figure 5.4. The numerical details of both of the models; the kernels, filters and channel dimensions are kept the same as in LeNet (LeCun et al., 1998).

Perhaps the first noticeable difference between *o*LeNet and *b*LeNet in Figure 5.4 is the sizes of the models. Having no activation layers, as in ReLU in *o*LeNet, *b*LeNet has less layers, and hence, is smaller architecturally compared to *o*LeNet. Given the fact that BNNs is a special type of QNNs, weights of the layers that require intense computation such as convolutional and FC layers are quantised into -1 or $+1$ for energy efficiency purposes in BNNs. Therefore, the names of convolutional and FC layers have the *Quant* prefix before their names in Figure 5.4b. Note that, because they do not carry weights, pooling layers and the *Flatten* layer are not in binarised forms in *b*LeNet. The only layers that are not binarised despite carrying weights are the BN layers in the architecture. Finally, the reason for the *InputLayer* attachment in Figure 5.4b following the *input* is to normalise the input data coming from the datasets to fit the BNN model.

5.3.2 Merged Neural Networks Experiment: Fashion-MNIST and Sound20 on a Modified LeNet

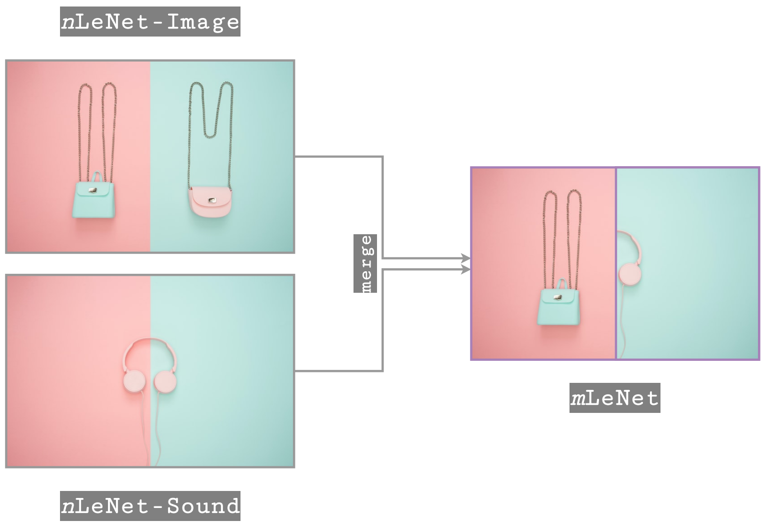


Figure 5.5: Illustration of the *merge* operation: *nLeNet*s building the neural reuse architecture.

We apply the same architecture that was used by Chou et al. for the neural reuse experiments that merges two CNNs: one recognising images and the other recognising sounds. The backbone of this architecture is a modified LeNet which is one FC layer shorter than *o*LeNet. Additionally, it does not contain any BN layers. After this point, the aforementioned singular LeNet model will be called as *n*LeNet considering it is used in the neural reuse experiment; and the overall neural reuse architecture will be called as *m*LeNet because it merges two *n*LeNet models. We illustrate the merge leading to *m*LeNet in Figure 5.5, and the interior of *m*LeNet[13] is displayed in Figure 5.6.

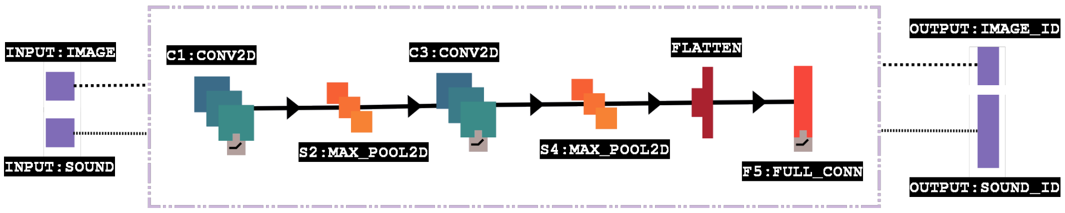


Figure 5.6: The interior of *m*LeNet constructed by two *nLeNet*s.

*n*LeNet has more units and kernels in the convolutional and FC layers than *o*LeNet does, which makes *n*LeNet computationally more intense and more expensive. The numerical details of the *n*LeNet architecture, e.g., kernel counts, are showcased in Table 5.7. We also showcase the numerical details of *o*LeNet in Table 5.6 and discuss our reasons to choose *n*LeNet over *o*LeNet to be used in the *m*LeNet architecture in Section 5.6.

Considering *m*LeNet is a “merger” that combines two separate CNNs in one body, it has two inputs and two outputs. Both inputs have the same dimensions, $32 \times 32 \times 1$. Fashion-MNIST comes with $28 \times 28 \times 1$ data format in TF. However, because Sound20 holds $32 \times 32 \times 1$ data, in order to keep them aligned, the merger design is fitted towards the larger data format. The outputs also differ in size, as Fashion-MNIST has 10, and Sound20 has 20 output classes. The graphics of Figure 5.6 is aimed to reflect these features of the neural reuse architecture.

5.4 Training Details of the Singular Networks

Only singular CNNs, *o*LeNet and *b*LeNet, are trained within this research. We use the MNIST dataset for the training. Because energy-efficiency is the main goal of this

[13]The stacked layers o *n*LeNet and *m*LeNet are the same.

research, our aim is to preserve energy as much as possible, thus we have taken some precautions to shorten the training period.

Along with preserving the energy, keeping the learning levels of the model high is also paramount. A threat towards a healthy learning in ANNs is known as the "*-fitting*" problem including *over* and *under* fitting. *Over-fitting* occurs when the learning level of the model is unsatisfactory in a sense that it does not learn, but memorise on the training data, hence can not generalise its knowledge on unseen data. The outcome of over-fitting can be observed as while the model produces high accuracy results on training data, these results get lower on test data. While over-fitting causes undesired results on the test data, *under-fitting* causes the same results on both training and test data. Under-fitting comes along with both low accuracy and high error rates on the overall dataset. Therefore, in order to avoid over and under fitting of the data, we have taken some precautions as well. The precautions taken to avoid both of them are listed below.

BN layers. Because BN, as its name implies, normalises the critical information in an ANN, it is seen as an inner shield towards the "*-fitting*" risks. However, adding BN layers has more advantages than to solely avoid the "*-fitting*" risks for ANNs. It is known to fasten the training, hence to reduce the training time which is a crucial factor in energy-saving. Due to these advantages, we apply BN to *o*LeNet and *b*LeNet.

Validation data. We use validation data along with training-only data when training the networks. The validation data is supplied from the training data itself in the amount of 20%. During the training, the validation data offers more options to the neural network to make more comparisons and update itself via back-propagation after every epoch, thus limits the chances of memorisation of data.

Early stopping in training. We use a fail-safe mechanism when training the CNNs, i.e., *EarlyCallback* mechanism in Keras, which halts and concludes the training if/when above aforementioned risk cases appear. This mechanism, by using the validation data, keeps track of the training *loss* (see Chapter 6 - Section 6.1.1.2) of a model. If the loss gets to increase after a certain point, the mechanism interrupts the training and finalises it. If such aforementioned risk cases do not appear, the training continues as long as the number of epochs assigned in the beginning of the training.

5.5 Pretrained Model Conversions

We explain conversion details of the pretrained models in this section. The start point of every conversion is pretrained FP-CNNs in this scope. In Section 5.5.1, conversion into SNNs; and in Section 5.5.2, conversion into QNNs are discussed.

5.5.1 Conversion of Full-Precision Neural Model into a Spiking Model

We experiment with both a rate coding, (Wu et al., 2022), and a temporal coding, (Stöckl and Maass, 2021), conversion techniques for our SNNs conversion experiments as illustrated in Figure 5.7. We paid attention to their claimed SotA performances and also evaluated them for ourselves based on the metrics we use that are mentioned in Section 6.1 when choosing these two representatives among the available research over ANN-SNN conversions. Note that, as we support the open-source projects, the straightforward usability and trivial adaptability of both techniques to our code also appealed to us from the software engineering point of view.

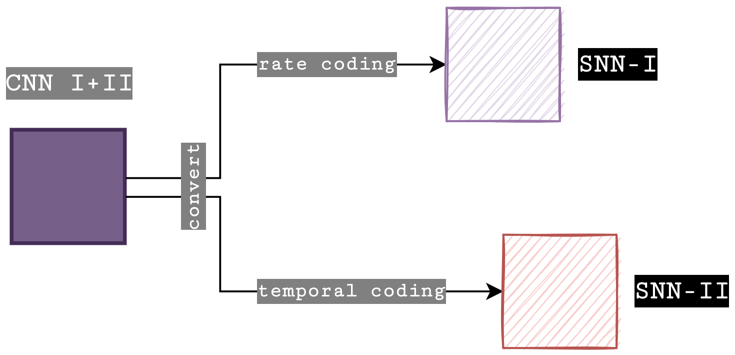


Figure 5.7: Pretrained FP-CNN to SNN conversions.

The rate coding based SNN converter by Wu et al. works on the high-level API of TF, Keras. It takes a trained CNN model and its training dataset as the input. The trained model could be fed as a pretrained and stored network model in the format of `.h5`, or as a freshly trained one in the same run-time. We apply the former method. The dataset is used to regulate the activation function of the new SNN model in this converter as the used images are always in discrete format. *ReLU* activation function regulator is readily built-in within the converter. The output of the converter is a Keras model with custom, spiking layers.

The temporal coding based SNN converter by Stöckl and Maass works on the low-level API of TF and on the functional API[14] of Keras. As the converter only processes and makes changes on the activation layers of a given neural network, it only requires a run-time call to its main function prior to uploading (solely) the weights of the pre-trained model. *Sigmoid* and *ReLU* activation function examiners are readily built-in within the converter. The output of the converter stays the same as the input by appearance, except the inner, i.e., numerical, changes in the activation layers.

Both of the converters are tested on *o*LeNet and *n*LeNet each of which comprises a singular CNN. Thereafter, by examining the results in Section 6.2.2.2, we choose to use the temporal coding based converter (Stöckl and Maass, 2021) during the experiments that involve *m*LeNet.

5.5.2 Conversion of Full-Precision Neural Model into a Quantised Model

We use TF-Lite which is an ANN optimisation module inside TF in order to convert our FP models into QNNs. Since our research concerns utilising the pretrained neural networks, the work of TF-Lite is to compress and hence to make the CNNs "lighter" in size enabling them to fit in and run on computationally limited devices, e.g., mobile phones, smart watches etc, in our research. TF-Lite makes such work possible by converting ANNs into QNNs.

Quantisation is available in both floating point, *float32*, and integer, *int8*, data types in TF-Lite[15]. A floating point quantisation refers to storing a 32-bit data, *float32*, in a 16-bit data format, *float16*. An integer based quantisation may refer to either degrading a 32-bit integer data into 16-bit or a 8-bit data format. Quantisation methods that we prefer to test in this research involves "from 32-bit float to 8-bit integer" quantisation. In other words, the quantisation that we mention for our experiments comprises a full integer, 8-bit, quantisation. This quantisation format stores the values in *int8* data type that otherwise would remain in type *float32* and it restricts the data to fit the range of [-128, 127][16].

We make our experiments over three quantisation formats as illustrated in Figure 5.8. All of them involve integer quantisation and two of them include floating

[14]Keras has two APIs to model ANNs: *Functional* and *Sequential* API.

[15]Resource for the information in this section: `https://www.tensorflow.org/lite/performance/model_optimization`

[16]This range is specific for activations and inputs of a model. The weights in the model are defined within the range of [-127, 127] (Jacob et al., 2018).

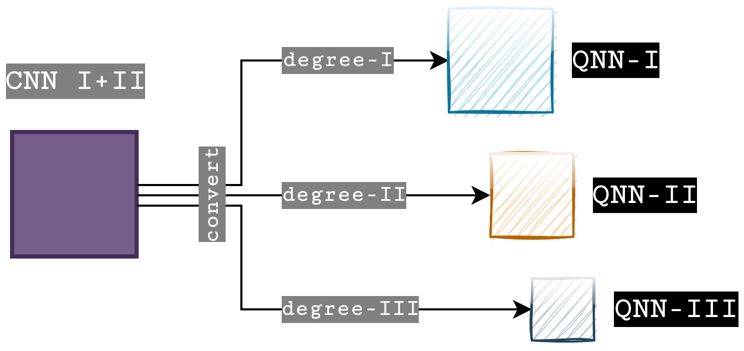


Figure 5.8: Pretrained FP-CNN to QNN conversions.

point data. We apply *Post-training integer quantisation* onto pretrained FP-CNNs. This quantisation method requires a sample of unlabelled training data and results with only a small accuracy loss[17], hence we prefer such method. The three quantisation method that we use during our experiments are explained below.

Degree I. Only the weights of the model are quantised into *int8*.

Degree II. The input and output of the model remain as *float32*. Everything else in the model is quantised into *int8*.

Degree III. All the features of the model are quantised into *int8*.

5.5.2.1 An Attempt to Convert a Full-Precision Model Directly into a Binarised Model

There hasn't been any tool or research that targets directly converting a FPs-CNN into a BNN in the literature so far. BNNs are trained from scratch to make the most of them. Moreover, conversion into binary directly from full-precision results in a great deal of accuracy loss when such a conversion occurs. However, for the sake of showing how much accuracy loss occurs and to show the ineligibility of such conversions, we added these results of a BNN conversion experiment. Therefore, we make an attempt for a direct translation between CNNs and BNNs.

$$x^b = Sign(x) = \begin{cases} +1 & \text{if } x \geq 0, \\ -1 & \text{otherwise.} \end{cases}$$

[17]Model optimisation — TF-Lite: `https://www.tensorflow.org/lite/performance/model_optimization`, final access on 04/10/2022.

We use the deterministic formula by Hubara et al. that is shown above to convert the pretrained data values of a model into binarised format. The above formula converts a FP value, x, into a binarised value, x^b, resulting the value to be either $+1$ or -1. We exclude the BN layers and bias values in the CNN when converting, which naturally occurs in any BNN training. However, such conversion results with a significant loss in both accuracy and measured error of the model. The experiment results are showcased and discussed in Chapter 6 - Section 6.2.2.1.

5.6 Discussion of Full-Precision Model Architectures

In this section, we would like to highlight the reason of choosing two different LeNet-like network models, i.e., *o*LeNet and *n*LeNet, for the experiments explained in the previous sections. *o*LeNet, described in Section 5.3.1.1, and *n*LeNet, described in Section 5.3.2, are the shortened names of the LeNet models to enable distinction between them. We summarise the characteristics of the two aforementioned FP LeNet architectures in Table 5.5 below.

Table 5.5: Summary of the FP-LeNet architectures that constitute the topic of this section.

Alias	Meaning	Origin (in *Section*)	Illustration (in *Figure*)	
			Layer-wise	Numerical
*o*LeNet	our LeNet architecture which approximates to the original LeNet-5 architecture	5.3.1.1	5.2	5.4a
*n*LeNet	a LeNet like singular architecture that is used in the neural reuse experiment	5.3.2	5.6	6.2a

The initial decision that we made was choosing a neural reuse experiment architecture and we chose the network that was used by Chou et al. because it was readily available, i.e., pretrained. In other words, *n*LeNet emerged earlier than *o*LeNet did in this research. We then chose not to use *n*LeNet during our singleton experiments that require training from scratch considering the following examinations.

The “Layer-wise” column under the title “Illustration” on Table 5.5 refers to the illustrations of the models that display how their layers are stacked. Furthermore, the “Numerical” column next to “Layer-wise” on Table 5.5 refers to the architectural display of the two LeNet models including numerical information supported with figures.

Table 5.6: Analysis of *o*LeNet that is described in Section 5.3.1.1.

Layers		Data Dimensions	Weights	
			(N)	(%)
Input	···	$28 \times 28 \times 1$		
Conv2D (C1)	\\\|/	– – – – – – –	156	0.2
	⋮	$28 \times 28 \times 6$		
BN + ReLU	μ|σ	– – – – – – –	24	0.0
	⋮	$28 \times 28 \times 6$		
Avg_Pool2D (S2)	Y	– – – – – – –	0	0.0
	⋮	$14 \times 14 \times 6$		
Conv2D (C3)	\\\|/	– – – – – – –	2,416	3.9
	⋮	$10 \times 10 \times 16$		
BN + ReLU	μ|σ	– – – – – – –	64	0.1
	⋮	$10 \times 10 \times 16$		
Avg_Pool2D (S4)	Y	– – – – – – –	0	0.0
	⋮	$5 \times 5 \times 16$		
Flatten	|||||	– – – – – – –	0	0.0
	⋮	400		
Dense (F5)	X	– – – – – – –	48,120	76.9
	⋮	120		
BN + ReLU	μ|σ	– – – – – – –	480	0.8
	⋮	120		
Dense (F6)	X	– – – – – – –	10,164	16.2
	⋮	84		
BN + ReLU	μ|σ	– – – – – – –	336	0.5
	⋮	84		
Dense (Output)	X	– – – – – – –	850	1.4
Softmax	···	10		

Table 5.7: Analysis of *n*LeNet that is described in Section 5.3.2.

Layers		Data Dimensions	Weights (N)	Weights (%)
Input	$\cdots$	$32 \times 32 \times 1$		
Conv2D (C1)	\\\|/	$------$	832	0.0
ReLU	$\vdots$	$32 \times 32 \times 32$		
Max_Pool2D (S2)	Y	$------$	0	0.0
	$\vdots$	$16 \times 16 \times 32$		
Conv2D (C3)	\\\|/	$------$	51,264	1.2
ReLU	$\vdots$	$16 \times 16 \times 64$		
Max_Pool2D (S4)	Y	$------$	0	0.0
	$\vdots$	$8 \times 8 \times 64$		
Flatten	|||||	$------$	0	0.0
	$\vdots$	4096		
Dense (F5)	X	$------$	4,195,328	98.5
ReLU	$\vdots$	1024		
Dense (Output)	X	$------$	10,250	0.2
Softmax	$\cdots$	10		

For the purposes of examining the models in terms of their computational overhead in this section, network architectures including their weight distributions are detailed in Table 5.6 and 5.7 respectively. Note that both tables display the *layer* names, in the column "Layers", as they are in `Keras`. The architecture of *n*LeNet is layer-wise smaller compared to *o*LeNet as seen in Table 5.7. However, it is densely packed, and overall has more weights than *o*LeNet. Its input dimensions, 32×32, is larger than the dimensions of *o*LeNet which is 28×28 and that aspect solely puts $\approx$ 1M[18] extra weights, and $\approx$ 8.5M[19] more FLOPs onto *n*LeNet.

The *neurons* of *n*LeNet, a.k.a *units* in Dense layers and *kernel counts* in Conv2D layers in ANNs, are also larger in quantity compared to *o*LeNet which is the root cause of overhead in any given ANN. The kernel counts of Conv2D layers are displayed as the third dimension from the left side, and the units of Dense layers are displayed solely in the "Data Dimensions" column of Table 5.6 and 5.7.

Layer **F5** holds the most of the weights in both architectures as shown in the Tables 5.6 and 5.7. We see that the share value of F5, 98.5 %, is even larger in *n*LeNet. However, from the FLOPs point of view which is the metrics of computational complexity, F5 is not the heaviest layer in neither of the models. The top three layers in FLOPs is listed in a descending order and shown in Table 5.8. C3 has the most computational complexity in both LeNet models according to results in the Table 5.8.

Table 5.8: Top three layers in FLOPs in *o*LeNet and *n*LeNet.

	*o*LeNet		*n*LeNet	
	Layer	FLOPs	Layer	FLOPs
1	C3	480.00 K	C3	26.21 M
2	C1	235.20 K	F5	8.39 M
3	F5	96.00 K	C1	1.64 M

Table 5.9: Top two operational results based on FLOPs in *o*LeNet and *n*LeNet.

	Operation	*o*LeNet		*n*LeNet	
		FLOPs	%	FLOPs	%
1	Conv2D	715.20 K	83.14	27.85 M	76.60
2	MatMul	117.84 K	13.70	8.41 M	23.13

We also examined the operations that have the most computational complexity, and hence, that cause the overall overhead in both LeNet architectures. These operations

[18] 4,257,674 *(current weights)* − 3,274,634 *(weights to be with* 28×28 *inputs)* = 983,040

[19] 36,361,276 *(current FLOPs)* − 27,844,156 *(FLOPs to be with* 28×28 *inputs)* = 8,517,120

are revealed as convolution operation, *Conv2D*, and matrix multiplication, *MatMul*, and shown along with their costs as FLOPs in Table 5.9. Both *Conv2D* and *MatMul* are the node names in the TF graph and hence portrayed as such in the table. Conv2D operations are caused by the convolutional layers, and MatMul operations belong to the FC layers in the architectures. Although, they seem light in terms of weights, convolutional layers, as being the backbone of any CNNs, are also the lead overhead which is seen clearly in Tables 5.8 and 5.9.

Table 5.10: The computational and memory complexity of the used CNN models.

Model	FLOPs	Weights			Model Size
		Total	Trainable	Non-Trainable	
*o*LeNet (Section 5.3.1.1)	860.28 K	62,610	62,158	452	242.80 KiB
*n*LeNet (Section 5.3.2)	36.36 M	4,257,674	4,257,674	0	16.24 MiB

All our examinations proved that computationally *n*LeNet has more complexity over *o*LeNet which makes it more expensive to train and test which includes inference stage as well. The overall complexity values are summarised in Table 5.10[20]. Because of such overhead, we chose to use *o*LeNet in singleton experiments in which we needed to train the networks from scratch.

5.7 Summary

In this chapter, we have presented our setup, showcased the CNN architectures, detailed the datasets to be used with the CNNs, and explained the experiments. At the same time, we have also discussed our reasons to prefer any aforementioned piece over other available options.

Table 5.11: Description of LeNet architectures that are used in the conducted experiments in this chapter.

Alias	Meaning	Origin (in *Section*)
*o*LeNet	our LeNet architecture which approximates to the original LeNet-5 architecture	5.3.1.1
*b*LeNet	our binarised LeNet architecture	5.3.1.2
*n*LeNet	a LeNet like singular architecture that is used in the neural reuse experiment	5.3.2
*m*LeNet	two merged *n*LeNet models comprising the neural reuse experiment	5.3.2

We use the LeNet-like architectures during our experiments since the variety in their layers and model size make them perfect fits to test our approach in this thesis.

[20]Note that the "Non-Trainable" parameters in Table 5.10 come from BN layers in the *o*LeNet.

Table 5.12: Summary of the experimental setup of this research.

CNN Architecture	Experiments				Datasets		
	Training from scratch	Pretrained Model Conversions					
		into SNNs	into QNNs	into BNNs	MNIST	Fashion-MNIST	Sound20
*o*LeNet	✓	✓	✓	✓	✓		
*b*LeNet	✓				✓		
*n*LeNet		✓	✓			✓	✓
*m*LeNet		✓	✓			✓	✓

We have mentioned four different LeNet architectures in this chapter differentiated by prefixes. We summarise the aforementioned LeNet architectures in Table 5.11 along with the sections in this chapter that introduced them. Additionally, we overview the architectures along with the datasets and the details of the experiments in Table 5.12. In the next chapter, Chapter 6, we proceed to report and evaluate the experiments that are introduced in this chapter.

Chapter 6

Evaluation and Results

We consider not only raw accuracy, but also rejection, training time, recognition time, and memory requirements.

LECUN ET AL., 1995

Previously, we gave an overview of the problem in the domain of CNNs in Chapter 2 and described our hypothesis as our solution offer to this problem, firstly in Chapter 3 theoretically, then in Chapter 4 algorithmically. Additionally, we explained the technical setup for our experiments that will help us prove our hypothesis in Chapter 5. We evaluate the hypothesis via experiments in this chapter.

We aim to provide as much detailed and well-measured research as possible, especially in the evaluation part. During the literature review for this thesis, we observed that earlier papers had a tendency to be more transparent. For example, the authors report the "computational complexity" in the earliest form of a trained ANN paper, *NETtalk* (Sejnowski and Rosenberg, 1986). The overall hours that took to train the ancestor of the modern CNNs were reported in the *LeNet-5* paper by LeCun et al. (1998). Nowadays, we always come across solely the "accuracy" metric, which is, without a doubt, a crucial metric to measure the success of ANNs. However, the vital question here is: *is it enough*? Therefore, we use two sets of metrics when evaluating our hypothesis: one set to measure the success of CNNs as usual in the ML domain including accuracy, and another set to measure their energy efficiency. Even though we don't come across large metrics sets as widely used in today's research, they were indeed an integrated part of the metrics in the early days of CNNs as LeCun et al. mentions. That is what inspires our evaluations as we believe it contributes to the transparency of any research.

6.1 Metrics

We divide the metrics into two sub-groups. On one hand, since our hypothesis is over CNNs which is a field under ML, ML and specifically ANN related metrics are used. On the other hand, because "the energy" is the main concern in the hypothesis, energy-specific metrics are used as well.

We have two aspects regarding our metrics usage. Firstly, we do not measure energy consumption in this research. We run all our experiments on a conventional machine which solely has a CPU; accordingly, we measure metrics which indicate energy consumption of the network models for better comparison in between them and for understanding the models. Secondly, we do not report all the metrics for all the neural networks. Metrics will be reported based on applicability which will be elaborated in the following sections.

6.1.1 On Evaluation of Artificial Neural Networks

Metrics that are used to evaluate the experiments in Section 6.2 are categorical accuracy, loss, and error rate. They are all calculated via Python library TF when evaluating the networks on the test datasets. An important note here is that, the accuracy is the higher, and the loss and error rates are the lower, the better.

We choose accuracy as the lead metric in this research. We believe that any metric should not be employed solely to evaluate a research, and should be accompanied by other metrics as we did so in this thesis. However, regardless of how small in size, how fast in inference, or how less computationally complex a neural model may be, if it is not accurate, then the model needs to be reconsidered before applying to the real world. Therefore, we firstly check accuracy levels of any model that is used in this research, before checking other metrics. The ultimate goal is maintaining the accuracy as high as possible and finding the best fit concerning energy efficiency for the end device.

$$\begin{aligned} i &= \text{index of test set,} \\ n &= \text{total number of data in the test set,} \\ t &= \text{target output in the test set; true value,} \\ o &= \text{predicted output by our system.} \end{aligned} \tag{6.1}$$

In Equation 6.1, we define the common symbols that take place in the formulas (Mitchell, 1997; Fürnkranz et al., 2011) of the metrics in this section. i, t, and o values vary in the formulas. However, n is a fixed number and its value set was shown in Chapter 5 - Table 5.3 earlier. For instance, if the evaluation runs on *Sound20* dataset, the corresponding n value is going to be 3727.

6.1.1.1 Accuracy

Accuracy is the most common metric among the experiments as it is the mostly preferred in the literature to evaluate CNNs. It has a Top-k category as well, e.g., Top-1 and Top-5 categorical accuracy, that is showcased in the benchmarks, such as the ILSVRC. k symbolises the top label count in which the correct category label is found to classify the given instance. Top-1 accuracy basically returns a *True* or *False* answer to the classification problem; when the *accuracy* is used solely, it means the Top-1 accuracy. Top-5 indicates that the true label is among the first 5 returned answers by the network which obviously increases the accuracy levels of the model in evaluation. Top-1 and Top-5 are the mostly referred ones among the accuracy metrics, hence we use both of them to showcase our results.

$$A(t,o) = \frac{1}{n}\sum_{i=0}^{n-1}\sum_{j=1}^{k} 1(t_i = o_{ij})$$

Above formula is used to calculate Top-k accuracy in our research. When k in the second part of the formula equals to 1, the overall formula turns into calculating the traditional accuracy that is Top-1.

6.1.1.2 Loss

The *loss* is the result of a differential function which determines the fitting value between a network model and a dataset, and produces their corresponding *misfit* value. As our tests are over the classification problem in this thesis, the loss function we refer to is *Categorical Cross-Entropy* which is also recommended for efficient use in calculation by Byerly et al..

$$L(t,o) = -\sum_{j=0}^{m-1} t_j \cdot \log o_j$$

We compute the *loss* value whose formula is shown above via TF-Keras. m represents the output size of a model, and j is the index of the output set in the formula.

We summarised the models along with their corresponding datasets in Chapter 5 - Table 5.12. To illustrate, given the model is *n*LeNet and the dataset is *Sound20*, based on Table 5.3, the corresponding *m* value would be 20.

6.1.1.3 Error Rate

The *error rate* metric refers to the (%) proportion of the miscategorised results by the neural network. Mean Squared Error (MSE) is used as the *error rate* function in this thesis. We observe in the literature that all the early neural network papers share their MSE results and that includes the famous LeNet papers since MSE has always been a crucial metric to evaluate the ML techniques. Therefore, we also share the MSE results as the *error rate* of our networks. The below formula is used to calculate MSE in our experiments.

$$E(t,o) = \frac{1}{n}\sum_{i=0}^{n-1}(t_i - o_i)^2$$

6.1.2 On Evaluation of Energy Efficiency

Because energy measurements vary based on the location on earth and the resources that are used for such measuring, we choose to use energy-efficiency indicator metrics rather than using formulas leading to measured energy results in *joule (J)* or *kilowatt/hour (kWh)*. Considering their relation to the energy consumption of ANNs today, discussed in Chapter 1 - Introduction, especially big technology companies, e.g., Google, aims to be more transparent, and shares their use of energy in their world-wide distributed data centres[1,2]. Such numerical data shows even for the same company, variety occurs in energy consumption and efficiency just based on placing. Green-AI (Schwartz et al., 2020) principles also state the unreliability of metrics such as *carbon emission* due to its being location-reliant; and *electricity usage* due to its being hardware-reliant, despite formulas are offered (Patterson et al., 2021) to encourage such metric usages.

The lead metrics that we use to evaluate energy efficiency are Multiply and ACcumulate operations (MACs) and FLOating-Point operations (FLOPs) in this research. Such metrics are considered as the most reliable and accurate indicators over other metrics towards energy efficient systems in the literature (Schwartz et al., 2020). As our

[1] `https://www.google.com/about/datacenters/efficiency/`
[2] `https://cloud.google.com/sustainability/region-carbon`

research emphasises the energy efficiency, our metrics include the measured time and model size in the machine's memory as well. The comparisons based on time measurement are over *x*-times of the base value within that category. We use *binary-byte* format to report model sizes, specifically Kilo Binary Byte (KiB) and Mega Binary Byte (MiB) as the sizes of the models used in this research do not exceed MiBs.

We offer a description for MACs and FLOPs in Section 6.1.2.1. Thereafter, we explain the measurements of time and model size in Section 6.1.2.2. Note that, all the metrics in this set are the lower, the better. Finally, a reminder here is that we do not apply all the metrics to all the comparisons in this research. For instance, if *inference time* is the same for all the compared networks in an experiment, then we do not report inference time since it loses its comparability value.

6.1.2.1 MACs and FLOPs

We use MACs and FLOPs as the main energy-efficiency references in this research. MACs are equivalent to the half of the count of FLOPs in the FP-ANNs because they comprise of two calculations in one: *multiply* and *accumulate* together. We report the MACs for the BNNs and QNNs, and the FLOPs when the FP-CNNs are the main subject of the experiments in this chapter. We benefit from the *larq* (Geiger and Team, 2020) and TF (Abadi et al., 2016b) to measure MACs and FLOPs of the neural networks.

6.1.2.2 Time and Model Size

We measure *training*, *conversion*, and *inference time* of the CNNs during the experiments. Because the machine we use for our tests include only a CPU and all the metrics are measured on the same machine, we benefit from the traditional measurement techniques to measure system time. The training time is the time passed during the determination of the weights of a model. The conversion time comprises the time when converting the pretrained CNN into other forms of neural networks, such as QNNs and SNNs. The inference time is the time when applying the weights to the model to determine the output. We report training and conversion times only in the discussion section as they are realised theoretically once throughout the lifetime of a neural network. We report inference time of the models whenever the comparison is applicable. Note that, during the measurement of training time, training data is used, and during the measurement of inference time, test data is used.

Model size is measured in *binary-byte* format as the model takes place in the storage. We use larq's summary tool to measure model sizes for the tests in Sections 6.2.1 and 6.2.2. As the neural reuse test requires the usage of low-level TF which is not compatible with larq, we measure the model size of the outcome of the test in Section 6.2.3 directly via memory storage.

Note that, when presenting the measurements of *time*, we use *x* for the base value instead of comparing the measured, real-number values. The reason behind this idea is that the "time" is considered as being not as exact as other metrics in the literature because the results may change when the tests are run on a different machine, or when the same tests are repeated on the very same machine (Schwartz et al., 2020). Since we use the same machine having only a CPU throughout the tests, running the tests 10[3] times and taking the average of the outcome, we find it appropriate to share our measurements as *x*-times by comparing them to one another. We also provide our system requirements in Chapter 5 - Section 5.1, and details of all the metric measurements in Section 6.1. Overall, we hope that reporting the results of all our aforementioned metrics will shed light for the reader and for researchers who would like to replicate our experiments.

6.2 Experimental Results

This section will report the results of and evaluate the experiments divided into three different sections: evaluating the outcomes of the models that are trained from scratch in Section 6.2.1, converting the pretrained models into different forms for energy-efficiency in Section 6.2.2, and the evaluation of the overall proposed approach in Section 6.2.3.

The four over six of the LeNet models on which we run our experiments were described and summarised in Chapter 5 - Section 5.7. Before elaborating the experiments, we update the summary in Table 5.11, and add the two final LeNet models that will be introduced in this chapter as displayed in Table 6.1. Both *q*LeNet and *s*LeNet are obtained via conversion of a pretrained model and such pretrained models are *o*LeNet and *n*LeNet in our experiments.

[3]Number 10 is chosen empirically here.

Table 6.1: Description of LeNet architectures that are used in the conducted experiments in this chapter.

Alias	Meaning	Origin (in *Section*)
*o*LeNet	our LeNet architecture which approximates to the original LeNet-5 architecture	5.3.1.1
*b*LeNet	our binarised LeNet architecture	5.3.1.2
*n*LeNet	a LeNet like singular architecture that is used in the neural reuse experiment	5.3.2
*m*LeNet	two merged *n*LeNet models comprising the neural reuse experiment	5.3.2
*q*LeNet	a quantised LeNet model that is obtained via converting a pretrained LeNet model	6.2.2.1
*s*LeNet	a spiking LeNet model that is obtained via converting a pretrained LeNet model	6.2.2.2

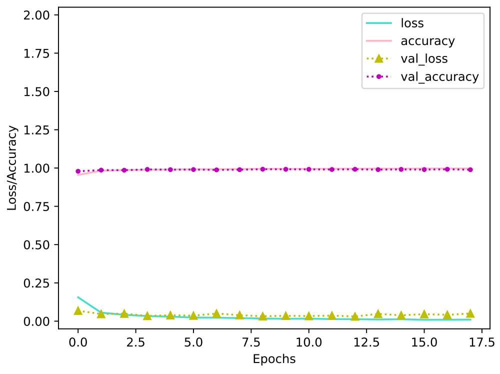
6.2.1 Training from Scratch: Full Precision vs. Binarised Neural Model

The weights of a BNN comprise instead of conventional *32-bit Float* weights, *4-bit Integer* weights which are either -1 or $+1$, ergo they offer solutions for edge devices and micro-controllers. Although BNNs promise to deliver 4-bit weights, the micro-controllers today do not include built-in 4-bit data storage units, hence instead they store such values in 8-bit data types (Banbury et al., 2021). Keeping that in mind, and considering our machine also stores such weights as 8-bit, we run our BNN experiment on our CPU machine.

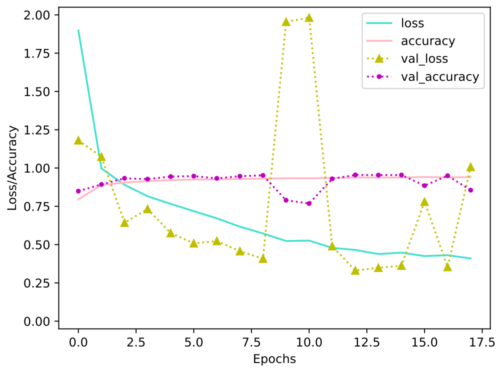
The only training that was made during this research is the training of the LeNet model exploiting the fact that it is a small CNN model, compared to deeper and wider CNNs today, since its being their forerunner. We train an FP-LeNet, *o*LeNet, and its counterpart BNN, *b*LeNet, to compare their differences which are the most obvious during training.

The main reason of such training is because when a direct conversion of CNN to BNN occurs, valuable metrics described in Section 6.1.1 drops tremendously (see Section 6.2.2.1 for the experimental details). Therefore, a BNN is only obtained by training via the techniques available today. We demonstrate the overall training data history of the two *LeNet*s in Figure 6.1. Both models are trained and tested on MNIST data for this experiment.

The data graphics that are displayed side-by-side in Figure 6.1 demonstrate the fundamental differences between FP-CNNs and BNNs clearly. While the training procedure of *o*LeNet goes smoothly when both measuring the loss and accuracy, it is on the contrary in the BNN equivalent version of the network, *b*LeNet. Maintaining the *loss* metric especially pose a challenge for the *b*LeNet as we observe in Figure 6.1b. It is important to remember that the metric *loss*, Section 6.1.1.2, measures the fitness of



(a) FP-CNN, *o*LeNet



(b) BNN, *b*LeNet

Figure 6.1: Data history illustrations of a CNN and its equivalent BNN during training. Metrics start with *val_* in the graphics indicate *validation data* which we use during both of the training sessions. **(6.1a)** illustrates the loss and accuracy values during the training epochs of FP-LeNet, *o*LeNet. **(6.1b)** illustrates the loss and accuracy values during the training epochs of binarised LeNet, *b*LeNet.

the model to the data. If Figure 6.1b were seen solely without a name tag, one might easily assume that the data and the neural model are not a good fit. However, we know that indeed they are by looking at Figure 6.1a which proves the hustle of training a BNN which reflects to, and also sheds light onto the lower metrics results of *b*LeNet in Table 6.2 against *o*LeNet.

Table 6.2: The metric results of FP-CNN and BNN on test data.

Model	Accuracy (%)		Loss	Error Rate	Number of MACs		Model Size
	Top-1	Top-5			1-bit	32-bit	
*o*LeNet	99.17	99.99	0.03	0.0013	–	416,520	242.8 KiB
*b*LeNet	95.75	99.91	0.32	0.0075	298,920	117,600	9.27 KiB

The salient observation based on Figure 6.1 and Table 6.2 is the loss in the ANN metrics, and the gain in the energy efficiency metrics when an equivalent BNN of a FP-CNN is trained. On one hand, the accuracy of the model drops 3.45% and loss becomes $\approx 11x$ more as seen in Table 6.2 in the case of *b*LeNet. On the other hand, the network model shrinks $\approx 26x$, and $\approx 72\%$ of the MACs of the model becomes 1-bit operations in *b*LeNet. We do not showcase the inference time of the models in Table 6.2 because we obtain almost the same measurements from both of them on our machine. Although memory usage is not among our metrics in this research, during the experiments it was also observed that the memory usage of *b*LeNet during both training and inference is $\approx 1.5x$ higher than the memory usage of *o*LeNet on the same machine. Overall, we conclude that unless the highlights of BNNs, i.e., reduced MACs and a small model size, are the best fit to the edge device, considering its computational overhead, training a BNN from scratch would be up to user preference.

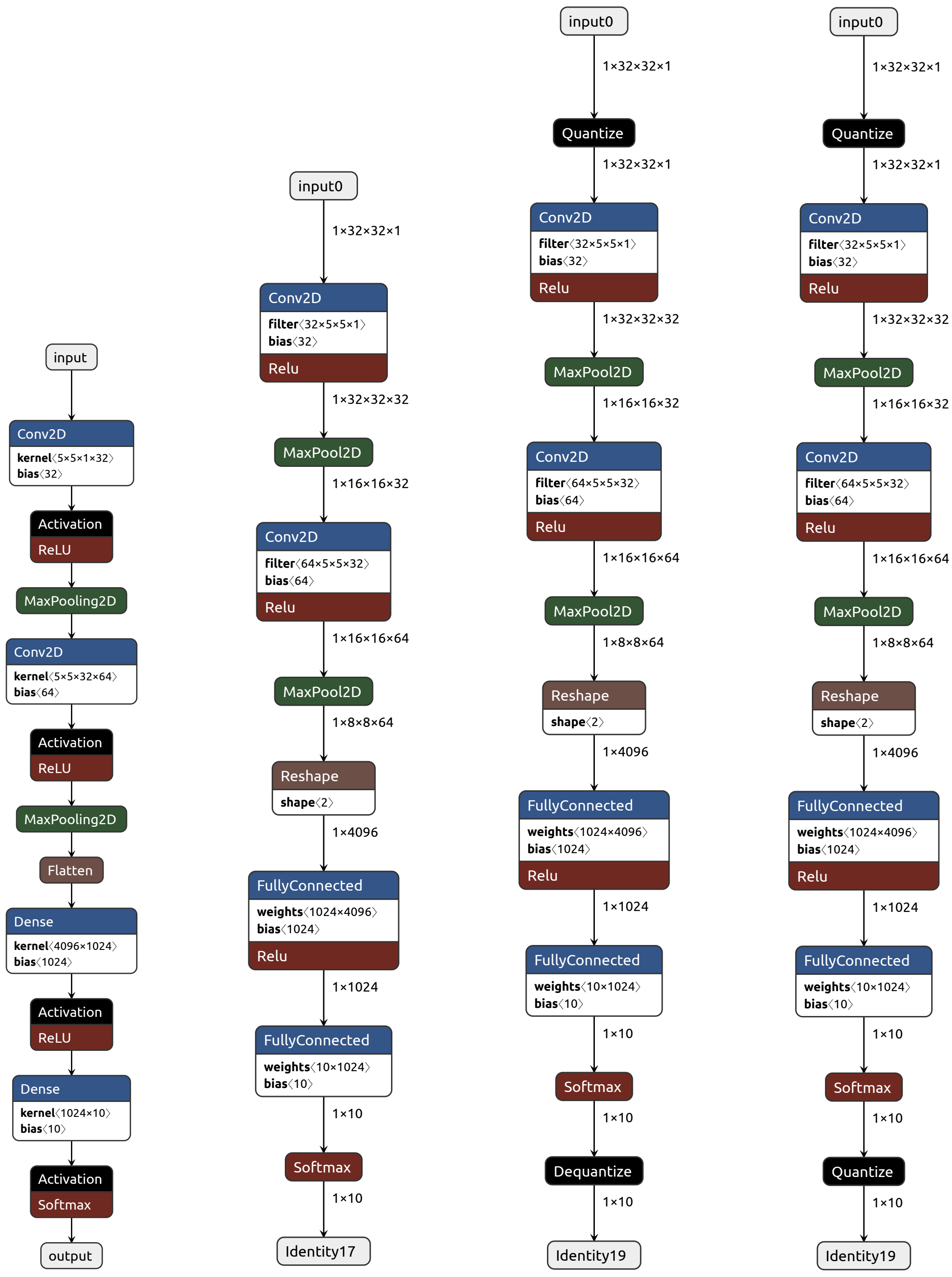
6.2.2 Converting the Existing Models

6.2.2.1 Quantisation of the Models

Binarisation and three degrees of quantisation of the pretrained models are experimented in this section. All the experiments are realised over pretrained, ready-to-use models, without any further training via conversion techniques. The degrees of quantisation concern the data precision. To elaborate, the original data type used in the building blocks of FP-CNNs is *float32*. Because we apply a *full integer quantisation*, the data types are quantised into either *int32* or *int8* depending on the quantisation degree, or in some quantisation degrees remain as they are, in type *float32*.

Quantisation experiments are reviewed in three parts within this section. We detail the changes in data types and in the architecture of the model when converted into a QNN in Part I. Thereafter, we review the performance of the models after quantisation in Part II and after binarisation in Part III. Throughout this section, the numerical levels following QNNs, *QNN-I-II-III*, refer to the quantisation degrees of the models which is defined in Chapter 5 - Section 5.5.2.

Architectural Aspects of the Quantisation We examine the changes that occur in the model after quantisation in terms of its architecture and data types in this part. We display a FP-CNN and three degrees of QNN counterparts in Figure 6.2 for that purpose. The overall schema in Figure 6.2 reflects the changes on a FP-CNN as the quantisation level increases with the degrees. The main CNN architecture chosen for this examination is *n*LeNet. *n*LeNet has the most compact architecture among all the architectures studied in this research. Because of its compactness, it would be more straightforward to investigate the effects of the quantisation process on it, ergo we decide on *n*LeNet.



(a) FP-CNN (b) QNN-I (c) QNN-II (d) QNN-III

Figure 6.2: The *n*LeNet trained on *image* data and its quantised forms by three different degrees.

*n*LeNet has an output containing either 10 neurons or 20 neurons depending on the classification task during our experiments. 10 neurons in case of image classification, and 20 neurons in case of sound classification. The *n*LeNet, which is the *image* classification part of the *m*LeNet architecture, is the FP-CNN leg of the QNN schema that displayed in Figure 6.2a. A reminder here is that, *n*LeNet and *m*LeNet were introduced in Chapter 5 - Section 5.3.2. Additionally, the architecture of *n*LeNet was elaborated in Section 5.6 - Table 5.7.

Figure 6.2b shows the 1st degree quantisation of the FP model, *n*LeNet. The architecture remains almost the same as before except the *Flatten* is turned into a *Reshape* layer, and the inner Activation layers, *ReLUs*, are tucked into their preceding layers by TF-Lite. Only the weights of the model are quantised into *int8* within this degree. However, as an exception to the situation, the weights of the first and last layers, which are *Conv2D* and *Softmax* respectively in the Figure, are kept in type *float32*. Moreover, all *bias* values in the model remain in type *float32*; the pooling layers are also realised over type *float32* within this degree. Overall, the new model, QNN-I, obtained via quantisation degree I, comprises *float32* and *int8* data types. Making data type alterations, i.e., using mixed data types, in the model to keep the accuracy as high as possible, ergo to salvage the accuracy, is indeed a common application[4] in quantisation. We will discuss more mixed types in quantised models in the proceeding paragraphs. As a note, we apply the 1st degree operation onto our pretrained models via the *default optimisation* flag in the TF-Lite.

Figure 6.2c shows the 2nd degree quantisation of the FP model, *n*LeNet. All the building blocks of the model, except input and output, are quantised within this degree. Such quantisation is realised into type *int8*. However, as an exception to that, bias values are quantised into type *int32*. Because the input and output remain in their original data type, *float32*, the "Quantize" and "Dequantize" processes occur after the input and before the output in Figure 6.2c. The new model, QNN-II, obtained via quantisation degree II, comprises *float32*, *int32*, and *int8* data types.

Figure 6.2d shows the 3rd degree quantisation of the FP model, *n*LeNet. All the building blocks of the model are quantised into type *int8* within this degree. However, the bias values in the model, once more, remain in type *int32*. Because the input and output are also quantised within this degree, we see only the "Quantize" process after and before them in the Figure. The new model, QNN-III, obtained via quantisation degree III, comprises *int8* and *int32* data types.

[4]This topic was also mentioned in Section 5.3.1.2 when introducing the BNNs into our experiments.

The performance results of the three degree quantisation operations will be discussed in the following part. Such results belonging to the architecture in Figure 6.2a can be found in Table 6.3 under the name of "*n*LeNet - *Image*".

Performance Aspects of the Quantisation Our initial intention was to experiment quantisation solely over *o*LeNet which was described in Section 5.3.1.1 and trained on MNIST data, and is computationally the smallest and lightest CNN in this research. However, the outcome we get from the aforementioned experiment did not deliver identifying results in terms of accuracy as we expected. Therefore, we also experiment with the *n*LeNet architecture with both *image* and *sound* data trained versions to get more concrete and decisive outcomes. We run three experiments in total and our experimental results are displayed in Table 6.3.

Table 6.3: Accuracy, model size, and comparative inference time of the three quantised models.

(a) Accuracy and model size.

Model	**Accuracy (%)**				**Model Size**			
	FP-CNN	QNNs			FP-CNN	QNNs		
		QNN-I	QNN-II	QNN-III		QNN-I	QNN-II	QNN-III
*o*LeNet	99.17	99.18	99.16	**99.20**	242.80 KiB	68.62 KiB	66.45 KiB	66.54 KiB
*n*LeNet - Image	91.57	**91.68**	91.16	91.17	16.24 MiB	4.07 MiB	4.07 MiB	4.07 MiB
*n*LeNet - Sound	**78.08**	77.92	76.52	72.79	16.28 MiB	4.08 MiB	4.08 MiB	4.08 MiB

(b) Inference time.

Model	Inference Time	
	FP-CNN	QNNs
*o*LeNet	15 x	x
*n*LeNet - Image	15 x	3 x
*n*LeNet - Sound	15 x	3 x

Before moving to review the results on the Table 6.3, there are two notes that we need to mention. Firstly, although the model sizes of QNNs seem the same for *n*LeNet models in the Table, these sizes differ in precision. Their differences are in byte levels. The real quantised size values of *image* and *sound* *n*LeNet are presented in 6.2 and 6.3 respectively.

$$4,268,288 \xrightarrow{-920} 4,267,368 \xrightarrow{+72} 4,267,440 \tag{6.2}$$

$$4,278,560 \xrightarrow{-912} 4,277,648 \xrightarrow{+72} 4,277,720 \tag{6.3}$$

Secondly, the FP-CNN model sizes of both *n*LeNet models differ from the measurements by Chou et al.. This goes to same for the merged model size as well. This is because while we use KiB and MiB throughout our research as unit of information measurement, they measure it via Kilobyte (KB) and Megabyte (MB)[5]. In this research, the reported sizes of the FP-CNN models are measured over their Keras saved .h5 files, and the sizes of their converted QNNs are measured over the saved TF-Lite models.

Up to 4*x* times smaller model size is obtained via quantisation as the results are displayed in Table 6.3a. Moreover, as seen in the quantised model versions of *o*LeNet in the Table, and in numerical representations in 6.2 and 6.3 of the quantised *n*LeNet models, the smallest model size is produced via QNNs degree II for all the models. However, considering that the size difference between quantisation degrees is in the byte levels, depending on the end device on which the model will run, size might not be the decisive feature at this stage.

Table 6.3b exhibits the power of QNNs over inference time of the models. Up to 15*x* times shorter inference time is obtained via quantisation of the models. Only one *inference time* report is observed in the Table 6.3b rather than three, owing to the fact that the metric resulted the same measurement for all the QNN degrees in this experiment.

Unlike in cases of model size and inference time metrics, we observe an inconsistency in the accuracy levels of the quantised models in Table 6.3a. To elaborate, *o*LeNet reaches the highest accuracy in QNN-III; *n*LeNet - *Image* reaches the highest accuracy in QNN-I; and *n*LeNet - *Sound* reaches the highest accuracy when in the FP form prior to quantisation. Under normal circumstances, a model is expected to achieve its highest accuracy when in the FP format. *n*LeNet - *Sound* in Table 6.3a is the representation of the expected outcome based on accuracy levels of a quantised model in this sense. The more a FP model is quantised, be it full integer quantisation or binarisation, the less its accuracy is expected to be. Nevertheless, we do not observe the reflection of such expectation on the QNNs accuracy outcome of *o*LeNet and *n*LeNet - *Image* in the Table.

Achieving a higher accuracy after quantisation is uncommon for FP-CNNs, yet it is still possible. We list the reasons below that may cause this situation.

[5]Such difference between the measurements is explained as $byte \xrightarrow{\times 1000} KB \xrightarrow{\times 1000} MB$ and $byte \xrightarrow{\times 1024} KiB \xrightarrow{\times 1024} MiB$.

1. The model might not be fully converged prior to quantisation.
2. Training data might be out of the optimal search space of the overall data.
3. Reducing bit inferences during quantisation might fit the model better and help the gradients to find their optimal point.

Because optimising the training process of ANNs is not within the scope of this thesis, we only examined the item 2 in the above list. All the tests throughout this thesis are done over test data; training and validation data are only used during training. However, when a quantised version of a model resulted with a higher accuracy than its original, FP, form, the other data performances also need to be reviewed. The training data accuracy of *o*LeNet is 99.71% and of *nLeNet - Image* is 99.15%. Both of the accuracy values are higher than the corresponding, test data accuracy, values in Table 6.3a. This proves the existence of a higher possible accuracy value depending on the data distribution when training with and testing on the data. Nonetheless, because our aim is not to put forward a SotA result via accuracy levels within this research, we do not apply complete search space operations over data to find a specific part that would result with the highest accuracy.

Binarisation vs. Quantisation Earlier, we demonstrated the training of a BNN in Section 6.2.1, discussed the results, and concluded that unless such a small model size, $\approx 26x$ smaller, is required to be used on the edge device, due to its computational overhead, training a BNN would be up to user preference. In this section, we experiment and observe the outcome of a direct conversion from FP-CNNs into BNNs. Table 6.4 displays the test results of *o*LeNet, a FP-CNN, along with its counterpart BNNs, one is trained and the other one is converted; all runs over MNIST test data.

Table 6.4: The metric outcomes of trained and converted BNNs along with a FP-CNN.

Model	Accuracy (%)		Loss	Error Rate
	Top-1	Top-5		
*o*LeNet	99.17	99.99	0.03	0.0013
*b*LeNet - *trained*	95.75	99.91	0.32	0.0075
*b*LeNet - *converted*	9.74	49.74	9.64	0.1759

There is a significant drop in the value of all the metric levels in Table 6.4. Because the outcomes that are shown in the Table are sufficient for reviewing, we do not display any inessential experiment results. Considering the unsatisfactory outcome of *b*LeNets in the above Table, we run the same experiment on other models that mentioned in this chapter as well. Nevertheless, we observe that the more complicated a model gets, the more metric levels suffer for both *trained* and *converted* BNNs during the experiment. For instance, when we run the same experiment via "*n*LeNet - Image", the *loss* value of the conversion climbed up to 44.4 where it was 9.64 for the *b*LeNet - *converted* in the Table. Overall, our experiment in this part proves that a rigorous formula is needed for such a direct conversion into BNNs. The results also emphasise the significance of the back-propagation algorithm, in other words the online training, for the BNNs.

* * *

We experiment and evaluate the two integer quantisation techniques up to this section: full integer quantisation involving the types *int8* and *int32*, and binarisation involving *int8*. We showcase the experiment results of the binarised and quantised versions of *o*LeNet in Tables 6.2 and 6.3 respectively. The results exhibit that QNNs, regardless of the degree, result with higher accuracy compared to the BNN versions of the same CNN, here *o*LeNet. The both quantisation techniques prove to produce more energy-efficient neural networks than their FP-CNN counterparts based on their energy-related metric results. Keeping that in mind and considering the accuracy drop in BNNs even if trained from scratch, we decide that a full integer QNNs is the best choice as long as the targeted edge device has enough memory and space. After all the examinations and reviews throughout this section, we choose to continue with QNN-I, considering its optimisation over model size, inference time, and accuracy values, for our final experiment in Section 6.2.3.

6.2.2.2 Spiking Neural Network Conversions

Metrics results of the two chosen FP-CNN to SNN conversion algorithms, detailed in Chapter 5 - Section 5.5.1, over *o*LeNet are evaluated and discussed in this section. Apart from the metrics that were introduced in Section 6.1, we also measure an SNN specific metric in this section, *spike count*. Spike count is seen as a energy-efficiency indicator in SNNs that the less the spikes, the higher it gets to preserve the overall energy in a system (Davidson and Furber, 2021). Thus, we also share the spike count, as *overall* and also as *average per neuron*, for the reviewed SNNs in this section. Note that, this metric will only be used in this section throughout this chapter where the subject of the comparison is SNNs. The metrics results are showcased in Table 6.5.

Table 6.5: Metrics based comparisons of FP-CNN to SNN conversion algorithms over *o*LeNet[6].

Model	Accuracy (%)	Loss	Inference Time	Spike Count Overall	Spike Count Average *(per Neuron)*
*o*LeNet	99.17	0.03	*x*	–	–
*s*LeNet *via rate-coding*	99.19	0.25	1.2 *x*	78,524	9.7
*s*LeNet *via temporal-coding*	99.17	0.03	3.4 *x*	12,520	1.2

We describe the architecture of *o*LeNet in Section 5.3.1.1. "*s*LeNet *via rate-coding*" in Table 6.5 is achieved by applying the research of Wu et al.. "*s*LeNet *via temporal-coding*" in the Table is achieved by applying the research of Stöckl and Maass. The two conversion techniques differ in nature as how they approach to converting from CNNs into SNNs. Temporal coding is considered to be more energy-efficient, and rate coding is known for its more broad and straightforward applicability in the literature; we explain such differences in Chapter 4 - Section 4.2.3. After experimenting both of the techniques on our representative CNN, *o*LeNet, we have made our decision based on the outcomes in Table 6.5.

Both conversion techniques prove themselves not to hurt accuracy levels based on the results in Table 6.5; the overall accuracy of *o*LeNet is preserved after FP-CNNs are converted into SNNs. Even an increase is observed in the accuracy of *rate-coded* *s*LeNet, *99.19%*, which is unusual as the highest accuracy is expected from the original FP-CNN. This topic was discussed in Section 6.2.2.1 since a quantised version of *o*LeNet resulted with *99.18%* accuracy. The reasoning of us back in the Section is relevant here as well. Please see the first two items in List 6.2.2.1 for more information.

The difference between the conversion techniques is observed over all the metrics except for accuracy in Table 6.5. On one hand, the loss becomes 8*x* higher, and *average spike count* gets ≈8 times more in case of *rate-coded* *s*LeNet. On the other hand, the inference time takes ≈3*x* longer in case of *temporal-coded* *s*LeNet. As long as it corresponds to *1*[7], the *average spike count per neuron* in such converted SNNs is considered as ideal for the converted models (Davidson and Furber, 2021) which coincides with the *temporal-coded* *s*LeNet in the Table. We should also note here that *MaxPool* layers are not supported by the technique of Wu et al. which constitute an important part in *n*LeNet, the subject of our neural reuse experiment. Due to all the aforementioned reasons in this paragraph, FP-CNNs to SNN conversion will be realised via the

[6]The loss values shown as 0.03 in the Table differ in precision. The FP values of loss in model *o*LeNet is 0.0294, and in *s*LeNet *via temporal-coding* is 0.0288.

[7]The real value was calculated as 1.72 in the correspondent paper (Davidson and Furber, 2021).

temporal-coding technique in the next section, Section 6.2.3.

Table 6.6: The metrics comparisons of CNNs and their correspondent (converted) SNNs.

Model	Accuracy (%)		Loss	Error Rate	Inference Time	Average Spike Count per Neuron
	Top-1	Top-5				
*n*LeNet - Image	91.57	99.87	0.67	0.0144	x	–
*s*LeNet - Image	89.64	99.84	0.46	0.0163	$3x$	1.2
*n*LeNet - Sound	78.08	94.69	1.65	0.0170	x	–
*s*LeNet - Sound	77.57	94.74	1.61	0.0173	$3x$	1.3

Before moving to our final experiment, the neural reuse, in which two *nLeNets* are firstly merged then converted, we display the spiking versions of such *n*LeNets in Table 6.6. *sLeNets* in the Table are realised via the temporal coding technique of Stöckl and Maass.

6.2.3 The Neural Reuse on Top of Everything

We use the merged LeNet-like model, *m*LeNet during our final neural reuse experiment. The both models in the merged model, that are addressed as *n*LeNet by architecture, were examined in Chapter 5 and experimented on during the previous sections in this Chapter. These experiments involved conversion into different types of ANNs and were grouped under two titles: QNNs in Section 6.2.2.1, and SNNs in Section 6.2.2.2 respectively.

Our final experiment comprises the conversion of *m*LeNet, a merged FP-CNN consisting of two CNNs, into firstly an SNN, then a QNN. The SNN is converted via temporal coding that were explained in Section 6.2.2.2, and the QNN corresponds to "degree-I" full integer quantisation in Section 6.2.2.1. The results of the final experiment is displayed in Table 6.7.

Before reviewing the results, we recap the models in Table 6.7 below.

- ***n*LeNet** is a FP pretrained CNN. *nLeNet - Image* is the image data trained version of it, and *nLeNet - Sound* is the sound data trained version of it.
- ***q*LeNet** is the *degree-I* quantised version of *n*LeNet.
- ***m*LeNet - Original** is obtained via merging *nLeNet - Image* and *nLeNet - Sound*. It is a FP-CNN.
- ***m*LeNet - Spiking** is an SNN converted over *mLeNet - Original* via temporal-coding.

Table 6.7: Neural reuse experiment results over FP, Spiking, and Quantised CNNs.

Model	Accuracy (%)		Model Size (MiB)	Inference Time
	Image	Sound		
*n*LeNet - Image	91.57	–	16.24	5 *x*
*n*LeNet - Sound	–	78.08	16.28	5 *x*
*q*LeNet - Image	91.68	–	4.07	*x*
*q*LeNet - Sound	–	77.92	4.08	*x*
*m*LeNet - **Original**	91.08	78.00	3.16	28 *x*
*m*LeNet - **Spiking**	89.27	67.45	3.16	33.5 *x*
*m*LeNet - **Quantised**	90.03	76.57	1.12	13.5 *x*

– ***m*LeNet - Quantised** is the *degree-I* quantised version of *mLeNet - Original*.

The metrics results of *n*LeNet and *q*LeNet models in Table 6.7 were also displayed in Table 6.3a. We display them here as well for comparison including the *m*LeNet models. In terms of accuracy results in Table 6.7, the values gradually drop from the top to the bottom of the Table. The highest accuracy is obtained via the *nLeNet*s and the lowest via the *spiking mLeNet*s with the difference between them being ≈2.6%.

As in case of accuracy, the model size gets smaller starting from the top to the bottom of the Table 6.7 which is desired for the model size metric. The most important part that we would like to highlight in the neural reuse experiment is the comparison of quantising and merging of neural networks. Integer quantisation of a singular CNN, *qLeNet*s in the Table, results with only a little accuracy drop (in some cases accuracy is preserved as it is), and up to a 4*x* model size compression as seen in the Table. This is our solution offer for energy-efficiency in case of singular CNNs.

When working with multiple neural networks at the same time, *merging* reveals promising outcomes: a smaller model size, and almost the same amount of accuracy drop compared to the singular QNNs. This is the reason that we start with the "merge" for energy-efficiency in case of multiple CNNs. Where *mLeNet*s come short compared to *qLeNet*s is the *inference time* metric. As illustrated in Figure 6.3 based on the results shown in Table 6.7, while the model size becomes 10.3*x*[8] smaller, the inference time increases 2.8[9] times when the *nLeNet*'s are merged and they become *mLeNet - Original*. Inference time is a vital metric in automated systems (Lenard

[8]Model size comparisons showcased in Figure 6.3 and 6.4 are made over the combined size of *nLeNet*s, i.e., CNN-I and CNN-II in the Figures, that is 32.52 MiB in total.

[9]Inference time of the two *nLeNet*s combined is taken as $5x + 5x = 10x$ based on Table 6.7. The inference time of the *mLeNet - Original* is 28*x* which makes its inference time 2.8 times slower compared to each of its parts, i.e., *nLeNet*s.

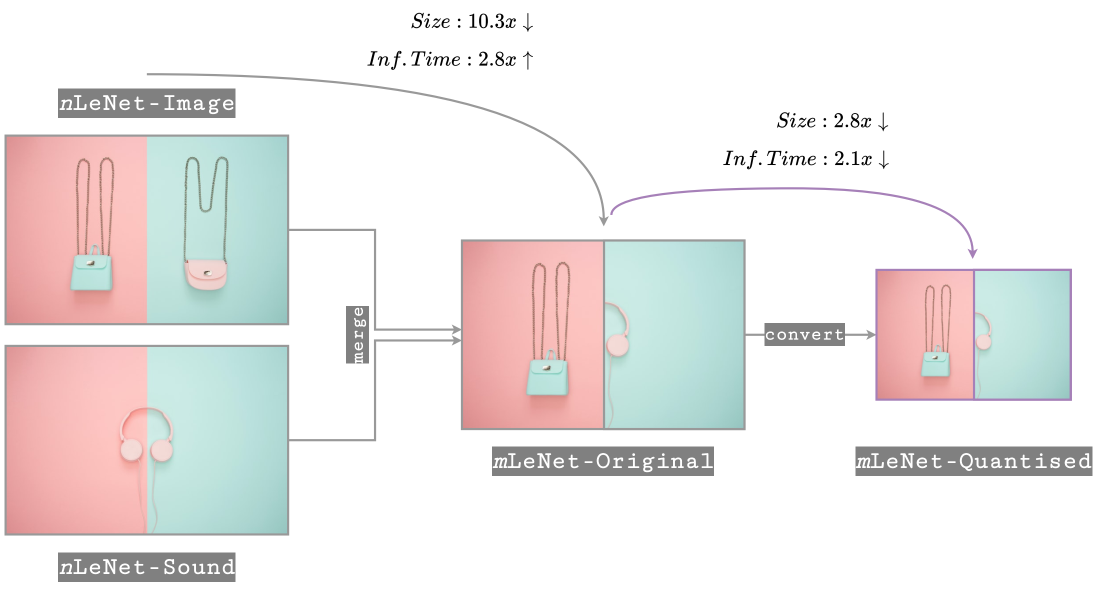


Figure 6.3: Final experimental results highlighting the difference in *model size* and *inference time* between the *n*LeNets, *m*LeNet-Original, and *m*LeNet-Quantised.

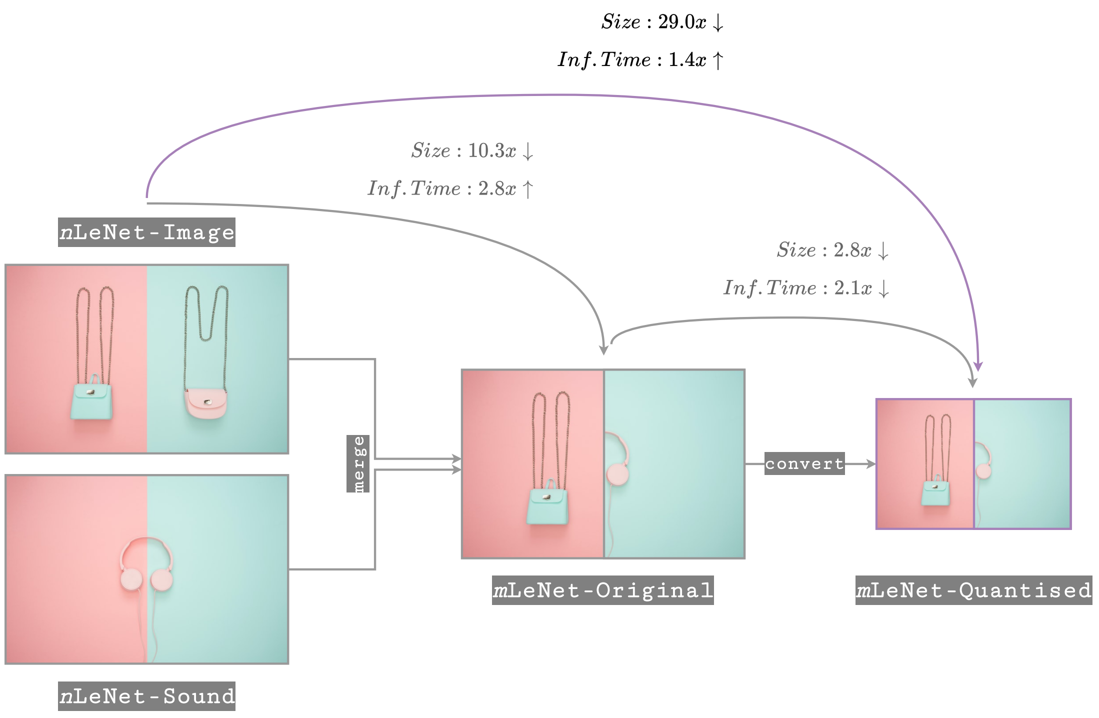

$Size : 29.0x \downarrow$
$Inf.Time : 1.4x \uparrow$
$Size : 10.3x \downarrow$
$Inf.Time : 2.8x \uparrow$
$Size : 2.8x \downarrow$
$Inf.Time : 2.1x \downarrow$
*n*LeNet-Image
merge
convert
*m*LeNet-Original
*m*LeNet-Quantised
*n*LeNet-Sound

Figure 6.4: Final experimental results highlighting the difference in *model size* and *inference time* between the *n*LeNets and *m*LeNet-Quantised.

et al., 2018), thus, to obtain further optimisation resulting with faster inference in our model, we quantise the *mLeNet*. With the quantisation of *mLeNet*, we achieve further decrease in the model size, 2.8*x*, and obtain 2.1 times faster inference as displayed in Figure 6.3. Overall, this experiment measures off and concludes our hypothesis that we recommend merging and quantising for multiple CNNs. With this way, from the two separate FP-CNNs to QNNs, our approach achieves 29 times smaller model size with the trade-off of 1.5 times slower inference time in total. We illustrate the final review of our experiment in Figure 6.4.

6.3 Summary and Discussion

We elaborate all the experiments within the thesis scope in an inductive manner in this chapter. Every section comprises experiments of a specific type of ANNs, and the following sections include the prior experiment results for the purpose of compare and eliminate on the way of reaching the most energy efficient of them all. Note that, our aim is to increase energy-efficiency in a multi-functional network, not in a singular network.

BNNs are a special type of QNNs as only consisting of weight values of -1 and $+1$. They were the topic of Sections 6.2.1 and 6.2.2.1 in this chapter. Because converting pretrained CNNs into BNNs is currently not a mature field, we firstly showcased metric outcomes of a from scratch trained BNN in Section 6.2.1, then experimented converting a pretrained CNN into a BNN in Section 6.2.2.1. Overall, we concluded that BNNs are essential for edge devices where the small model size along with 1-bit MACs has the uttermost priority.

Three degrees of integer quantisation is experimented in Section 6.2.2.1. Our experiments in the aforementioned Section involve quantisation of the network elements into the range of $[-127, 127]$. Quantisation of ANNs is a frequently utilised technique especially in the field of *TinyML* (Banbury et al., 2021; David et al., 2020) that getting more attention by year due to the increasing demand of energy efficient systems. We quantise the pretrained CNNs which delivers a more compact model, smaller in size, faster in inference with a little drop in the accuracy metric.

We experimented and discussed converting pretrained CNNs into SNNs in Section 6.2.2.2. Although SNNs are proved to be highly efficient on neuromorphic hardware, on traditional hardware, regardless of training them from scratch (Hazan et al., 2018) or applying the most efficient, lossless conversion techniques (Li and Furber, 2021), they are still in their infancy (Pfeiffer and Pfeil, 2018; Davidson and Furber, 2021). Because our research targets efficient computation on conventional hardware and edge devices, we did not run our experiments on neuromorphic hardware. Our experiments show that quantisation is more favourable compared to SNNs in both singular and merged networks.

Our experiments show that providing multi-functionality to CNNs comes with a performance cost as the inference time of a merged network is 5.6*x* slower compared to the singular inference time of its parts, whereas merging networks provide substantial decrease in model sizes. However, further steps are taken in this research to find a better trade-off between accuracy, model size, and inference time where we applied quantisation to the merged network. With this way, we further reduce the model size while increasing the performance since quantised calculations are faster compared to FP arithmetic. Our biggest contribution is that after we demonstrate the merging of two networks, we then use quantisation onto the merged network as a next step, which increases the performance while reducing the model size without significant loss to the accuracy.

Table 6.8: Training and conversion time comparisons of *o*LeNet.

Model		Time	
		Training	Conversion
FP-CNN	*o*LeNet	264 *x*	–
BNN	*b*LeNet	203 *x*	–
SNNs	*s*LeNet *via temporal-coding*	–	*x*
	*s*LeNet *via rate-coding*	–	33 *x*
QNNs	*q*LeNet - I	–	1.2 *x*
	*q*LeNet - II	–	1.3 *x*
	*q*LeNet - III	–	1.3 *x*

Finally, in Table 6.8, we display the training and conversion time comparisons of the CNN models used in this research. Because training is, unless TL is included, made once, and then the inference occurs throughout the use of the neural network, we do not use it as a metric in the previous sections. Nonetheless, we share this information in the discussion part of the experiments considering it may be of interest to whom wants to replicate the same experiments.

Chapter 7

Conclusion

The purpose -where I start- is the idea of use. It is not recycling, it is reuse.

ISSEY MIYAKE

We conclude by revisiting the research questions and objectives, besides discussing our contributions to the existent literature in Section 7.1; offering a final overview of the thesis in Section 7.2; and pointing to possible future directions of this research in Section 7.3.

7.1 Revisiting the Research Questions and Objectives

We revisit the research questions and objectives that were introduced in Chapter 1 - Section 1.3 in this section. The contributions of the thesis and the novelty statement took place in Section 1.5. We also mention them in this section and link the contributions to each answer below.

RQ1. What are the efficient ways to work with multiple CNNs on conventional hardware resources?

This research involves a multi-disciplinary literature review and inspirations from several fields. The variety in our literature review can be seen in Chapter 1 - Section 1.1, Chapter 2, and Chapter 3. Most apparently, this research is inspired by the *Neural Reuse* theory that we discuss at length in Chapter 3. Neural reuse theory investigates the functional structure of the brain, and suggests that the same neural paths are used and then reused in the brain to realise sometimes seemingly unrelated functions in order to preserve energy within. This is where we get our inspiration to apply onto CNNs. We concluded our literature survey by that both creating something new, $ANN\ analogy \rightarrow training\ from\ scratch$, and recycling, $ANN\ analogy \rightarrow TL$, takes effort; while reuse remains the best strategy for energy efficiency in, indeed, any domain in the world today. Keeping that in mind, we utilised pretrained CNNs in this work. Within our knowledge, this research is the first work to emulate the neural reuse theory by using pretrained traditional ANNs, and it is the first research as "neural reuse in the computational sciences" in this extent in the literature.

RQ2. How can we make multiple CNNs more energy efficient without losing their effectiveness?

By emulating the developmental route of neural reuse discussed in Chapter 3, in Chapter 4 we achieve multi-functionality by combining two pretrained CNNs. Firstly, we merge the two separate CNNs based on their similarities. The merge makes their overall size smaller as the similar parts are now combined and work as one body. In addition to this, we convert the merged network into different, more energy efficient sub-types of ANNs to fully emulate the neural reuse theory. Several experiments were run in this research by utilising two generations and four types of ANNs in order to achieve efficiency in multiple CNNs. 2^{nd} and 3^{rd} generation of ANNs; and traditional FP-CNNs, QNNs, BNNs, and SNNs were used during our experiments in Chapter 5. We compared and discussed the differences of such networks based on the results of the experiments in Chapter 6. To the best of our knowledge, this is the first work that made such extended comparisons between different ANNs types.

RQ3. How do we measure energy efficiency in comparison to others without requiring specialised devices?

In Chapter 6 we introduced and applied two sets of metrics to evaluate our experiments in order to address the research question. The main metric we used in every experiment was accuracy as obtaining "accurate" results for any ANN has the utmost value. However, because our ultimate aim is to increase efficiency of the neural models, we firstly assessed the performance/efficiency trade-off by making use of and value other metrics results. The metrics we used to measure energy efficiency is our second set of metrics as described in Section 6.1.2 in Chapter 6. Such metrics are MACs, FLOPs, model size, and time, i.e., training, conversion, and inference time. We do not report all the metric results for every experiment we run in this thesis. This is because not all metrics offer comparable results in every experiment. We also do not report the overall energy consumed by the models in *joule (J)* form within this research. The metrics we choose for our research is the result of our literature survey involving Green-AI that represents the energy efficient AI.

7.2 Summary of the Thesis

The increased usage, availability, and the promising results obtained via the synthetic datasets and benchmarks of ANNs resulted with *Jevons Paradox* (Giampietro and Mayumi, 2018) especially during the recent years (Strubell et al., 2020; Patterson et al., 2021). Moreover, necessitated by the increased mobility of our day, the ANNs count per device, e.g., mobile devices, robots, and autonomous cars, is also multiplied which also implies the multiplication in their computational cost. Considering the undergoing, we targeted to make multiple CNNs, one of the most widely utilised type of ANNs, more energy efficient in this thesis.

We utilised and compared three, that we see as fundamental, energy saving methods in this thesis: neural reuse, quantisation&binarisation, and SNN conversion from the FP-CNNs. Furthermore, we used two separate metrics set, seven metrics in total to evaluate our hypothesis as we aim to deliver our research as transparent as possible. Our first metrics set comprise conventional ML metrics, such as accuracy, loss, and error rate. Our second metrics set comprise metrics that indicate energy consumption of CNNs. Such metrics include MACs, FLOPs, model size, and inference time. In our research, we did not measure energy directly in *J* or *kilowatt/hour (kWh)* as such measurements are reliant to their location and the computational resources that are used regarding that. Instead, we reported numerical values that are constant within the model. Inference time, because it is also reliant to the computational resource on which it is obtained, is less common to report compared to the other aforementioned metrics. However, considering the importance of the “time” factor today, we wanted to report it based on *x times* values for the comparison purposes. Because not all the metrics contribute to differentiating between the models, all the metrics were not reported for all the experiments in the thesis.

Neural reuse is a theory aiming to shed light onto energy efficiency of the human brain (Anderson, 2010, 2015). It reflects such efficiency as reusing the built-in, innate, appropriate paths in the brain to realise new functionalities. In other words, neural reuse suggests that the brain, rather than building new paths, utilises the old ones, as much as possible, to preserve its energy. Inspired by the neural reuse theory, we made use of the pretrained CNNs and merged them (Chou et al., 2018) in this work. We obtained both a smaller model size, compared to the sizes of the two singular neural networks combined, and multi-functionality in the networks via the merge. Merging the CNNs resulted with $\approx 10x$ smaller model size against the total sizes of the separate models. Moreover, the highest accuracy loss of the merged model, compared to the original accuracy of the models, was 0.5%. However, the model was slow in inference: the total inference time of the merged model got increased $\approx$3 times. Therefore, we applied further techniques to lighten and accelerate the merged model.

Binarisation requires a from scratch training to work effectively. Otherwise, monumental losses in metrics rates occur as shown in Chapter 6 - Table 6.4. FP-CNN to SNN and QNN conversions allow us to use pretrained networks with a dispensable loss over the original metrics rates. Among the aforementioned techniques, we achieved the most efficient results via the QNN conversions which lowered the inference time of the model $\approx 2x$ along with providing further decrease in model size in the rate of $\approx 3x$. We discussed the results of our experiments at every step in Chapter 6. In conclusion, we showcased and elaborated the final results in Section 6.2.3 and 6.3 in Chapter 6. The end result we achieved is a trade-off between the size and inference time, that is $29x$ smaller model size with $1.4x$ higher inference time after the quantisation of the merged model. The thesis outputs "a case study". The overall technical product is a prototype, not a general framework with dynamical inputs.

7.3 Future Directions

The future directions of the thesis are grouped and discussed as follows:

Compile, optimise, and accelerate the ANNs. accelerating the runtime of ANNs is targeted by many since it represents a significant bottleneck for the future of AI. One way to accelerate ANNs is going through the compiler level by examining the operations that are underneath the ANNs. An initiative launched for this purpose is ApacheTVM[1], an open-source ML compiler framework (Chen et al., 2018a,b; Zheng et al., 2020). TVM *(Tensor Virtual Machine)* works end to end; and analyses, optimises, and deploys models into desired end machines, e.g.,

[1]`https://tvm.apache.org/`

CPU, GPU, by accelerating the tensor operations constituting ANNs. A recent publication by the ApacheTVM showcased 3.8*x* faster performance execution obtained on *Intel* CPU (Zheng et al., 2020) which was the mainframe we used in this research.

Approximate computing. Another way to accelerate ANNs is to lighten, ergo, accelerate the calculations underneath the ANNs. We applied quantisation for this very purpose to our work. There are also other flourishing opportunities to make the ANNs more energy efficient from this perspective. One promising approach is *approximate computing*. According to the current SotA in approximate computing that targets *matrix multiplication*, experiments from diverse domains exhibited up to 100*x* acceleration compared to exact computation, which is achieved via vector quantisation without using *multiply-adds* (Blalock and Guttag, 2021).

Energy measurements. We do not directly measure energy consumption in this research in *J* or *kWh* values. Many research attempted or directly measured the energy consumption via specialised devices, e.g., GPU and edge devices (Rodrigues et al., 2018; García-Martín et al., 2019; Wang et al., 2020; Sun et al., 2021). We report "energy-focused" metric results, such as MACs, FLOPs, model size, and inference time. Such metrics we measured can be used to measure energy consumption when this research is combined with an energy-specific hardware device. The three of the former aforementioned metrics report numerical, within their model, constant values. On the other hand, the latter among them, inference time is reported as the mean result of over 10 experiments. We observed that the difference between the runs were not significant. However, as future work, statistical data, such as *standard deviation* and *standard error of the mean (SEM)* will also be reported along with mean results to demonstrate the difference between the runs.

Working on more ANN types. CNNs were targeted in this research considering their being widely used. Transformer type networks emerged as a forthcoming source to solve NLP problems via ANNs in the recent years (Vaswani et al., 2017). They reached the highest accuracy values in this domain and proved themselves to stay in the area as being the future of the topic as well. CNNs were the remaining and only solution in the image domain until so far. However, recently emerged *vision transformers* currently hold the Top-1 accuracy in *ImageNet* dataset (Dai et al., 2021). Considering the high cost of transformer networks, discussed in Chapter 1 and displayed numerically in Table 1.1, more research including transformers and other costly ANNs is necessary in our field.

A collective *framework* approach. We prototyped our hypothesis with a case study and a working example. The next step for such a research would be generating a framework which will automatically merge appropriate ANNs partially or as a whole with the support of "accelerating" the network accompanied by the necessary techniques. The framework should also report necessary metrics such as accuracy, model compression size, and inference time comparisons.

More inspiration from nature. With the ongoing advances in the computational neuroscience, the more brain atlases get published, the more the chances that theories like *neural reuse* would emerge. Replicating the exact same spiking neural connections by examining such atlases[2,3,4] would not be far away under the light of the upcoming evidences (Scheffer et al., 2020).

[2]Mouse Visual Cortex: `https://www.microns-explorer.org/cortical-mm3`
[3]Fruitfly connectomics: `http://www.virtualflybrain.org/`
[4]Various animal connectomes: `https://neurodata.io/project/connectomes/`

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2016a). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv preprint arXiv:1603.04467*.

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016b). TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016*, pages 265–283.

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., Imam, N., Nakamura, Y., Datta, P., Nam, G. J., Taba, B., Beakes, M., Brezzo, B., Kuang, J. B., Manohar, R., Risk, W. P., Jackson, B., and Modha, D. S. (2015). TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10):1537–1557.

Alem, S., Perry, C. J., Zhu, X., Loukola, O. J., Ingraham, T., Søvik, E., and Chittka, L. (2016). Associative Mechanisms Allow for Social Learning and Cultural Transmission of String Pulling in an Insect. *PLoS Biology*, 14(10):e1002564.

Amir, A., Taba, B., Berg, D., Melano, T., Mckinstry, J., Di Nolfo, C., Nayak, T., Andreopoulos, A., Garreau, G., Mendoza, M., Kusnitz, J., Debole, M., Esser, S., Delbruck, T., Flickner, M., and Modha, D. (2017). A low power, fully event-based gesture recognition system. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:7388–7397.

Amunts, K., Ebell, C., Muller, J., Telefont, M., Knoll, A., and Lippert, T. (2016). The Human Brain Project: Creating a European Research Infrastructure to Decode the Human Brain. *Neuron*, 92(3):574–581.

Anantrasirichai, N. and Bull, D. (2022). Artificial intelligence in the creative industries: a review. *Artificial Intelligence Review*, 55(1):589–656.

Anderson, M. L. (2010). Neural reuse: A fundamental organizational principle of the brain. *Behavioral and Brain Sciences*, 33(4):245–266.

Anderson, M. L. (2015). Précis of after Phrenology: Neural Reuse and the Interactive Brain. *Behavioral and Brain Sciences*, 39(May).

Ardila, A., Bernal, B., and Rosselli, M. (2016). How Localized are Language Brain Areas? A Review of Brodmann Areas Involvement in Oral Language. *Archives of Clinical Neuropsychology*, 31(1):112–122.

Arena, P., Patane, L., and Strauss, R. (2013). The Insect Mushroom Bodies: a Paradigm of Neural Reuse. In *Artificial Life Conference Proceedings 13*, pages 765–772. MIT Press.

Attwell, D. and Laughlin, S. B. (2001). An energy budget for signaling in the grey matter of the brain. *Journal of Cerebral Blood Flow and Metabolism*, 21(10):1133–1145.

Aytar, Y., Vondrick, C., and Torralba, A. (2017). See, Hear, and Read: Deep Aligned Representations. *arXiv preprint arXiv:1706.00932*.

Banbury, C., Zhou, C., Fedorov, I., Navarro, R. M., Thakker, U., Gope, D., Reddi, V. J., Mattina, M., and Whatmough, P. N. (2021). MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers. *Proceedings of Machine Learning and Systems*, 3:517–532.

Basu, A., Acharya, J., Karnik, T., Liu, H., Li, H., Seo, J. S., and Song, C. (2018). Low-Power, Adaptive Neuromorphic Systems: Recent Progress and Future Directions. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(1):6–27.

Beaulieu-Laroche, L., Brown, N. J., Hansen, M., Toloza, E. H., Sharma, J., Williams, Z. M., Frosch, M. P., Cosgrove, G. R., Cash, S. S., and Harnett, M. T. (2021). Allometric rules for mammalian cortical layer 5 neuron biophysics. *Nature*, 600(7888):274–278.

Benson, L. V., Candadai, M., and Izquierdo, E. J. (2020). Neural reuse in multifunctional neural networks for control tasks. In *The 2020 Conference on Artificial Life*, pages 210–218. MIT Press - Journals.

Blackiston, D. J., Casey, E. S., and Weiss, M. R. (2008). Retention of memory through metamorphosis: Can a moth remember what it learned as a caterpillar? *PLoS ONE*, 3(3):e1736.

Blalock, D. and Guttag, J. (2021). Multiplying Matrices Without Multiplying. *arXiv preprint arXiv:2106.10860*.

Brandli, C., Berner, R., Yang, M., Liu, S. C., and Delbruck, T. (2014). A 240x180 130 dB 3 us latency global shutter spatiotemporal vision sensor. *IEEE Journal of Solid-State Circuits*, 49(10):2333–2341.

Brette, R. (2015). Philosophy of the spike: Rate-based vs. Spike-based theories of the brain. *Frontiers in Systems Neuroscience*, 9(November):151.

Byerly, A., Kalganova, T., and Dear, I. (2021). No routing needed between capsules. *Neurocomputing*, 463:545–553.

Candadai, M. and Izquierdo, E. (2018). Multifunctionality in embodied agents: Three levels of neural reuse. *Proceedings of the Conference of the Cognitive Science Society. Madison, Wisconsin.*, abs/1802.0:1–6.

Chen, J. M. (2021). Carbon neutrality: Toward a sustainable future. *The Innovation*, 2(3):1–2.

Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Cowan, M., Shen, H., Wang, L., Hu, Y., Ceze, L., Guestrin, C., and Krishnamurthy, A. (2018a). TVM: An automated end-to-end optimizing compiler for deep learning. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018*, pages 578–594. USENIX Association.

Chen, T., Zheng, L., Yan, E., Jiang, Z., Moreau, T., Ceze, L., Guestrin, C., and Krishnamurthy, A. (2018b). Learning to optimize tensor programs. *Advances in Neural Information Processing Systems*, 2018-Decem:3389–3400.

Chen, W., Wilson, J. T., Tyree, S., Weinberger, K. Q., and Chen, Y. (2015). Compressing neural networks with the hashing trick. In *32nd International Conference on Machine Learning, ICML 2015*, volume 3, pages 2275–2284. PMLR.

Chou, Y. M., Chan, Y. M., Lee, J. H., Chiu, C. Y., and Chen, C. S. (2018). Unifying and merging well-trained deep neural networks for inference stage. In *IJCAI International Joint Conference on Artificial Intelligence*, volume 2018-July, pages 2049–2056.

Cohen, N. and Denham, J. E. (2019). Whole animal modeling: piecing together nematode locomotion. *Current Opinion in Systems Biology*, 13:150–160.

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv preprint arXiv:1602.02830.*

Czekanski-Moir, J. E. and Rundell, R. J. (2020). Endless forms most stupid, icky, and small: The preponderance of noncharismatic invertebrates as integral to a biologically sound view of life. *Ecology and Evolution*, 10(23):12638–12649.

Dacrema, M. F., Cremonesi, P., and Jannach, D. (2019). Are we really making much progress? A worrying analysis of recent neural recommendation approaches. In *RecSys 2019 - 13th ACM Conference on Recommender Systems*, pages 101–109, New York, NY, USA. ACM.

Dai, Z., Liu, H., Le, Q. V., and Tan, M. (2021). CoAtNet: Marrying Convolution and Attention for All Data Sizes. *arXiv preprint arXiv:2106.04803.*

D'Amour, A., Heller, K., Moldovan, D., Adlam, B., Alipanahi, B., Beutel, A., Chen, C., Deaton, J., Eisenstein, J., Hoffman, M. D., Hormozdiari, F., Houlsby, N., Hou, S., Jerfel, G., Karthikesalingam, A., Lucic, M., Ma, Y., McLean, C., Mincu, D., Mitani, A., Montanari, A., Nado, Z., Natarajan, V., Nielson, C., Osborne, T. F., Raman, R., Ramasamy, K., Sayres, R., Schrouff, J., Seneviratne, M., Sequeira, S., Suresh, H., Veitch, V., Vladymyrov, M., Wang, X., Webster, K., Yadlowsky, S., Yun, T., Zhai, X., and Sculley, D. (2020). Underspecification Presents Challenges for Credibility in Modern Machine Learning. *arXiv preprint arXiv:2011.03395.*

David, R., Duke, J., Jain, A., Reddi, V. J., Jeffries, N., Li, J., Kreeger, N., Nappier, I., Natraj, M., Regev, S., Rhodes, R., Wang, T., and Warden, P. (2020). TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems. *arXiv preprint arXiv:2010.08678*.

Davidson, S. and Furber, S. B. (2021). Comparison of Artificial and Spiking Neural Networks on Digital Hardware. *Frontiers in Neuroscience*, 15.

Davies, M., Wild, A., Orchard, G., Sandamirskaya, Y., Guerra, G. A., Joshi, P., Plank, P., and Risbud, S. R. (2021). Advancing Neuromorphic Computing with Loihi: A Survey of Results and Outlook. *Proceedings of the IEEE*, 109(5):911–934.

Davies, S. (2012). *Learning in Spiking Neural Networks*. PhD thesis, University of Manchester.

Davison, A. P., Brüderle, D., Eppler, J., Kremkow, J., Muller, E., Pecevski, D., Perrinet, L., and Yger, P. (2009). PyNN: A common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, 2(JAN):11.

Dayan, P. and Abbott, L. F. (2001). *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. Computational Neuroscience Series.

Estebon, M. D. (1997). Perceptrons : An Associative Learning Network. Technical report, Virginia Polytechnic Institute and State University.

Ferris, C. F., Cai, X., Qiao, J., Switzer, B., Baun, J., Morrison, T., Iriah, S., Madularu, D., Sinkevicius, K. W., and Kulkarni, P. (2019). Life without a brain: Neuroradiological and behavioral evidence of neuroplasticity necessary to sustain brain function in the face of severe hydrocephalus. *Scientific Reports*, 9(1).

Flynn, A., Tsachouridis, V. A., and Amann, A. (2021). Multifunctionality in a reservoir computer. *Chaos*, 31(1):013125.

Friston, K. (2010). The free-energy principle: A unified brain theory? *Nature Reviews Neuroscience*, 11(2):127–138.

Fukushima, K. (1975). Cognitron: A self-organizing multilayered neural network. *Biological Cybernetics*, 20(3-4):121–136.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202.

Fukushima, K. and Miyake, S. (1982). Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*, 15(6):455–469.

Furber, S. B., Galluppi, F., Temple, S., and Plana, L. A. (2014). The SpiNNaker project. *Proceedings of the IEEE*, 102(5):652–665.

Fürnkranz, J., Chan, P. K., Craw, S., Sammut, C., Uther, W., Ratnaparkhi, A., Jin, X., Han, J., Yang, Y., Morik, K., Dorigo, M., Birattari, M., Stützle, T., Brazdil, P., Vilalta, R., Giraud-Carrier, C., Soares, C., Rissanen, J., Baxter, R. A., Bruha, I., Baxter, R. A., Webb, G. I., Torgo, L., Banerjee, A., Shan, H., Ray, S., Tadepalli, P., Shoham, Y., Powers, R., Shoham, Y., Powers, R., Webb, G. I., Ray, S., Scott, S., Blockeel, H., and De Raedt, L. (2011). Model Evaluation. In *Encyclopedia of Machine Learning*, pages 683–683. Springer, Boston, MA.

Garcia, G. P., Camilleri, P., Liu, Q., and Furber, S. (2017). PyDVS: An extensible, real-time Dynamic Vision Sensor emulator using off-the-shelf hardware. In *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016*. Institute of Electrical and Electronics Engineers Inc.

Garcia, J. O., Ashourvan, A., Thurman, S. M., Srinivasan, R., Bassett, D. S., and Vettel, J. M. (2020). Reconfigurations within resonating communities of brain regions following tms reveal different scales of processing. *Network Neuroscience*, 4(3):611–636.

García-Martín, E., Rodrigues, C. F., Riley, G., and Grahn, H. (2019). Estimation of energy consumption in machine learning. *Journal of Parallel and Distributed Computing*, 134:75–88.

Geiger, L. and Team, P. (2020). Larq: An Open-Source Library for Training Binarized Neural Networks. *Journal of Open Source Software*, 5(45):1746.

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., and Keutzer, K. (2022). A Survey of Quantization Methods for Efficient Neural Network Inference. In *Low-Power Computer Vision*, pages 291–326. Chapman and Hall/CRC.

Ghosh-Dastidar, S. and Adeli, H. (2009). Third generation neural networks: Spiking neural networks. In *Advances in Intelligent and Soft Computing*, volume 61 AISC, pages 167–178. Springer, Berlin, Heidelberg.

Giampietro, M. and Mayumi, K. (2018). Unraveling the complexity of the Jevons Paradox: The link between innovation, efficiency, and sustainability. *Frontiers in Energy Research*, 6(APR):26.

Gong, Y., Liu, L., Yang, M., and Bourdev, L. (2014). Compressing Deep Convolutional Networks using Vector Quantization. *arXiv preprint arXiv:1412.6115*.

Grill-Spector, K. and Malach, R. (2004). the Human Visual Cortex. *Annual Review of Neuroscience*, 27(1):649–677.

Grosu, R. (2020). ResNets, NeuralODEs and CT-RNNs are Particular Neural Regulatory Networks. Technical report, TU Wien.

Guo, W., Fouda, M. E., Eltawil, A. M., and Salama, K. N. (2021). Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems. *Frontiers in Neuroscience*, 15:212.

Han, S., Mao, H., and Dally, W. J. (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*.

Han, S., Pool, J., Tran, J., and Dally, W. J. (2015). Learning both weights and connections for efficient neural networks. *Advances in Neural Information Processing Systems*, 2015-Janua:1135–1143.

Hao, Y., Campana, B., and Keogh, E. (2012). Monitoring and mining insect sounds in visual space. In *Proceedings of the 12th SIAM International Conference on Data Mining, SDM 2012*, pages 792–803.

Hasani, R. M. (2020). *Interpretable Recurrent Neural Networks in Continuous-time Control Environments*. PhD thesis, TU Wien.

Hay, J. C., Lynch, B. E., and Smith, D. R. (1960). Mark I Perceptron Operators' Manual. Technical report, Cornell Aeronautical Laboratory, Inc., Buffalo 21, NY, USA.

Hazan, H., Saunders, D. J., Khan, H., Patel, D., Sanghavi, D. T., Siegelmann, H. T., and Kozma, R. (2018). BindsNET: A machine learning-oriented spiking neural networks library in python. *Frontiers in Neuroinformatics*, 12:89.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-Decem, pages 770–778.

He, X., Wang, X., Zhou, Z., Wu, J., Yang, Z., and Thiele, L. (2021). On-Device Deep Multi-Task Inference via Multi-Task Zipping. *IEEE Transactions on Mobile Computing*.

He, X., Zhou, Z., and Thiele, L. (2018). Multi-Task Zipping via Layer-wise Neuron Sharing. In *32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*, pages 2–8, Montréal, Canada.

Heisenberg, M. (2003). Mushroom body memoir: From maps to models. *Nature Reviews Neuroscience*, 4(4):266–275.

Hodgkin, A. L. and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544.

Horowitz, M. (2014). Computing's Energy Problem: (and what we can do about it).

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861*.

Hu, J., Shen, L., Albanie, S., Sun, G., and Wu, E. (2020). Squeeze-and-Excitation Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(8):2011–2023.

Hu, X., Chang, M. C., Chen, Y., Sridhar, R., Hu, Z., Xue, Y., Wu, Z., Pi, P., Shen, J., Tan, J., Lian, X., Liu, J., Wang, Z., Liu, C. H., Han, Y. S., Sung, Y. Y., Lee, Y., Wu, K. C., Guo, W. X., Lee, R., Liang, S., Wang, Z., Ding, G., Zhang, G., Xi, T., Chen, Y., Cai, H., Zhu, L., Zhang, Z., Han, S., Jeong, S., Kwon, Y., Wang, T., and Pan, J. (2021). The 2020 Low-Power Computer Vision Challenge. In *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems, AICAS 2021*, pages 1–4. Institute of Electrical and Electronics Engineers Inc.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). Binarized neural networks. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

Iandola, F., Moskewicz, M., Karayev, S., Girshick, R., Darrell, T., and Keutzer, K. (2014). DenseNet: Implementing Efficient ConvNet Descriptor Pyramids. *arXiv preprint arXiv:1404.1869*.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡0.5MB model size. *arXiv preprint arXiv:1602.07360*.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *32nd International Conference on Machine Learning, ICML 2015*, volume 1, pages 448–456.

Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14(6):1569–1572.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. (2018). Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2704–2713.

Jones, M. (2020). Numerals and neural reuse. *Synthese*, 197(9):3657–3681.

Joubert, A., Belhadj, B., Temam, O., and Héliot, R. (2012). Hardware spiking neurons design: Analog or digital? In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–5. IEEE.

Kaiser, L., Gomez, A. N., Shazeer, N., Vaswani, A., Parmar, N., Jones, L., and Uszkoreit, J. (2017). One Model To Learn Them All. *arXiv preprint arXiv:1706.05137*.

Karaiskou, A., Swalla, B. J., Sasakura, Y., and Chambon, J. P. (2015). Metamorphosis in solitary ascidians. *Genesis*, 53(1):34–47.

Kennedy, J., Blunden, J., Alvar-Beltrán, J., and Kappelle, M. (2021). State of the Global Climate 2020. Technical report, World Meteorological Organization.

Kim, M. and Smaragdis, P. (2016). Bitwise Neural Networks. *arXiv preprint arXiv:1601.06071*.

Kopuklu, O., Babaee, M., Hormann, S., and Rigoll, G. (2019). Convolutional Neural Networks with Layer Reuse. In *Proceedings - International Conference on Image Processing, ICIP*, volume 2019-Septe, pages 345–349, Taipei, Taiwan.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 25:1097–1105.

Larson, S. D., Gleeson, P., and Brown, A. E. (2018). Connectome to behaviour: Modeling Caenorhabditis elegans at cellular resolution. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 373(1758):8–10.

Lecun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1990). Handwritten Digit Recognition with a Back-Propagation Network. In Touretzky, D., editor, *Advances in Neural Information Processing Systems 2 (NIPS*89)*, pages 396–404, Denver, CO. Morgan Kaufmann.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323.

LeCun, Y., Cortes, C., and Burges, C. J. (2010a). MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2.

LeCun, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., Denker, J., Drucker, H., Guyon, I., Muller, U., Sackinger, E., Simard, P., and Vapnik, V. (1995). Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks*, volume 60, pages 53–60.

LeCun, Y., Kavukcuoglu, K., and Farabet, C. (2010b). Convolutional networks and applications in vision. In *ISCAS 2010 - 2010 IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems*, pages 253–256.

Lenard, J., Welsh, R., and Danton, R. (2018). Time-to-collision analysis of pedestrian and pedal-cycle accidents for the development of autonomous emergency braking systems. *Accident Analysis and Prevention*, 115:128–136.

Li, C. and Furber, S. (2021). Towards Biologically-Plausible Neuron Models and Firing Rates in High-Performance Deep Spiking Neural Networks. In *International Conference on Neuromorphic Systems 2021 (ICONS 2021), July 27–29, 2021*, page 7, Knoxville, TN, USA. Association for Computing Machinery.

Li, F., Zhang, B., and Liu, B. (2016). Ternary Weight Networks. *arXiv preprint arXiv:1605.04711*.

Lin, X., Zhao, C., and Pan, W. (2017). Towards accurate binary convolutional neural network. *Advances in Neural Information Processing Systems*, 2017-Decem:345–353.

Linnainmaa, S. (1970). The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. *Master's Thesis, Univ. Helsinki*.

Lipton, Z. C. and Steinhardt, J. (2019). Research for practice: Troubling trends in machine-learning scholarship. *Communications of the ACM*, 62(6):45–53.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L. J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. (2018). Progressive Neural Architecture Search. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11205 LNCS, pages 19–35.

Liu, Q., Pineda-García, G., Stromatias, E., Serrano-Gotarredona, T., and Furber, S. B. (2016). Benchmarking spike-based visual recognition: A dataset and evaluation. *Frontiers in Neuroscience*, 10(NOV):496.

Lizbinski, K. M. and Jeanne, J. M. (2020). Connectomics: Bringing Fly Neural Circuits into Focus. *Current Biology*, 30(16):R944–R947.

Ma, N., Zhang, X., Zheng, H. T., and Sun, J. (2018). Shufflenet V2: Practical guidelines for efficient cnn architecture design. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11218 LNCS, pages 122–138.

Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671.

Marcus, G. (2018). Deep Learning: A Critical Appraisal. *arXiv preprint arXiv:1801.00631*.

Markram, H., Meier, K., Lippert, T., Grillner, S., Frackowiak, R., Dehaene, S., Knoll, A., Sompolinsky, H., Verstreken, K., DeFelipe, J., Grant, S., Changeux, J. P., and Sariam, A. (2011). Introducing the Human Brain Project. In *Procedia Computer Science*, volume 7, pages 39–42. Elsevier.

Mayr, C., Höppner, S., and Furber, S. (2019). SpiNNaker 2: A 10 million core processor system for brain simulation and machine learning. In *Concurrent Systems Engineering Series*, volume 70, pages 277–280.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

Milde, M. B., Blum, H., Dietmüller, A., Sumislawska, D., Conradt, J., Indiveri, G., and Sandamirskaya, Y. (2017). Obstacle avoidance and target acquisition for robot navigation using a mixed signal analog/digital neuromorphic processing system. *Frontiers in Neurorobotics*, 11(JUL):28.

Minsky, M. and Papert, S. A. (1969). *Perceptrons*. MIT Press, Cambridge, MA.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill New York.

Ning, L. and Shen, X. (2019). Deep reuse: Streamline CNN inference on the fly via coarse-grained computation reuse. In *Proceedings of the International Conference on Supercomputing*, pages 438–448, New York, New York, USA. ACM Press.

Niven, J. E. and Chittka, L. (2010). Reuse of identified neurons in multiple neural circuits. *Behavioral and Brain Sciences*, 33(4):285.

Oh, K. S. and Jung, K. (2004). GPU implementation of neural networks. *Pattern Recognition*, 37(6):1311–1314.

Patanè, L., Strauss, R., and Paolo, A. (2018). *Nonlinear Circuits and Systems for Neuro-Inspired Robot Control*. Springer International Publishing, 1 edition.

Patterson, D., Gonzalez, J., Le, Q., Liang, C., Munguia, L.-M., Rothchild, D., So, D., Texier, M., and Dean, J. (2021). Carbon Emissions and Large Neural Network Training. *arXiv preprint arXiv:2104.10350*.

Peng, F. and Chittka, L. (2017). A Simple Computational Model of the Bee Mushroom Body Can Explain Seemingly Complex Forms of Olfactory Learning and Memory. *Current Biology*, 27(2):224–230.

Pfeiffer, M. and Pfeil, T. (2018). Deep Learning With Spiking Neurons: Opportunities and Challenges. *Frontiers in Neuroscience*, 12:774.

Phillips, S. F. and Pylkkänen, L. (2021). Composition within and between languages in the bilingual mind: Meg evidence from korean/english bilinguals. *eNeuro*, 8(6).

Pulido, C. and Ryan, T. A. (2021). Synaptic vesicle pools are a major hidden resting metabolic burden of nerve terminals. *Science Advances*, 7(49):9027.

Pulvermüller, F. (2018). Neural reuse of action perception circuits for language, concepts and communication. *Progress in Neurobiology*, 160:1–44.

Rajalingham, R., Kar, K., Sanghavi, S., Dehaene, S., and DiCarlo, J. J. (2020). The inferior temporal cortex is a potential cortical precursor of orthographic processing in untrained monkeys. *Nature Communications*, 11(1):1–13.

Rakowski, F. and Karbowski, J. (2017). Optimal synaptic signaling connectome for locomotory behavior in Caenorhabditis elegans: Design minimizing energy cost. *PLoS Computational Biology*, 13(11):e1005834.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. (2016). XNOR-net: Imagenet classification using binary convolutional neural networks. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9908 LNCS, pages 525–542. Springer.

Rhodes, O., Peres, L., Rowley, A. G., Gait, A., Plana, L. A., Brenninkmeijer, C., and Furber, S. B. (2020). Real-time cortical simulation on neuromorphic hardware. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2164).

Rodrigues, C. F., Riley, G., and Luján, M. (2018). SyNERGY: An energy measurement and prediction framework for Convolutional Neural Networks on Jetson TX1. In

Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), pages 375–382.

Romero, A., Ballas, N., Kahou, S. E., Chassang, A., Gatta, C., and Bengio, Y. (2015). FitNets: Hints for thin deep nets. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

Ross, A. I., Schenk, T., Billino, J., Macleod, M. J., and Hesse, C. (2018). Avoiding unseen obstacles: Subcortical vision is not sufficient to maintain normal obstacle avoidance behaviour during reaching. *Cortex*, 98:177–193.

Rössler, W. (2019). Neuroplasticity in desert ants (Hymeno ptera: Formicidae) – importance for the ontogeny of navigation. *Myrmecological News*, 29:1–20.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning Internal Representations by Error Propagation. Technical report, University of California, Institute for Cognitive Science, San Diego.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

Ryan, J. F. and Chiodin, M. (2015). Where is my mind? How sponges and placozoans may have lost neural cell types. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 370(1684).

Santos, M. S., Ludermir, T. B., Santos, F. L., de Melo, C. P., and Gomes, J. E. (1998). Artificial nose and data analysis using Multi Layer Perceptron. In *Proceedings of the International Conference on Data Mining*, volume 19, pages 251–263.

Sarma, G. P., Lee, C. W., Portegys, T., Ghayoomie, V., Jacobs, T., Alicea, B., Cantarelli, M., Currie, M., Gerkin, R. C., Gingell, S., Gleeson, P., Gordon, R., Hasani, R. M., Idili, G., Khayrulin, S., Lung, D., Palyanov, A., Watts, M., and Larson, S. D.

(2018). OpenWorm: Overview and recent advances in integrative biological simulation of Caenorhabditis elegans. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 373(1758).

Scheffer, L. K., Xu, C. S., Januszewski, M., Lu, Z., Takemura, S. Y., Hayworth, K. J., Huang, G. B., Shinomiya, K., Maitin-Shepard, J., Berg, S., Clements, J., Hubbard, P. M., Katz, W. T., Umayam, L., Zhao, T., Ackerman, D., Blakely, T., Bogovic, J., Dolafi, T., Kainmueller, D., Kawase, T., Khairy, K. A., Leavitt, L., Li, P. H., Lindsey, L., Neubarth, N., Olbris, D. J., Otsuna, H., Trautman, E. T., Ito, M., Bates, A. S., Goldammer, J., Wolff, T., Svirskas, R., Schlegel, P., Neace, E. R., Knecht, C. J., Alvarado, C. X., Bailey, D. A., Ballinger, S., Borycz, J. A., Canino, B. S., Cheatham, N., Cook, M., Dreher, M., Duclos, O., Eubanks, B., Fairbanks, K., Finley, S., Forknall, N., Francis, A., Hopkins, G. P., Joyce, E. M., Kim, S., Kirk, N. A., Kovalyak, J., Lauchie, S. A., Lohff, A., Maldonado, C., Manley, E. A., McLin, S., Mooney, C., Ndama, M., Ogundeyi, O., Okeoma, N., Ordish, C., Padilla, N., Patrick, C., Paterson, T., Phillips, E. E., Phillips, E. M., Rampally, N., Ribeiro, C., Robertson, M. K., Rymer, J. T., Ryan, S. M., Sammons, M., Scott, A. K., Scott, A. L., Shinomiya, A., Smith, C., Smith, K., Smith, N. L., Sobeski, M. A., Suleiman, A., Swift, J., Takemura, S., Talebi, I., Tarnogorska, D., Tenshaw, E., Tokhi, T., Walsh, J. J., Yang, T., Horne, J. A., Li, F., Parekh, R., Rivlin, P. K., Jayaraman, V., Costa, M., Jefferis, G. S., Ito, K., Saalfeld, S., George, R., Meinertzhagen, I. A., Rubin, G. M., Hess, H. F., Jain, V., and Plaza, S. M. (2020). A connectome and analysis of the adult drosophila central brain. *eLife*, 9:1–74.

Schemmel, J., Billaudelle, S., Dauer, P., and Weis, J. (2022). Accelerated Analog Neuromorphic Computing. In *Analog Circuits for Machine Learning, Current/-Voltage/Temperature Sensors, and High-speed Communication*, chapter Accelerate, pages 83–102. Springer.

Schemmel, J., Brüderle, D., Grübl, A., Hock, M., Meier, K., and Millner, S. (2010). A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1947–1950. IEEE.

Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. (2020). Green AI. *Communications of the ACM*, 63(12):54–63.

Sejnowski, T. J. and Rosenberg, C. R. (1986). NETtalk: A parallel network that learns to read aloud. Technical Report JHU/EECS-86/01, The Johns Hopkins University.

Setzler, M. and Izquierdo, E. J. (2017). Adaptability and Neural Reuse in Minimally Cognitive Agents. *Proceedings of the 39th Annual Conference of the Cognitive Science Society. London, UK: Cognitive Science Society*, pages 3119–3124.

Shankar, S., Halpern, Y., Breck, E., Atwood, J., Wilson, J., and Sculley, D. (2017). No Classification without Representation: Assessing Geodiversity Issues in Open Data Sets for the Developing World. *arXiv preprint arXiv:1711.08536.*

Siegmund, J., Peitek, N., Brechmann, A., Parnin, C., and Apel, S. (2020). Studying programming in the neuroage. *Communications of the ACM*, 63(6):30–34.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR.

Smith-Ferguson, J. and Beekman, M. (2020). Who needs a brain? Slime moulds, behavioural ecology and minimal cognition. *Adaptive Behavior*, 28(6):465–478.

Sodhani, S., Jaiswal, M. S., Baker, L., Sinha, K., Shneider, C., Henderson, P., Lehman, J., and Lowe, R. (2020). Ideas for Improving the Field of Machine Learning: Summarizing Discussion from the NeurIPS 2019 Retrospectives Workshop. *arXiv preprint arXiv:2007.10546.*

Standley, T., Zamir, A., Chen, D., Guibas, L., Malik, J., and Savarese, S. (2020). Which tasks should be learned together in multi-task learning? In *37th International Conference on Machine Learning, ICML 2020*, volume PartF16814, pages 9057–9069.

Steinkraus, D., Buck, I., and Simard, P. Y. (2005). Using GPUs for machine learning algorithms. In *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, volume 2005, pages 1115–1120.

Sterling, P. and Laughlin, S. (2015). *Principles of Neural Design.* MIT Press.

Stöckl, C. and Maass, W. (2021). Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes. *Nature Machine Intelligence*, 3(3):230–238.

Striemer, C. L., Chapman, C. S., and Goodale, M. A. (2009). ”Real-time” obstacle avoidance in the absence of primary visual cortex. *Proceedings of the National Academy of Sciences*, 106(37):15996–16001.

Strotzer, M. (2009). One century of brain mapping using Brodmann areas. *Clinical Neuroradiology*, 19(3):179–186.

Strubell, E., Ganesh, A., and McCallum, A. (2020). Energy and policy considerations for modern deep learning research. In *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence*, volume 34, pages 13693–13696.

Sun, C., Shrivastava, A., Singh, S., and Gupta, A. (2017). Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2017-Octob, pages 843–852. Institute of Electrical and Electronics Engineers Inc.

Sun, Y., Ou, Z., Chen, J., Qi, X., Guo, Y., Cai, S., and Yan, X. (2021). Evaluating Performance, Power and Energy of Deep Neural Networks on CPUs and GPUs. *Communications in Computer and Information Science*, 1494 CCIS:196–221.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 07-12-June, pages 1–9. IEEE Computer Society.

Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., and Liu, C. (2018). A survey on deep transfer learning. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11141 LNCS, pages 270–279.

Tanaka, G., Yamane, T., Héroux, J. B., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D., and Hirose, A. (2019). Recent advances in physical reservoir computing: A review. *Neural Networks*, 115:100–123.

Tanveer, M. S., Khan, M. U. K., and Kyung, C. M. (2020). Fine-tuning DARTS for image classification. In *Proceedings - International Conference on Pattern Recognition*, pages 4789–4796, Milan, Italy. IEEE.

Tedjakumala, S. R. and Giurfa, M. (2013). Rules and mechanisms of punishment learning in honey bees: The aversive conditioning of the sting extension response. *Journal of Experimental Biology*, 216(16):2985–2997.

Umuroglu, Y., Fraser, N. J., Gambardella, G., Blott, M., Leong, P., Jahre, M., and Vissers, K. (2017). FINN: A framework for fast, scalable binarized neural network inference. In *FPGA 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 65–74.

Vanhoucke, V., Senior, A., and Mao, M. (2011). Improving the speed of neural networks on CPUs. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, pages 1–8.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 2017-Decem, pages 5999–6009.

Viol, K., Aas, B., Kastinger, A., Kronbichler, M., Schöller, H. J., Reiter, E. M., Said-Yürekli, S., Kronbichler, L., Kravanja-Spannberger, B., Stöger-Schmidinger, B., Aichhorn, W., and Schiepek, G. K. (2019). Erroneously disgusted: FMRI study supports disgust-related neural reuse in obsessive-compulsive disorder (OCD). *Frontiers in Behavioral Neuroscience*, 13:81.

Vogt, K., Schnaitmann, C., Dylla, K. V., Knapek, S., Aso, Y., Rubin, G. M., and Tanimoto, H. (2014). Shared mushroom body circuits underlie visual and olfactory memories in Drosophila. *eLife*, 3:e02395.

Voss, P. (2019). Brain (Re)organization following visual loss. *Wiley Interdisciplinary Reviews: Cognitive Science*, 10(1):e1468.

Wagstaff, K. L. (2012). Machine learning that matters. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, volume 1, pages 529–534.

Wang, Y., Wang, Q., Shi, S., He, X., Tang, Z., Zhao, K., and Chu, X. (2020). Benchmarking the Performance and Energy Efficiency of AI Accelerators for AI Training. In *Proceedings - 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGRID 2020*, pages 744–751.

Werbos, P. (1974). Beyond Regression: New tools for prediction and analysis in the behavioral sciences. *PhD dissertation, Harvard University.*

White, J. G., Southgate, E., Thomson, J. N., and Brenner, S. (1986). The structure of the nervous system of the nematode Caenorhabditis elegans. *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, 314(1165):1–340.

Wu, C. E., Chan, Y. M., and Chen, C. S. (2019). On Merging MobileNets for Efficient Multitask Inference. In *Proceedings - 2019 2nd Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications, EMC2 2019*, pages 16–20.

Wu, D., Yi, X., and Huang, X. (2022). A Little Energy Goes a Long Way: Build an Energy-Efficient, Accurate Spiking Neural Network From Convolutional Neural Network. *Frontiers in Neuroscience*, 16:365.

Wu, J., Leng, C., Wang, Y., Hu, Q., and Cheng, J. (2016). Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR*, abs/1708.0.

Yamazaki, K., Vo-Ho, V. K., Bulsara, D., and Le, N. (2022). Spiking Neural Networks and Their Applications: A Review. *Brain Sciences*, 12(7):1–30.

Yu, J., Yang, L., Xu, N., Yang, J., and Huang, T. (2019). Slimmable neural networks. *7th International Conference on Learning Representations, ICLR 2019.*

Yue, X. L. and Gao, Q. X. (2018). Contributions of natural systems and human activity to greenhouse gas emissions. *Advances in Climate Change Research*, 9(4):243–252.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer.

Zerilli, J. (2019). Neural Reuse and the Modularity of Mind: Where to Next for Modularity? *Biological Theory*, 14(1):1–20.

Zhang, Y. and Yang, Q. (2018). An overview of multi-task learning. *National Science Review*, 5(1):30–43.

Zheng, L., Jia, C., Sun, M., Wu, Z., Yu, C. H., Haj-Ali, A., Wang, Y., Yang, J., Zhuo, D., Sen, K., Gonzalez, J. E., and Stoica, I. (2020). Ansor: Generating high-performance tensor programs for deep learning. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020*, pages 863–879. USENIX Association.

Zhou, A., Yao, A., Guo, Y., Xu, L., and Chen, Y. (2017). Incremental network quantization: Towards lossless cnns with low-precision weights. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*.

Ziegler, J. C., Montant, M., Briesemeister, B. B., Brink, T. T., Wicker, B., Ponz, A., Bonnard, M., Jacobs, A. M., and Braun, M. (2018). Do words stink? Neural reuse as a principle for understanding emotions in reading. *Journal of Cognitive Neuroscience*, 30(7):1023–1032.