MONASH University

Efficient Deep Neural Networks Aimed for Resource Constrained Embedded Devices

Luis Enrique Guerra Fernandez

Doctor of Philosophy

A Thesis Submitted for the Degree of Doctor of Philosophy at
Monash University in 2024
Electrical and Computer Systems Engineering Department

Copyright notice

Notice

I certify that I have made all reasonable efforts to secure copyright permissions for third-party content included in this thesis and have not knowingly added copyright content to my work without the owner's permission.

Abstract

Deep neural networks initially developed in the 1980s and revived in the 2010s have triggered a new machine learning wave now tangible and soon to be ubiquitous in our everyday lives. Large foundation models have taken by storm both the research community as well as the general public, awakening once again the dream of Artificial General Intelligence (AGI) depicted for long in sci-fi novels and art.

Particularly in the realm of computer vision, deep networks have accomplished remarkable success across the entire field, finding numerous applications ranging from face recognition, image generation to self-driving autonomous vehicles, and with more applications constantly appearing fueled by the community's creativity. Nevertheless, this technology is yet to mature in order to migrate from laboratories and large cloud-based clusters, and be integrated in our every-day electronic devices. One of the main obstacles for this being the large amount of computational resources neural networks require. Steps have been taken towards reducing their computational burden, enabling mobile phones to run them in on-demand applications; nevertheless, additional progress has yet to be made in order for networks to be feasibly equipped in highly resource constrained devices such as specific purpose embedded devices.

In this thesis, the problem of excessive resource requirements is attacked from different angles, aiming at optimizing three resources without sacrificing accuracy: latency, storage and energy consumption. The thesis parts from analyzing how the practicality of some algorithms and not necessarily their performance have made them more widespread than others. Similarly how hardware has influenced software itself.

This is followed by a general introduction to neural networks optimization presented along with a section on optimization under constraints. Afterwards, a series of works produced during my PhD are presented chronologically: starting from revisiting basic strategies such as the decades old pruning approach, which consists in removing unnecessary sections of the networks, directly saving on memory storage and indirectly on energy by reducing memory accesses. Passing through the more recent approach of quantization, which consists in reducing the resolution of the network components, attaining high compression rates and enabling logic-based cheap computation while being more hardware-friendly than the previous approach. And finally through the even more recent approach of network architecture design, which approaches the problem from a higher level perspective. All the methods proposed are complimentary and set bases for further exploration discussed in the final section of the thesis.

Declaration

This thesis is an original work of my research and contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Signature: _____

Print Name: Luis Enrique Guerra Fernandez

Date: _____

Publications during enrolment

Conference and arXiv articles:

1) Guerra, Luis, and Tom Drummond. "Automatic pruning for quantized neural networks." 2021 Digital Image Computing: Techniques and Applications (DICTA). IEEE, 2021.

2) Guerra, Luis, et al. "Switchable precision neural networks." arXiv preprint arXiv:2002.02815 (2020).

3) Guerra, Luis, et al. "Training 1-Bit Networks on a Sphere: A Geometric Approach." Artificial Neural Networks and Machine Learning–ICANN 2022: 31st International Conference on Artificial Neural Networks, Bristol, UK, September 6–9, 2022, Proceedings, Part III. Cham: Springer Nature Switzerland, 2022.

4) Guerra, Luis, and Tom Drummond. "FlyNet: Max it, Excite it, Quantize it." British Machine Vision Conference (BMVC), 2022.

Acknowledgements

I would like to thank Professor Tom Drummond for his trust, support and supervision during my PhD.

I would like to thank Bohan, Ajanthan, Gil and Yan for their direct contributions to my research. Additionally, I would like to thank all my labmates for their indirect contributions and friendship.

I would like to thank my family for their support, especially during the second half of my PhD which was undertaken remotely at home due to the COVID pandemic.

I would like to thank the Mexican government through the CONACYT dependency and the ECSE department at Monash for supporting me with grants and hosting me.

I would also like to thank the now defunct Australian Centre for Robotic Vision (ACRV) for fostering collaborations across Australian universities and research funding.

Finally, I would like to thank Prof. Gustavo Carneiro and Prof. Niko Suenderhauf for examining this thesis and providing constructive feedback.

Contents

List of Figures

List of Tables

Abbreviations

DL	**D**eep **L**earning
CV	**C**omputer **V**ision
CNN	**C**onvolutional **N**eural **N**etwork
MLP	**M**ulti **L**ayer **P**erceptron
KL	**K**ullback **L**eiber
RL	**R**einforcement **L**earning
RNN	**R**ecursive **N**eural **N**etwork
AR	**A**rtificial **R**eality
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit
DSP	**D**igital **S**ignal **P**rocessor
AR	**A**rtificial **R**eality
FC	**F**ully **c**onnected
GPU	**G**raphics **P**rocessing **U**nit
LSTM	**L**ong **S**hort **T**erm **M**emory
IoT	**I**nternet **o**f **T**hings
SoC	**S**ystem **o**n **C**hip
SGD	**S**tochastic **G**radient **D**escent
TPU	**P**rocessing **T**ensor **U**nit
w.r.t	With respect to

Notation

	Numbers and Arrays
a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
$\mathsf{\mathbf{A}}$	A tensor
$\boldsymbol{I}_n$	Identity matrix with n rows and n columns
$\boldsymbol{I}$	Identity matrix with dimensionality implied by context
a	A scalar random variable
$\mathbf{a}$	A vector-valued random variable
$\mathbf{A}$	A matrix-valued random variable
	Sets and Graphs
$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
$\{0, 1\}$	The set containing 0 and 1
$\{0, 1, \dots, n\}$	The set of all integers between 0 and n
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b
$\mathbb{A} \backslash \mathbb{B}$	Set subtraction
$\mathcal{G}$	A graph
a_i	Element i of vector $\boldsymbol{a}$, with indexing starting at 1
$A_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$	Row i of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	Column i of matrix $\boldsymbol{A}$
$\mathsf{\mathbf{A}}_{i,j,k}$	Element (i, j, k) of a 3-D tensor $\mathsf{\mathbf{A}}$
$\mathsf{\mathbf{A}}_{:,:,i}$	2-D slice of a 3-D tensor

Calculus

$\frac{dy}{dx}$	Derivative of y with respect to x
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\nabla_{\boldsymbol{x}} y$	Gradient of y with respect to $\boldsymbol{x}$
$\nabla_{\boldsymbol{X}} y$	Matrix derivatives of y with respect to $\boldsymbol{X}$
$\nabla_{\mathbf{X}} y$	Tensor containing derivatives of y with respect to $\mathbf{X}$
$\frac{\partial f}{\partial \boldsymbol{x}}$	Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ of $f : \mathbb{R}^n \to \mathbb{R}^m$
$\nabla_{\boldsymbol{x}}^2 f(\boldsymbol{x})$ or $\boldsymbol{H}(f)(\boldsymbol{x})$	The Hessian matrix of f at input point $\boldsymbol{x}$
$\int f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral over the entire domain of $\boldsymbol{x}$

Probability and Information Theory

$P(\mathrm{a})$	A probability distribution over a discrete variable
$p(\mathrm{a})$	A probability distribution over a continuous variable
$\mathrm{a} \sim P$	Random variable a has distribution P
$\mathbb{E}_{\mathrm{x} \sim P}[f(x)]$ or $\mathbb{E}f(x)$	Expectation of $f(x)$ with respect to $P(\mathrm{x})$
$\mathrm{Var}(f(x))$	Variance of $f(x)$ under $P(\mathrm{x})$
$\mathrm{Cov}(f(x), g(x))$	Covariance of $f(x)$ and $g(x)$ under $P(\mathrm{x})$
$H(\mathrm{x})$	Shannon entropy of the random variable x
$D_{\mathrm{KL}}(P \| Q)$	Kullback-Leibler divergence of P and Q
$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution over $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Functions

$f : \mathbb{A} \to \mathbb{B}$	The function f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f(\boldsymbol{x}; \boldsymbol{\theta})$	A function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$.
$\log x$	Natural logarithm of x
$\|\|\boldsymbol{x}\|\|_p$	L^p norm of $\boldsymbol{x}$
$\|\|\boldsymbol{x}\|\|$	L^2 norm of $\boldsymbol{x}$
$\mathbf{1}_{\mathrm{condition}}$	is 1 if the condition is true, 0 otherwise

Chapter 1

Introduction: The Importance of Practicality in Algorithms

Artificial Intelligence is a goal that has long been dreamed and speculated about. Creating synthetic machines that resemble, aid or even enhance both our capabilities and the cognitive processes in our minds is a very appealing idea, and although our conceptions of intelligence as such, correctness, ethical implications, our stance and our goals themselves have matured from the original sci-fi robots, the trend and excitement is more fervent than ever.

Over the last decade, the research devoted to this field has grown exponentially after coming from a long winter, and although over the decades similar hype has come and faded fueled by unsupported claims, we now have a more clear path of action, which has already started to bear fruits. The "bitter lesson" essay [1] summarized how trends based on searching and learning that leverage both computation and available data are the ones that have prevailed throughout the years rather than hand-crafted heuristics. The most recent trend is not the exception. We now realize that human-like reasoning is not around the corner; nevertheless, forms of practical decision making previously deemed too complex are now possible.

Deep Learning (DL), an advanced form of signal processing mixed with statistics, is the workhorse behind this recent trend. Inspired by Restricted Boltzmann Machines (RBMs) and belief networks, sparked by the rebirth of Convolutional Neural Networks (CNNs), and further extended by transformers and graph networks, DL takes its name from the

models it relies on. Deep models, or architectures, are comprised of basic building blocks stacked sequentially, a concept taken from Multi-Layer Perceptrons (MLP) and scaled up to conform models orders of magnitude deeper, that is, with more layers.

Nowadays, DL has touched almost every field of science, including but not limited to Natural Language Processing (NLP), audio, chemistry, materials, biotechnology, medicine, ecology and more. Among its numerous application domains Computer Vision (CV) has been a prominent one. In fact, it was the field of CV that initiated the momentum through the large scale ImageNet challenge, which consists in classifying images into one of 1000 categories. CNNs excel at pattern recognition; visual sensory data, being one of the most abundant, highly structured and rich in information, offered fertile land for CNNs to germinate and continues to bear fruits. By means of this observation, in this thesis, we focus on applications in the domain of CV. Particularly, in the tasks of image classification, segmentation and objection detection. However, our ideas can be extended to other tasks, such as video action recognition, image generation, localization, navigation, scene understanding, etc.

Interestingly, the mathematical foundations and algorithms for DL had been set for decades; nevertheless, it was not until recent years that it took off. The reason being the lack of large amounts of easily accessible training data and hardware capable of handling it, in essence, data and compute. Graphical Processing Units (GPUs), originally designed for video processing, quickly grew due to the video games and movies industry. Eventually, the CV community adopted them for fast parallel image processing, which led to the implementation of the first CNN on a GPU [2]. Simultaneously, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [3] was released, consisting of 1.2 million training images distributed across 1000 classes. These two events set the scenario for DL to flourish.

This leads us to rethink the importance of the part played by implementations of algorithms in their success. The hardware lottery essay [4] by Sara Hooker analyzed this perspective in detail. In the essay, the author exposes that historically it has been software and hardware the factors that determine which research ideas succeed and which ones fail. Currently, although DL has achieved remarkable success in real-world tasks, these advancements still do not reflect in our daily lives. The reason being precisely the aforementioned issue, efficient implementations. A mainstream example being the

cost-effectiveness of OpenAI's ChatGPT, an openly available large network with chatting capabilities and multimodal understanding, unable to offset the costs of inference with advertisements.

The first neural networks to appear had parameters in the order of a few thousand [5] but in the past decade quickly scaled to tens of millions [2, 6], billions [7, 8] and even trillions [9]. Power consumption, storage memory and latency have become the bottleneck in the progress of DL as neural models growth has outpaced available compute [10]. It is reported that training a GPT-3 model used for NLP costs millions of dollars to train for just one iteration [11], not including the cost of hyperparameter sweeps and experiments. This model comprises 175 billion parameters. Strubell et. al [12] estimated the cost of training several mainstream DL models both in terms of cost (USD) and CO_2 emissions (lbs). They found that the carbon emission of training a large model surpasses 300,000 kg, which corresponds to about 5 times the carbon emitted by a car during its lifetime. "The carbon impact of artificial intelligence" [13], published in Nature, delved more into details of the environmental and financial implications of training these models.

Nevertheless, long DL training sessions is only one side of the problem. The second being inference. This is, once these DL models have been trained they must be deployed in order to have practical importance. For that reason, DL neural networks are mostly used in academic laboratories. Industry has started to integrate them through cloud-based applications running on remote computer clusters. However, steps have to be taken in order for them to be implemented in all kinds of electronic devices.

Initial steps to reduce these costs have already been taken. In the recent years, ranging from low rank matrix decompositions [14, 15], pruning [16–18] and sparsification [19, 19–21], quantization [22–25], architecture search [26–29], efficient submodules [30–33], weight sharing [34] and more methods for compressing DNNs have appeared in all forms. However, as observed by [13], 90% of papers from the 2018 Annual Meeting of the Association for Computational Linguistics, 80% from the 2018 Conference on Neural Information Processing Systems (NeurIPS), and 75% from the 2019 Conference on Computer Vision and Pattern Recognition prioritized accuracy over efficiency.

Mobile phones-focused DL implementations are only the starting point towards developing real highly efficient deep CNNs. The remaining challenges ahead are still plenty, and in this thesis, part machine learning, part systems, I aim to develop tools towards

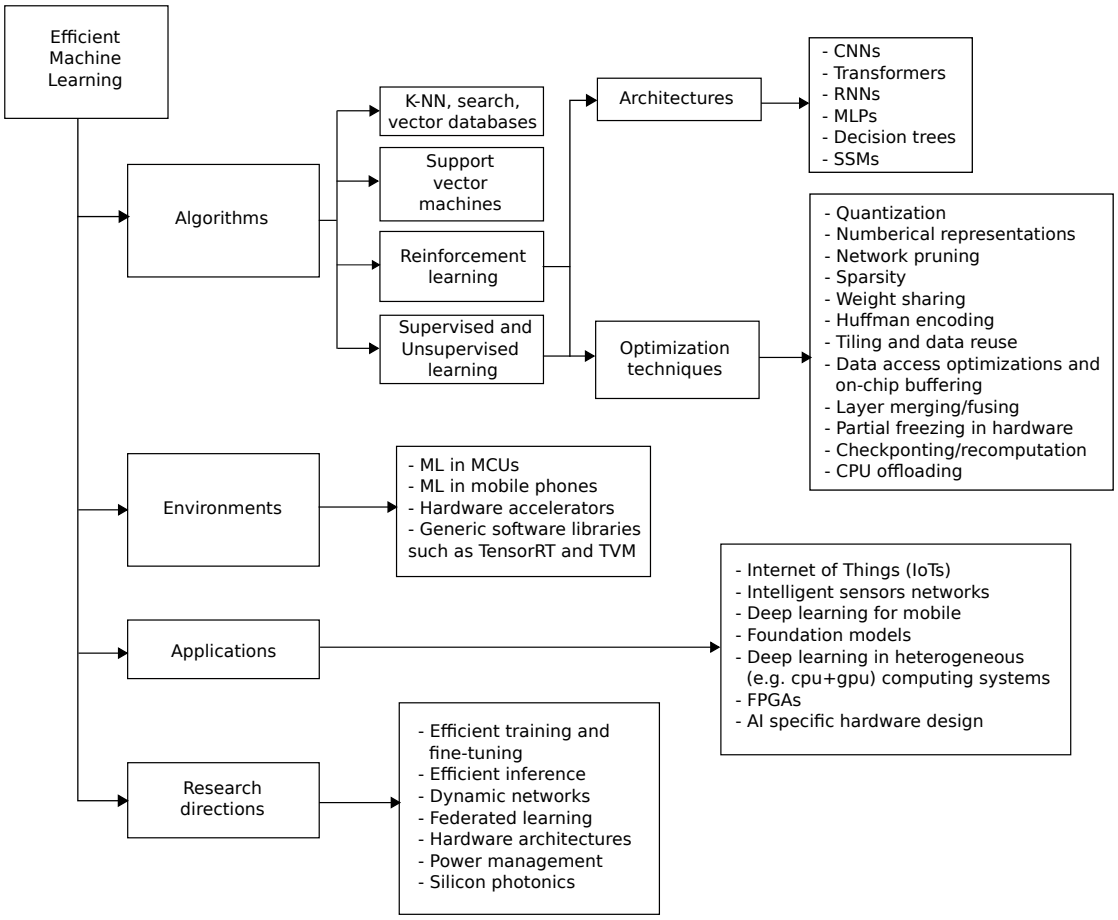


FIGURE 1.1: Layout of efficient machine learning.

that goal. Figure 1.1 depicts a broad perspective of the field of efficient machine learning including algorithms, environments, applications and research directions.

1.1 Deep Learning for Vision

The vision community has a distinction between image processing and computer vision. Image processing is used to refer to low-level vision (e.g. denoising, stitching, keypoint correspondence matching, fixing of aberrations, super resolution), while computer vision is used to refer to high-level vision that involves reasoning over the semantics of the images (e.g. classification, semantic segmentation, localization, visual question answering, action recognition). However, deep learning has made that distinction blurry by simply dominating the entire spectrum. Deep neural networks have replaced most hand-crafted solutions prevailing until not so long.

Deep Learning is used to refer to the set of models relying on the paradigm of multi-layer neural networks trained through backpropagation, historically also known as *connectionism*. There are numerous types of neural network layers; however, all deep networks follow the same basic design guideline, sequentially stacking parameterized non-linear transformations.

This simple design approach of stacking learnable non-linear transformations has proven to be quite powerful at recognizing underlying patterns in data. Understanding the inner mechanism of this paradigm has been an active subject of study since its initial success. Initially criticized for their black-box nature, nowadays considerable amount of literature has shed light on the inner organization of deep neural networks. From a broad perspective, the initial layers perform the low-level vision. They learn border detectors, smooth detectors, corner detectors and so on. Consecutive layers combine those low-level features to recognize higher-level concepts like circles, squares, textures, body parts and simple objects. Moving on higher on the hierarchy, the final layers will recognize even higher level concepts such as dogs, cat, planes and cars.

In the full process, first the sensory input images get digitized into a matrix. The set of all possible input images is known as *image space*. Deep networks transform this matrix into a feature vector, and analogously the set of all possible feature vectors is denoted the *feature space*. Ideally, these feature vectors should group similar semantic concepts next to each other. Training a deep network to do so is a topic of study that has produced a vast amount of literature over the past decade. For example, endowing the feature space with a metric where distances have meaning will create a *metric space*. Learning this feature space with target labels is denoted *supervised learning*. Learning it without or sparse labels is denoted *semi/un-supervised learning*. By finding a transformation that will invert the process, that is, learning to sample from the data distribution, is coined *generative training*. Learning feature spaces that are robust to perturbation in the input data is denoted *adversarial training*. The way these feature spaces are shaped is usually through regularizations and creative loss functions. In general, the overall field is also sometimes referred as *representation learning*.

The selection and configurations of the layers of neural networks are commonly referred as *architectures*. Finally, tuneable training settings are known as *hyperparameters*.

Different neural network layers exist for different purposes and will be briefly described below.

Linear layer. Proposed in Rumelhart et al. [35], this is the most basic type of layer. It consists of an affine transformation of the input data with two learnable vectors denoted as weights and biases: $f(\boldsymbol{x}) = \sigma(\boldsymbol{x}\boldsymbol{W} + \boldsymbol{b})$ where $\boldsymbol{x} \in \mathbb{R}^k, \boldsymbol{W} \in \mathbb{R}^k, l, \boldsymbol{b} \in \mathbb{R}^k$ and σ is an activation function. Each row vector of $\boldsymbol{W}_{:,i}$ along with its corresponding bias b_i is called a *neuron* and it "fires" when it correlates with the input. The activation function σ is commonly non-linear, and the layer is then referred as *fully-connected* or *dense layer*.

Linear layers are usually placed as the last layer in a deep model given that they can observe the entire feature vector and use it to make decisions, such as classification. This layer is sometimes called the head of the network. Additionally, a softmax$(\cdot)$ layer is usually placed on top of it to turn it into a differentiable one-hot encoding layer (where all the elements of the vector are zero except for one). This softmax$(\cdot)$ is simply replaced by an argmax$(\cdot)$ during inference.

A sequence of multiple consecutive linear layers is usually referred as feedforward network or as a Multi-Layer Perceptron (MLP).

Convolutional Layer. The convolutional layer originally proposed by LeCun et al. [36] and popularized by AlexNet and VGG [2, 37] is tailored for images, audio and structured grid-like data. It can be extended to unstructured data such as graphs, however, it requires a notion of neighborhood and deal with non-equidistant nodes. It is based in the convolutional operation used for filtering in the signal processing field, with the difference that filters are learned rather than hand-crafted (e.g. Laplacian, Gausssian and Sobel filters [38]). Formally, a discrete 1-dimensional convolution is defined as:

$$(f * g)[n] = \sum_m f[m]g[n-m] \tag{1.1}$$

In essence, a convolution takes an input signal g indexed by n and takes the dot product with the filter f of size m at every location, usually $m << n$. In simple words, it is a dot product with a sliding window. Just like linear layers, it is easy to see that this operation is highly parallelizable. In fact, in practice, convolutions are implemented as linear layers preceded by an unfold operation (the unfold operation takes the input g and extracts patches of size m around each element indexed by n).

Convolutional layers are very flexible and are parameterized by a set of arguments described below:

- Input channels: From the perspective of signal processing, signals from the same domain but from a different source are denoted channels. In RGB images the red, blue and green denote different input channels.
- Output channels: Each filter in a convolutional layer creates a new output channel; thus, this number denotes the number of filters per layer.
- Kernel size: Used for specifying the size of the filter.
- Stride: Used for spatial downsampling, it controls the jump length.
- Padding: Used for filling the space around the image when the convolutional filter exceeds the valid dimensions.
- Dilation: Controls the spacing between the filter kernel elements. Used for controlling effective view field.
- Groups: Denotes the number of input channels each filter can access. Used to reduce the number of parameters and FLOPs.

Recurrent layers. Recurrent layers are used to process sequential data. They are characterized by having a feedback loop from the output to the input and maintaining a state. In general, given an input sequence $\boldsymbol{x} = [x_1, ..., x_T]$ of length T, RNNs operate by repeatedly applying a function f_h, which generates a hidden state $\boldsymbol{h}_t$ for time step t:

$$\boldsymbol{h}_t = f_h(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}) \tag{1.2}$$

Depending on the function f_h, there are many types popular types of RNNs described below:

- Recurrent Neural Network (RNN) [35]. Is the most basic type of RNN, it implements the function: $f_h(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}) = \sigma(\boldsymbol{x}_t\boldsymbol{W} + \boldsymbol{h}_{t-1}\boldsymbol{U} + \boldsymbol{b})$, where $\boldsymbol{W}$ and $\boldsymbol{U}$ correspond to the weights of the input and the recurrent connections.

- Long Short Term Memory (LSTM) [39]. A more complex version of the RNN, it is composed of a cell, an input gate, an output gate and a forget gate. The additional gates are used to deal with the vanishing and exploding gradients.
- Gated Recurrent Unit (GRU) [40]. A simplified version of the LSTM where an output gate has been removed.

Transformers. Attention-based models such as transformers [41] originated in the Natural Language Processing (NLP) field. Similarly to RNNs they are used to model sequential data; however unlike RNNs they model relationships rather than storing a state, meaning that, the distance between two inputs in the sequence will be irrelevant to the performance of the task, unlike RNNs that slowly forget previous inputs. The attention mechanism operates in the following way: Every input in the sequence is combined with a positional encoder in order to create unique inputs for each value-position combination. Subsequently, each combination is lifted into a learned feature space of dimension d_{model}, this sequence of combinations is named *queries*. The same sequence (in the case of self-attention) is lifted into a second learned feature space, this sequence is named *keys*. Alternatively, a different sequence can be used (in the case of cross-attention) as keys. Self-attention is computed in these lifted feature spaces. The cross dot-products are computed and a softmax per input is applied; thus, now there is a distribution vector per query relating to each element of the keys. A copy of the keys, denote *values*, is created. Such values are mixed by a weighted combination defined by those distributions. This process is executed for n heads in parallel for different learned feature embeddings, and lastly, the n feature vectors per query are combined with a linear layer. Therefore, the output of each layer is of the same size as the input. Formally, let $\boldsymbol{Q}, \boldsymbol{K}$ and $\boldsymbol{V}$ denote the queries, keys and values, the attention vectors are computed as:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}})\boldsymbol{V}, \tag{1.3}$$

where $\sqrt{d_k}$ is a scaling factor used to counteract the effect of large values of d_k which in turn generate large values of dot-products, pushing the softmax into regions of saturation.

Multi-head attention allows the model to jointly perform attention in different learned subspaces:

$$\begin{aligned} &\text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = [head_1, ..., head_h]\boldsymbol{W}_O, \\ &\text{where} \quad head_i = \text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V})_i, \end{aligned} \tag{1.4}$$

with $\boldsymbol{W}_O \in \mathbb{R}^{hd_v \times d_{model}}$.

Transformers have impacted the CV field in two ways. One, as complimentary building blocks along with convolutional and linear layers [42, 43]. Two, as stand-alone CV models [44–46], performing tasks solely based on transformers. These second type of models were only developed recently and although they appear promising, are not still on par with CNNs on the efficiency-accuracy trade-off [47].

Restricted Boltzmann Machine (deprecated). RBMs [48] are probabilistic undirected graphical models trained with Markov Chains Monte Carlo (MCMC) methods. They can be interpreted as generative bidirectional neural networks with stochastic neurons. They resemble modern-day feedforward networks and diffusion models [49]. They became popular in the pre-deep learning era and worked as stepping stone; however, these days are falling in misuse. The reason being mainly its difficulty to train and binary nature even after the development of Gaussian RBMs [50]. RBMs, similarly to deep learning networks, are stacked to form *belief networks* [51].

1.2 Embedded Systems.

Specific purpose embedded devices are electronic systems that, unlike smartphones or computers, serve only a single purpose (e.g. microwaves, wearable gadgets and music players), they can't be reprogrammed to perform a different task. Sometimes rather than operating as standalone systems, they are part of a larger system (e.g. cars and computers). One particular characteristic of embedded devices is that they are intended to be as optimal as possible: starting from the selection of the components (sensors, integrated circuits), to energy consumption and computational resources, particularly the battery-powered ones. Additionally, they commonly operate in real-time. For example, a microcontroller can be as cheap as a fraction of a US dollar, consume 150μA/MHz on active mode and run for years on a coin cell battery.

In general, an embedded system consists of a microprocessor, memory and input/output peripherals. Often several components are packed together in a microcontroller. Specialized microcontrollers with processors dedicated for FLOPs and common mathematical routines (e.g. Fourier transforms) denominated Digital Signal Processors (DSP) also exist.

The number of embedded systems in the world surpasses overwhelmingly the number of general-purpose devices; however, most of them pass unnoticed. On average a new car contains around 30 embedded systems, with some luxury models containing up to 70. With the rise of the Internet of Things (IoT) these numbers are expected to grow significantly and be interconnected. The global market value of embedded systems in 2020 was estimated to be USD 4 billion and is projected to rise to 8 billion by 2025.

Designing tailored deep-learning algorithms for embedded devices and furthermore, tailored for each particular embedded system is an interesting economic and research avenue that will continue to grow as deep-learning systems start to experience diminishing returns. This is, once the task performance gains coming from deep learning systems start becoming marginal and revenue switches to efficiency gains.

In this thesis, we have developed techniques that allow deep neural networks to operate in the lower computational regimes of embedded devices and that can be further customized for particular hardware. One example of this would be training a single large SP-Net (see Chapter 5) and deploying different switches based on the hardware capabilities. This would allow to simultaneously save training costs and enhance the quantized switches. Another example would be to have a slider to trade-off between memory and FLOPs using Multihead-Depthwise Convolutions (see Chapter 8) depending on the characteristics of the hardware.

1.2.1 Deep Learning on the Edge.

In order for DL to truly become a practical technology that influences our lives it must make the jump into embedded devices, and more generally speaking to computation on the edge. Nevertheless, the task is not trivial. Edge devices are characterized for operating under tight constraints while simultaneously running at real-time.

In the markets, some companies have already started developing deep learning software frameworks for mobile platforms: Pytorch Mobile by Facebook, TensorFlow Lite by Google, ELL by Microsoft, ARM-NN by ARM.

Of particular interest are DL ARM targeted implementations. ARM is a hardware design company that licenses its Instruction Set Architecture (ISA) design to chip manufacturers for them to design their own System on Chips (SoCs). ARM supplies with an instruction set that compilers can use to translate higher-level code into machine code interpretable by their processors. Intel's x86 architecture used in desktop CPUs, uses a CISC (Complex Instruction Set Computer) processor, unlike ARM processors which use a RISC (Reduced Instruction Set Computer) architecture preferred in mobile devices and growing in popularity among cloud servers. ARM has market dominance over the mobile market, to this day over 160 billion devices that include ARM processors have been shipped. This dominance is an example of efficiency outperforming complexity in the markets.

From the hardware side, there have been efforts as well. ARM has already started designing hardware IP for heterogeneous chips targeting DL applications, such as the Mali GPU and the Ethos Neural Processing Unit (NPU), which are operable through the ARM-NN, ARM Compute-Library and CMSIS-NN interfaces, which in turn are operable through high level frameworks such as Pytorch, TensorFlow Lite and the Open Neural Network Exchange (ONNX). Other available platforms are Nvidia's Jetson, Intel Movidius Neural Compute USB stick, Google's Coral board based on their TPUs, BeagleBone AI by Texas Instruments and the Akida Neural Processor by Brainchip.

Among the obstacles faced in trying to translate DL applications into edge devices are: low power consumption, small memory footprint, low latency and high throughput demands.

The energy consumption in a system is dictated both by the operations taking place in the processor and by the memory accesses. In fact, moving data requires considerably more energy than executing the operations, especially if data is fetched from the off-chip DRAM. This is an important consideration when designing CNNs given that the available SRAM is regularly much smaller than the size of the models. To a lesser extent, the power consumption of computation is also dictated by the precision of the data. Table 1.1 lists the energy consumption of different operations. In this regard, ShuffleNetV2

TABLE 1.1: Energy consumption (pJ) of processor operations. Source: Bill Dally, Cadence Embedded Neural Network Summit, 1 February 2017

Operation	Energy (pJ)
8 bit int ADD	0.03
16 bit int ADD	0.05
32 bit int ADD	0.1
16 bit float ADD	0.4
32 bit float ADD	0.9
8 bit MULT	0.2
32 bit MULT	3.1
16 bit float MULT	1.1
32 bit float MULT	3.7
32 bit SRAM READ	5.0
32 bit DRAM READ	640

TABLE 1.2: Hardware comparison for CloudML, MobileML and TinyML

Operation	Architecture	Memory	Storage	Power	Price
CloudML	GPU	HBM	SSD/Disk		
Nvidia V100	Nvidia Volta	16GB	TB PB	250W	$9K
MobileML	CPU	DRAM	Flash		
Cell Phone	Mobile CPU	4GB	64GB	8W	$750
TinyML	MCU	SRAM	eFlash		
F446RE	Arm M4	128KB	0.5MB	0.1W	$3
F746ZG	Arm M7	320KB	1MB	0.3W	$5
F767ZI	Arm M7	512KB	2MB	0.3W	$8

[52] provided design guidelines to design architectures with reduced number of memory accesses.

Both the volatile and non-volatile memory in the devices are also two factors to consider. Non-volatile used to store the network's weights and parameters is usually small compared to the size of the current networks. Additionally, the volatile memory required to store the activations is also not abundant. Traditional DL models have drastically higher peak memory than an embedded system can provide. For example, Lin *et al.* found in [53] that at similar ImageNet accuracy, MobileNetV2 [32] reduces the model size by 4.6× compared to Resnet-18 [34], but the peak activation size increases by 1.8×, making it even harder to fit in a microcontroller. Table 1.2 illustrates the different platforms and the significant differences in resources.

Latency and throughput are also factors to be considered, particularly when there are real-time constraints. MicroNets [54] characterized different layers and found a strong linear relation between FLOPs and latency.

Ultra-low-power machine learning, or TinyML, is the recent term used to describe ML for embedded devices. TinyML will enable a new class of smart, always-on applications that can revolutionize the collection and processing of data. In order to have a fair and reliable method of comparison, Banbury *et al.* [55] investigated the problem of benchmarking TinyML systems. They selected four datasets to represent common TinyML application domains: Visual wake words, audio wake words, image classifications and anomaly detection.

Audio and visual wake words datasets are examples of always-on ML. It is an audio or video binary classification problem that labels every event as either a "wake word" or "not wake word". Anomaly detection is a sequential data problem that labels as "normal" or "abnormal" a time series. It can be used to identify faults in products or equipment in factories, where different types of data can be collected to perform predictive maintenance.

As TinyML grows, it will continue to find new applications. A domain where TinyML can have a significant impact is Augmented Reality (AR) in head sets whose fundamental features are always-on and battery-powered. In these headsets the computation can be offloaded to mobile phones; however, it would benefit significantly from computing on the device itself.

Finally, the last obstacle faced by mobile and TinyML are hardware and software heterogeneity. Mobile phones and embedded devices are much more varied than GPUs of TPUs used in CloudML. The mobile phones and embedded markets are well balanced with multiple vendors and even novel architectures like event-based neural processors. Additionally, there are diverse options when it comes to performance, power and capabilities.

1.3 Machine Learning and Systems

Despite the contributions of this thesis being mainly theoretical, the topic relies in the intersection of ML and systems. Linking these fields is crucial for further growth. Deep Learning offers the promise of creating disruptive change; however, in order to have a positive, meaningful impact, its usability must improve. The community has identified this weakness and taken actions to address it. The recently started conference of Machine

Learning and Systems (MLSys) [56], precisely aims at this task. It is a joint effort from major universities and industry research departments.

The inaugural organizing committee points out the following concerns that arise in DL projects as DL gains broader usage:

- Cost. Both of development and deployment.
- Accessibility. Usable in multiple languages in order not to marginalize minorities.
- Automation. Defining pipelines such as MLOps (short for ML Operations), which includes data engines, hyperparameters tuning and experiments management.
- Latency. Real-time operations.

Specifically, while it is easier to run ML models than before, implementing them in real-world applications is increasingly harder. The so-called "Software 2.0" involves a paradigm shift which is data-centered and involves redefining several processes like quality assurance. This shift opens opportunities for both high and low level interfaces. Shortening the development cycles and lowering the skill set required for training DL models is one of the targets of the conference.

Another set of concerns more related to the DL models themselves are:

- Fairness. Machine learning models can perpetuate and amplify societal biases if they are trained on biased data.
- Bias. Models representative of the true data distribution. Bias can be tested by comparing the model's performance on subgroups within the data.
- Robustness. Models should be robust to distribution drift and adversarial attacks to ensure their reliability.
- Security. Guaranteeing the model's integrity is important, just like avoiding the injections of back-doors and revealing sensitive data. Avoiding misuse is another cause of concern.
- Privacy. Preventing unauthorized access of sensitive information, both during training and after deployment remains a challenge.

- Interpretability. Explaining decisions and debugging is important in order to further improve deep models iterations.
- Causality. It is important to understand what factors in the input affected the output in order to improve model performance and enhance reasoning capabilities by incorporating domain knowledge about relationships in the data.

Additionally, it recognizes the line, although blurry sometimes, between high and low-level ML systems. In fact, DL itself could help in the design of systems to run DL applications.

Although this thesis does not have a direct impact on multiple of the aforementioned topics, it does foster accessibility by lowering the cost of deployment of DL systems. We hope that by providing affordable technologies aids in closing the wealth and educational gap in marginalized individuals and communities.

1.4 The carbon impact of artificial intelligence

Although not visible at first sight, AI has had a non-trivial impact at global scale in terms of resources consumption and has had its share when it comes to the climate crisis. As stated by Dhar [13], AI has a double role in climate change. On one hand, it can help reduce the impact of humanity by aiding in the development of low-emission infrastructure, modeling climate change predictions and aiding in the smart grid design. On the other hand, AI itself is a major emitter of carbon.

As previously mentioned, the carbon footprint of training a large NLP model can surpass 300,000 kg of carbon dioxide emissions. However, this is only the direct cost and indirect factors should also be considered. Among them are: the location of the server and the energy grid it uses, the length of the training procedure, and the hardware. In order to estimate the real impact of training AI models, Lacoste *et al.* presented a Machine Learning Emissions Calculator [57]. Although their analysis focuses on the training phase rather than on inference, the authors do provide a section on efficient hardware. According to them, a CPU can be 10 times less efficient than a GPU while an ASIC like Google TPU can be 4 to 8 times more efficient than a GPU. However, in the context of

embedded systems, a GPU optimized for low-power can be 10 to 20 times more efficient than a regular GPU.

According to Amodei and Hernandez [58], the compute used in AI training models had been doubling every 3.4 months since 2012, accounting for a 300k times increase.

In this thesis we focus on efficiency during inference rather than training, whose impact is harder to quantify, but equally importantly reducing DL models power consumption and hardware requirements.

1.5 Privacy

There are a few reasons why it is preferable to run DL models on the edge rather than on the cloud, among them are reduced latency and reduced power consumption caused by eliminating the overhead from communication. However, with the advent of the IoT, a growing concern is privacy. With governments and telecommunications companies facing rumors of spying on the general population, major companies being denounced by collecting users' personal data, hacking and data theft, it is no surprise that users of mobile devices are concerned about their privacy. A harmless cleaner robot equipped with a camera could accidentally expose recordings of a person's house by uploading videos that can be intercepted by malicious software.

Apart from mobile devices, there are domains where people would prefer to keep their records secure, such is the case of medical data. Therefore, there is a need to develop *both* training and inference algorithms that can run on the edge.

1.6 Federated learning.

Federated learning [59, 60] offers a natural solution for the privacy concern of training on the edge. Federated learning is a form of collaborative decentralized learning where a single model is trained with large amounts of heterogeneous data from several in-field devices, and devices in different locations maintain their own copy of the model. There are several topologies in federated learning, depicted in Figure 1.2; however the main ones include learning on each device according to the available data, and sending the

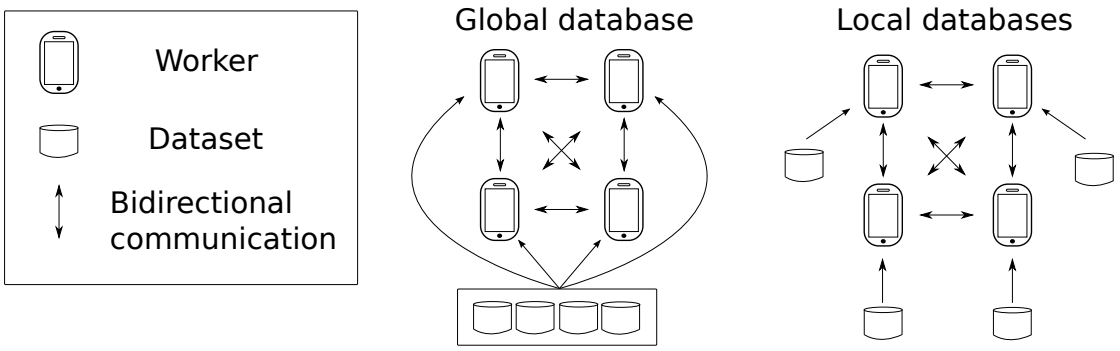


FIGURE 1.2: Figure illustrating how decentralization in different layers leads to different learning systems. Global database: A fully centralized system where workers sample from shared and shuffled data. Local databases: Based on the previous, workers maintain their own data sources, making it decentralized in the application layer. Our framework and theory are applicable to all kinds of decentralized learning systems.

updated model back to a centralized node where all the models get combined in some way [61]. This is with the purpose of not transmitting the training data ensuring privacy. Federated learning also allows for personalization. Through clustering, similar models can be grouped to create personalized models according to their local data.

Since training takes place in the device, it is also required to develop efficient re-training routines, opening opportunities for efficient training.

Related concepts to federated learning are the ones of distributed learning and fog computing, where a large process gets split in several small nodes, for example training a DL model. The main difference between distributed learning and federated learning lies in the assumptions made on the properties of the local datasets, federated learning does not assume identical independent distributed (i.i.d.) data observed by each device.

Although the federated learning approach was only recently proposed, it has received quick acceptance and still a lot of challenges lie ahead.

1.7 Applications of Deep Learning on the Edge

Intelligent Sensor Systems and IoT. Internet of Things (IoT) is one of the disruptive technologies driving the currently named fourth industrial revolution, and one of the ones that has a tangible impact in our daily-lives. It consists in endowing all kind of electronic devices with networking capabilities sharing the same protocols that define the existent global internet network to create a massively interconnected shared network of devices

and people. This means that we will be able to interact with our appliances, as well as devices will be able to interact between each other, through an internet connection.

DL along with the internet of things will be the driver for smart cities with networks of interconnected vehicles, traffic controllers and transportation systems in general, surveillance cameras, electricity distribution through a smart grid, water management systems, infrastructure management, environmental monitoring such as noise pollution, air and water quality, atmospheric conditions, wildlife. Finally, IoT can be used for emergencies, such as earthquake or tsunami early warning systems.

Among the industrial applications of IoT are manufacturing, agriculture, food industry and maritime industries. In the manufacturing field, smart factories with sensor networks and production line coordination include connected equipment, operational technology, storage units monitoring, identification systems, process automation, predictive maintenance, and energy optimization. In farming, IoT can be used for data collection on temperature, rainfall, humidity, wind speed, pest infestation and soil content. Additionally, minimize risk and waste, crop management and precision fertilization. Agricultural tractors equipped with vision sensors can be used to pick crops. Drone and satellite hyperspectral imagery can be used for crop plague inspection and diseases. In the food industry, food supply chains can be improved, automated proof of delivery, increase efficiency of logistics, cold chain and short life products monitoring and food inspection. In the maritime field, it can be used for systems on boats and yachts, for maintenance and early alert of damage.

Across organizational applications, such as Medical and healthcare, it has applications in remote health monitoring and emergency notification systems, as well as health data analysis and assisted diagnosis. In buildings and home automation, it has applications in the monitoring and control of mechanical, electrical and electronic systems in order to create smart buildings, real-time monitoring for energy consumption and monitoring of occupant behavior.

Finally, in consumer applications, it has uses in smart homes where appliances, heating and A/C systems, entertainment, security and illumination can be configured according to the resident's preferences and schedules. Additionally, in elder care and assistance to people with disabilities and health monitoring.

Smart mobile phone apps. DL has a big potential for cell phone applications, some have already started appearing. Mobile vision applications range from identification by face-detection, text recognition, photo editing, augmented reality for games and for tutoring on how to perform a task, motion videos creation from photos, skin care apps, recognition of animals species, automatic animations creation, and photo style transfer. In other domains apart from vision have also been successfully applied like dating apps, music and media recommendations, restaurants recommendations, retailers use it for product recommendations, chatbots, restaurants reviewing, speech recognition, text classification and translation, and auto-completion.

1.8 Applications on Foundation Models

Foundation Models, also referred as Large Models (LMs), comprise a new set of mainstream models developed by large laboratories by scaling up neural networks to extreme regimes on all aspects: size, data and compute. Their training sets often include entire encyclopedias, books, online wikis, software and most written text ever produced by humans. Given the extensiveness of the corpus of data collected to train them, these models become few-shot learners [11]. This means that they can perform new tasks not seen during training without the need to fine-tune them. This is commonly accomplished by presenting the model with example inputs and output pairs. Interacting with these models to produce desired functionality has been popularly been denote as "prompt engineering", or somewhat sarcastic as "spell casting".

Foundation models, although at first sight might seem like trivial research, in combination with creative and clever usage, have come to revolutionize the research community, helping solve tasks previously approached from completely different perspectives. One example is robotics: commonly approached through Reinforcement Learning (RL) actions and rewards system, it has been reformulated as an instruction executor, with instruction sets coming from foundation models detailing how to perform the desired actions.

These models are usually made available online to the general public via some visual interface, and to researchers through APIs for bulk prompting. Their popularity and demand have exploded, and new versions are being released by the day. Ranging from

parameters in the billions [8] to the trillions [9], searching for efficient ways to train them is a hot topic of research [62, 63] as it will allow to deploy them more quickly and make them more cost-effective.

1.9 Thesis structure

Now that we have outlined the progress and denoted the necessity for efficient models, we will detail our contributions in the remaining chapters. This thesis is organized in major sections in the following way:

- In Chapter 2 the foundations of neural networks optimization are presented. Training lighter networks is both a combination of proper network design as well as proper training techniques. Therefore, this chapter will function as an introduction for the subsequent chapters. Of particular interest is the ADAM optimizer which significantly improves quantized neural networks training.
- Techniques for optimizing learning problems with constraints in the solution space are presented in Chapter 3. These constraints are selected to reflect as savings during the inference phase.
- In Chapters 4, 5, 6, 7 and 8 approaches for designing lightweight Neural Networks are presented along with the improvements proposed during the duration of my PhD.
- Chapter 9 concludes this thesis by adding implementation details and outlines promising future directions in this line of research.

Chapter 2

Neural Networks Optimization

The field of optimization consists in general in finding the parameters θ of a mathematical model that reduces a cost function $J(\theta)$ which typically includes a performance measure evaluated on some target set as well as other regularizations.

Optimization in the context of machine learning differs slightly from the field of pure optimization. Machine learning problems usually deal with uncontrolled real-world scenarios and they act indirectly both on the data and on the task. The methodology usually involves partitioning the available training data into 2 or 3 sets: training, validation, test. The reason for this practice is that the training data is only a sample of the entire data distribution which is inaccessible for one of the following reasons: we don't have access to the device or scenarios where it was acquired, or given that the world is continuous, endless amount of data could be collected unnecessarily making training intractable. Therefore, in machine learning the usual scenario consists in maximizing some performance measure P that is defined with respect to a test set and is intractable. Thus, P is optimized indirectly by reducing a cost function $J(\theta)$ hoping it will improve P. This is in contrast to pure optimization, where minimizing J is a goal itself. Additionally, usually the cost functions usually include regularizations or different types of supervision.

Typically, the cost function is written as an average over the training set:

$$J(\theta) = \mathbb{E}_{\boldsymbol{x},y\sim\hat{p}_{data}} L(f(\boldsymbol{x};\theta), y), \tag{2.1}$$

where L is the per-example loss function, $f(\boldsymbol{x};\theta)$ is the predicted output when the input is $\boldsymbol{x}\sim \hat{p}_{data}$ is the empirical distribution and y is the target output.

Equation 2.7 defines an objective function w.r.t the training set. However, it would be preferable to minimize the objective function where the expectation is taken over the data-generating distribution p_{data} rather than over the finite training set:

$$J^*(\theta) = \mathbb{E}_{\boldsymbol{x},y\sim p_{data}} L(f(\boldsymbol{x};\theta), y), \tag{2.2}$$

denote the expected generalization error or *risk*. As mentioned before, if the true distribution p_{data} was known, risk minimization would be a classical pure optimization problem; however, we only have access to a set of samples. Therefore, the true distribution is replaced with the empirical distribution $\hat{p}(\boldsymbol{x}, y)$ and we minimize the *empirical risk*

$$\mathbb{E}_{\boldsymbol{x},y\sim \hat{p}_{data}}[L(f(\boldsymbol{x};\theta), y)] = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\theta), y^{(i)}), \tag{2.3}$$

where m is the number of training examples.

The process of minimizing this average training error is known as *empirical risk minimization*. One issue of empirical risk minimization is that it is prone to overfitting. This means that models with high capacity can memorize the training data. The aforementioned validation set is used to halt the training process when overfitting starts to occur.

An optimization problem can also be viewed from the perspective of maximum likelihood estimation:

$$\theta_{ML} = \arg\max_{\theta} \sum_{i=1}^{m} \log\, p_{model}(\boldsymbol{x}^{(i)}, y^{(i)}; \theta). \tag{2.4}$$

Maximizing this sum is equivalent to maximizing the expectation over the empirical distribution defined by the training set:

$$J(\theta) = \mathbb{E}_{\boldsymbol{x},y\ \hat{p}_{data}} \log\, p_{model}(\boldsymbol{x}, y; \theta). \tag{2.5}$$

Computing this expectation is very expensive since it requires evaluating the model on every example in the entire dataset. In practice, it can be computed by randomly sampling a small number of samples from the dataset. Most optimization algorithms

converge faster if they are allowed to rapidly compute approximate estimates of the gradient rather than slowly computing the exact gradient. One of the reasons for this is the redundancy in the dataset. Additionally, it provides a regularization effect due to the noise added as long as samples are shuffled, otherwise the gradient estimate will be biased.

Algorithms that rely on the entire training set are denominated batch or deterministic gradient methods. Although batch/minibatch is also used to refer to its counterpart, Stochastic Gradient Descent (SGD), where the batch size refers to the number of examples averaged per iteration. Another type of sampling is the denoted *online*, which is used to refer to samples that are dynamically generated typically based on the previous actions or decisions made by the algorithm. Online learning is commonly found in the context of Reinforcement Learning (RL), which won't be covered in this thesis.

For more on neural networks optimization, we refer the reader to [64].

2.1 Challenges in Neural Network Optimization

Ill-Conditioning. Conditioning has been defined in several, nevertheless, closely related ways. It is a measurement of how sensitive a model is to certain input perturbations or directions, meaning that, small deltas in inputs can cause large deltas in the output. The conditioning of a matrix is related to the curvature of the loss surface. It can additionally be analyzed from the perspective of the eigenvalues of the parameter matrix. A large curvature in a particular direction and small in another one, will cause the mentioned effect. Similarly, a large ratio between the largest and the smallest eigenvalues will have the same effect.

Ill-conditioned parameter matrices usually led to poor convergence, overshooting and divergence, or post-training adversarial vulnerabilities.

Local minima. Local minima refer to solutions of the optimization problem that are not optimal; nevertheless, the algorithm cannot find a direction to move that will reduce the cost function and gets stuck. Several methods have been proposed in order to avoid falling into this type of minima including momentum, regularizations, weight decay,

data augmentations, etc. However, it has been observed that in high dimensions, local minima are rare and are usually almost or just as good as the global minimum.

Plateaus, Saddle Points. Apart from local minima, other interesting regions occur in the optimization landscape. Examples are flat regions with small gradients and saddle points where gradients are positive in some directions and negative in others, while having no curvature at the saddle. While they are not of major relevance to first-order optimization methods, they can be critical for second-order methods such as Newton's method.

Exploding and Vanishing Gradients. Exploding gradients occur when the magnitude of the weights starts growing conversely causing the magnitude of the gradients to grow in a cycle, leading to an explosion in the magnitude of both. This issue is usually avoided by good initialization strategies. More recently batch-normalization layers have also alleviated it. Vanishing gradients on the other hand, occur when gradients begin to shrink as they are propagated from the last layer of the network to the initial ones preventing them from learning. Vanishing gradients regularly occur due to the saturation regions of activation functions. Residual connections are an effective solution to this issue.

2.2 Training Algorithms

2.2.1 Stochastic Gradient Descent (SGD)

SGD is the most popular of the gradient descent methods. It consists of obtaining an estimate of the real gradient by taking the average gradient over a minibatch of size m, and subsequently taking a step on the negative direction of the gradient with a step size defined by η, the learning rate. A complete SGD step is depicted in Algorithm 1.

2.2.2 ADAM

Given that the sensitivity of some directions in the loss landscape is higher, it makes sense to try to incorporate this into the training process. Adaptive learning rate optimizers such as ADAM [65] aim precisely at this, using a separate learning rate for each parameter in a DNN and automatically tuning them throughout the training process.

Algorithm 1: One Stochastic Gradient Descent step

Require: Learning rate η_k
Require: Initial parameter θ
$k \leftarrow 1$
while *stopping criterion not met* **do**
  Sample a minibatch of m examples from the training set $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(m)}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), \boldsymbol{y}^{(i)})$
  Apply update: $\theta \leftarrow \theta - \eta_k \boldsymbol{g}$
  $k \leftarrow k + 1$
end

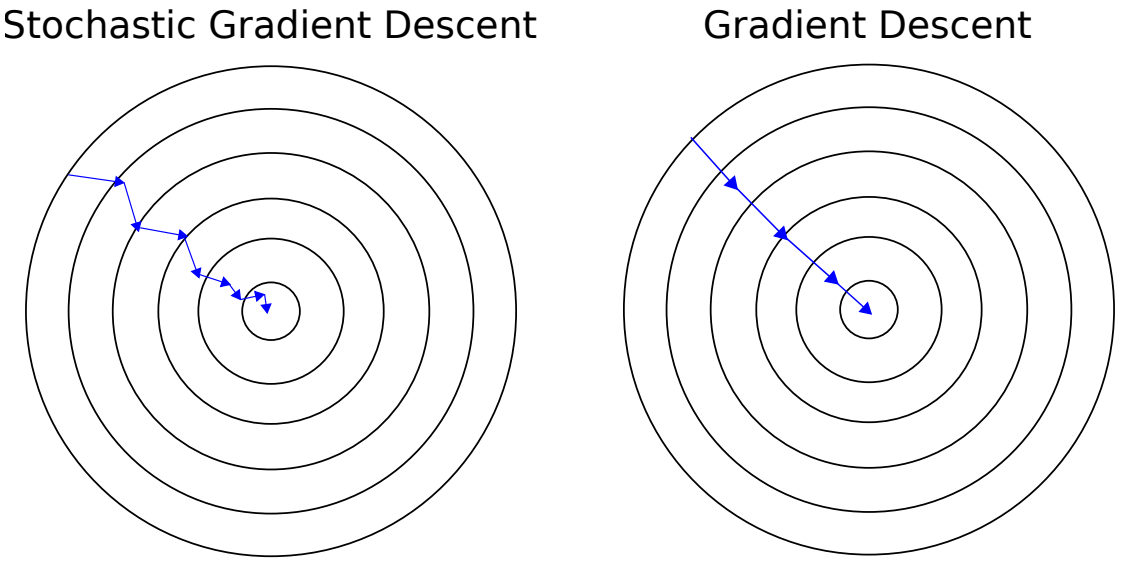


FIGURE 2.1: Illustration of Stochastic Gradient Descent (SGD) vs Gradient Descent. SGD estimates the true gradient from a mini-batch of samples, whereas gradient descent uses the entire training data which might not fit in the memory of the training device.

The name "Adam" derives from "adaptive moments". Adam removes the exponential moving average from the gradients and rescales by the variance. Additionally, it includes a bias correction to the estimates of the first and second moments. Although SGD is generally the choice of the optimizer, ADAM is more commonly used in DNN quantization. We conjecture that the estimate of the gradient, due to the rounding during the forward pass, is too noisy; therefore, ADAM compensates for that noise.

The algorithm for ADAM is depicted in Algorithm 2.

2.2.3 Newton's Method

Newton's method is the most widely used of the second order methods, they make use of second order derivatives. It is based on the second order Taylor series expansion to

Algorithm 2: One ADAM step

Require: Learning rate η_k
Require: Exponential decay rates for momentum estimates, ρ_1 and ρ_2 in [0,1) (defaults: 0.9 and 0.999)
Require: Small constant δ used for numerical stability (default 10^{-8})
Require: Initial parameters θ
Initialize 1st and 2nd moment variables $s = 0, r = 0$ Initialize time step $t = 0$
while *stopping criterion not met* **do**
  Sample a minibatch of m examples from the training set $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(m)}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), \boldsymbol{y}^{(i)})$
  $t \leftarrow t + 1$
  Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
  Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
  Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$
  Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$
  Compute update: $\Delta\theta = \eta \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}}+\delta}$
  Apply update: $\theta \leftarrow \theta + \Delta\theta$
end

approximate $J(\theta)$ near some point θ_0, ignoring derivatives of higher order:

$$J(\theta) \sim J(\theta_0) + (\theta - \theta_0)^\intercal \nabla_\theta J(\theta_0) + \frac{1}{2}(\theta - \theta_0)^\intercal \boldsymbol{H}(\theta - \theta_0), \tag{2.6}$$

where $\boldsymbol{H}$ is the Hessian of J with respect to θ evaluated at θ_0. If we solve for the minimum of this function, we obtain the Newton update rule:

$$\theta* = \theta_0 - \boldsymbol{H}^{-1} \nabla_(\theta_0). \tag{2.7}$$

This method presents different challenges not mentioned here; however, despite these challenges, Newton's method is limited by the significant computational burden it imposes. The number of elements in the Hessian is squared in the number of parameters. Additionally, it involves the operation of inverting a matrix, which is expensive itself. In HAWQ [66], an approximation to the Hessian will be used for a quantization method based on ADAM without introducing additional overhead.

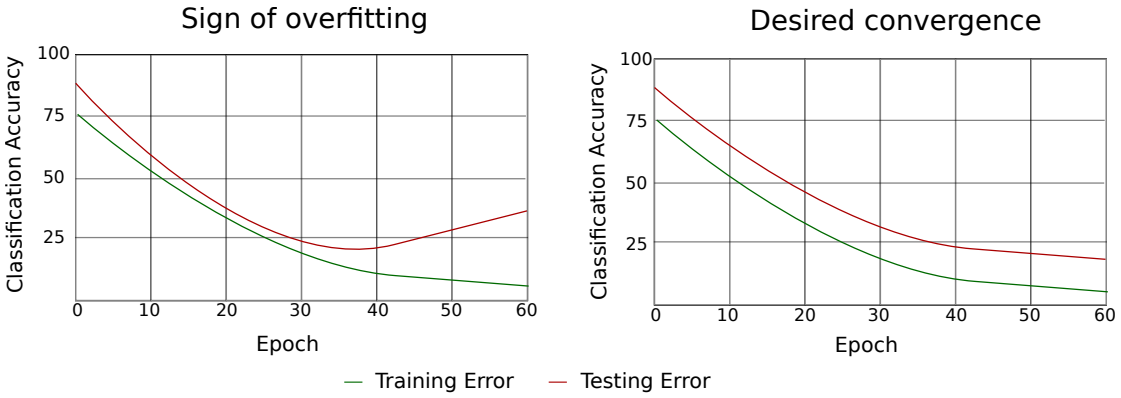


FIGURE 2.2: Signs of overfitting. The plot on the left shows an inflection point where the testing error starts to increase while the training error keeps decreasing. The increased training has caused the model to overfit and perform poorly on the testing set. In contrast, the right plot shows the desired relationship between the training and testing set.

2.3 Regularizing Neural Networks

In DNN optimization, overfitting occurs when a network has enough capacity and rather than learning the underlying patterns, it starts memorizing the training data. It is easy to identify because the performance in the training data will continue to improve while it will get stuck or even get worse on the test data.

It is a central problem in neural networks and several approaches have been proposed in order to address it. The most popular ones include parameters norm penalization, drop-out [67], data augmentation [68], noise injection and stochastic variational reparameterizations [69–71]. One of the major approaches to tackle such problem is by preventing neurons co-adaptation by randomly suppressing activations across the layers of the network with a pre-defined probability. Sparked by Drop-out [67] several variants have been proposed such as DropConnect [72], StochasticDepth [73], DropPath [74] and Dropblock [75]. Drop-out has been modeled as a Gaussian noise injection process during the learning process [76]. Shake-shake regularization [77] followed the same principle of adding noise but with a different approach by randomly weighing paths across the network during training. Batch-norm, originally believed to reduce the internal covariate shift was recently shown to be successful in part due to the same noise-injecting principle in the form of normal additive noise and scaled inverse Chi multiplicative noise which depend on the batch size[78]. Other works include in this line include [79, 80].

Chapter 3

Learning Under Constraints

Learning DNNs with software constraints that translate into savings in hardware and computational resources is the main approach to lightweight DL. Prior to the DL era, numerous approaches dealt with constraints in pure optimization problems. Some of those approaches have been adapted to DL more successfully than others.

3.1 Classical algorithms

3.1.1 Integer Linear Programming

Integer programming is an optimization problem with some of the variables restricted to be integers. If not all the variables are discrete the problem is known as mixed-integer programming. An integer linear program in canonical form is expressed as:

$$\begin{aligned} \textit{maximize} \quad & \boldsymbol{c}^\intercal \boldsymbol{x} \\ \textit{subject to} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \\ & \boldsymbol{x} \geq 0, \\ \textit{and} \quad & \boldsymbol{x} \in \mathbb{Z}^n. \end{aligned}$$

It is commonly used when dealing with problems like budget and resources allocation. Usually each element in $\boldsymbol{x}$ is assigned a profit $\boldsymbol{c}$ and an availability $\boldsymbol{A}$. $\boldsymbol{c}^\intercal \boldsymbol{x}$ is the profit

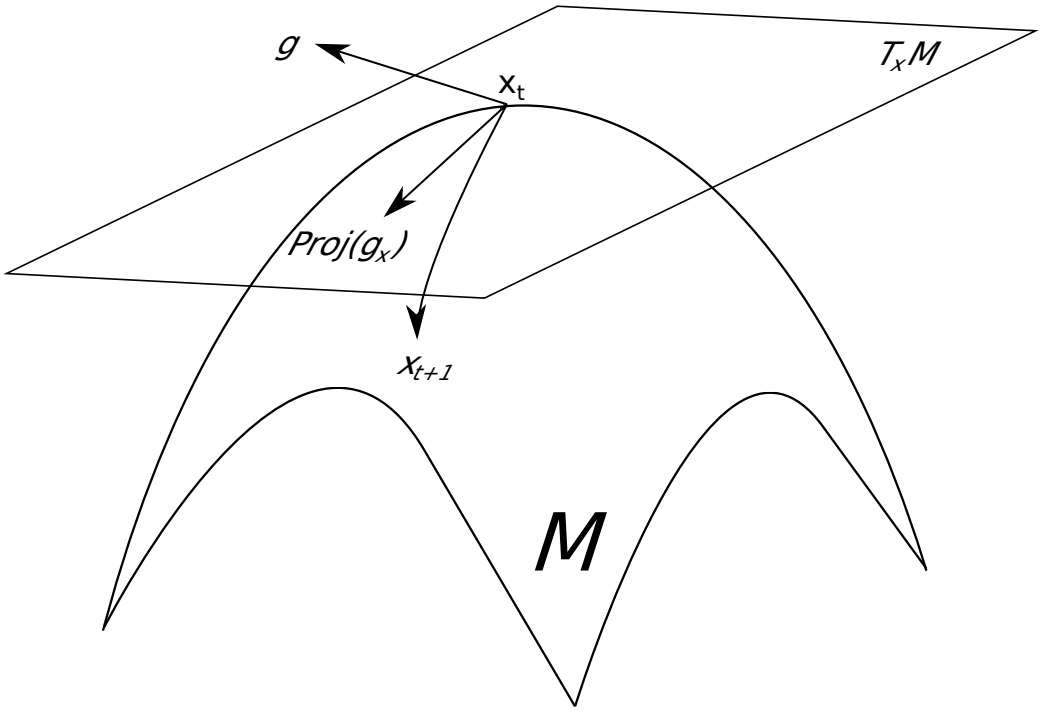


FIGURE 3.1: Optimization on a non-Euclidean manifold. The operations of projection and retraction are illustrated.

we are trying to maximize subject to availability defined by $\boldsymbol{Ax}$. No negative or partial resources can be allocated.

Integer programming has been used in the context of DL in [81] to assign different precision (bitwidth) to each layer in a network. The authors aim at maximizing speedup or saving in power consumption without violating a predefined constraint on network accuracy degradation or compression.

3.1.2 Optimization on Manifolds

Mathematical constructions, such as vectors and tensors, live in an Euclidean space (d-dimensional in this case) generated by its free parameters. Such constructions and parameters can be used to parameterize optimization problems. Occasionally, either the nature of the task or the data at hand imposes constraints on the possible configurations, or solutions space, giving rise to submanifolds embedded inside this Euclidean higher dimensional manifold. The mentioned submanifolds need not be Euclidean; in fact, commonly their structure present negative and positive curvatures. Riemannian manifolds represent a particularly interesting subset of such submanifolds, given that the mathematical tools, namely differential geometry, for traversing them have already

been developed. Thus, providing priors to the optimization framework, in the form of constraints, can considerably reduce the search space to only feasible solutions while preserving a smooth surface. An example is depicted in Figure 3.1.

Geometric optimization relies on the concepts of retraction and vector transport, which are relaxations of the classical geometric concepts of motion along geodesics and parallel transport, in order to solve constrained optimization problems where the solution lies on a manifold.

Consider the problem of optimizing a real-value DNN, but constraining the L_2 norm of the weights to a unit hypersphere $w \subset \mathbb{S}^{d-1}$, or to the more general case of an arbitrary Riemannian manifold $w \subset \boldsymbol{M}^{d-1}$. This is feasible due to the concepts of gradient projection and parameters retraction which will be explained below.

First order optimization methods like vanilla Stochastic Gradient Descent (SGD) are based on the update formulation:

$$x_{t+1} = x_t - \alpha_t g_t \tag{3.1}$$

where $g_t \in \mathbb{R}^n$ is a vector in the loss surface pointing in the slope direction and along with $\alpha_t \in \mathbb{R}$ defines the step size. Analog theory has been developed for optimization on non-linear manifolds defined by constraints:

$$\min_{w \subset \boldsymbol{M}^{d-1}} \boldsymbol{L}(w). \tag{3.2}$$

In order to rely on conventional unconstrained DNN optimization techniques, we consider:

$$\min_{w \in \mathbb{R}^d} \boldsymbol{L}(w) \tag{3.3}$$

and restrict w to lie on the manifold M^{d-1} embedded in the Euclidean space $\mathbb{R}^n$. To do so, the generalization of equation 3.8 to a manifold $\boldsymbol{M}$ consists of a 2 step process. First the gradient g_t is projected to a tangent plane to $\boldsymbol{M}$ at x_t to obtain $\hat{g}_t$. The update step is performed and $x_{t+1/2}$ now lies on this tangent plane; however, has escaped $\boldsymbol{M}$. Second, in order to bring back $x_{t+1/2}$ to the curve, the operation of retraction is performed, which is only valid for elements lying on the mentioned tangent plane, resulting in $x_t + 1$.

Formally, the tangent space at $x_t \in \boldsymbol{M}$, denoted $T_t^x \boldsymbol{M}$ is the linear subspace of $\mathbb{R}^n$ defined by:

$$T_t^x \boldsymbol{M} = \{v \in \mathbb{R}^n : v = f'(0) \text{ for a smooth } f : \mathbb{R} \mapsto \boldsymbol{M} \text{ such that } f(0) = x_t\}. \tag{3.4}$$

With the tangent plane defined, if f is a network on the Euclidean space, then $\hat{f}$, its restriction to $\boldsymbol{M}$, will have gradient:

$$\hat{g}_t = \text{grad}\hat{f}(x_t) = \text{Proj}_t^x(g_t) \tag{3.5}$$

that is, the classical gradient followed by a projection onto the tangent space. Subsequently, an SGD step is taken followed by a retraction mapping, completing the sequence of steps:

$$x_{t+1} = \text{Ret}_x(x_{t+1/2}), \qquad x_{t+1/2} = x_t - \alpha_t \hat{g}_t \tag{3.6}$$

The exponential mapping Exponentials are mappings $\text{Exp}_x : T_x\boldsymbol{M} \mapsto \boldsymbol{M} : \xi \mapsto \text{Exp}_x(\xi)$ that, given a point x on a manifold and a tangent vector ξ at x, generalize the concept of addition. In an Euclidean space, the sum is trivial. On a manifold equipped with a connection, $\text{Exp}_x(\xi)$ is a point on the manifold that can be reached by moving in the direction ξ from x while remaining on the manifold. Furthermore, the trajectory follows a geodesic and the trajectory equals the norm of ξ

We refer the reader to [82] for deeper insights on Riemannian optimization.

3.1.3 Proximal methods

Similar to SGD and Newton's method in smooth optimization problems, proximal algorithms can be viewed as an analogous tool for non-smooth, constrained versions of these problems. Proximal methods are at a higher level of abstraction than classical algorithms: the base operation is evaluating the proximal operator of a function, which itself involves solving a small convex optimization problem. These problems, which generalize projecting a point onto a convex set, often admit closed-form solutions.

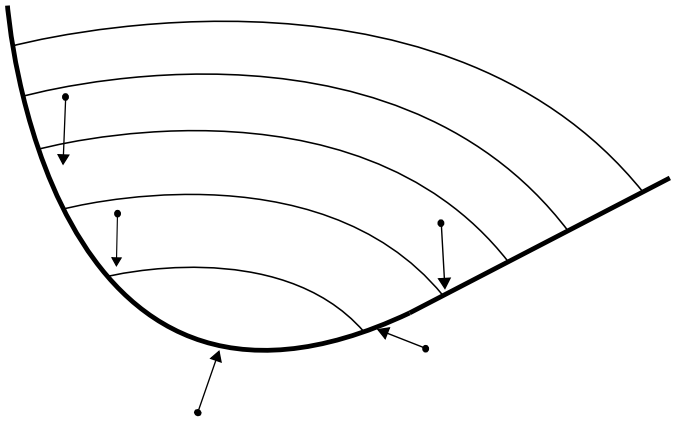


FIGURE 3.2: Evaluating a proximal operator at various points.

Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ be a closed convex function. The proximal operator $\text{prox}_f : \mathbb{R}^n \to \mathbb{R}^n$ of f is defined by:

$$\text{prox}_f(\boldsymbol{v}) = \arg\min_{\boldsymbol{x}}(f(\boldsymbol{x}) + \frac{1}{2}||\boldsymbol{x} - \boldsymbol{v}||_2^2), \tag{3.7}$$

where $||\cdot||_2$ denotes the Euclidean norm. The function minimized on the righthand side is strongly convex and has a unique minimizer for every $\boldsymbol{v}$.

Figure 3.2 depicts a graphical interpretation of the proximal operator. The thick black line denotes the feasible solution space, or the boundary of its domain. Evaluating prox_f minimizes the optimization problem while simultaneously pushing the solution to lie near to the feasible constrained set. A hyperparameter λ can be added to weigh the influence exerted by the constraint penalization factor.

In the context of DNN, proximal operators have been used in NN compression through quantization [83].

For more on proximal methods, we refer the reader to [84].

3.2 Constraints in Deep Learning

In recent years, Deep Neural Networks (DNNs) have driven scientific research in multiple Artificial Intelligence (AI) modalities (*e.g.*, computer vision [6, 85, 86], natural language processing [41, 87] and audio processing). However, DNNs capabilities still

do not translate into end-user applications on resource constrained mobile and embedded devices. Therefore, reducing the gap between experimental and feasible real world implementations is an increasingly active topic of research.

Traditional DNN architectures were designed to improve accuracy in diverse tasks; hence, they are commonly highly overparameterized for the task at hand. For example, ResNet-50 for Imagenet contains 26 million parameters compared with ResNet-18, which has 11 million parameters at an expense of a small accuracy decrease. Additionally, full-precision parameters are not efficient or even required. Given the massive amount of Multiply-Accumulate (MAC) operations performed during dot-product calculations, parameter redundancy and floating-point operations represent the bottleneck toward low latency, memory and power efficient DNNs.

In order to tackle these obstacles, a number of strategies have been proposed which can be grouped in five categories, namely, model compression, quantization, dynamic routing, Neural Architecture Search (NAS) and Automatic Machine Learning (AutoML). Model compression, encompasses the subcategories of network pruning, matrix decomposition and weight-sharing. Quantized Neural Networks (QNN) and their special case, Binarized Neural Networks (BNN), aim at representing DNNs using a limited discrete set of values. Dynamic routing considers the problem of taking decisions on which sections of the algorithm to compute at run-time based on the input data. Neural Architecture Search leverages reinforcement learning to introduce the network design itself into the error-minimization process, it targets at discovering high-performing networks by stacking up DNNs building blocks (e.g. convolutional layers, residual connections, activation functions) in an optimal configuration. Not surprisingly, an optimal architecture for latency might not be optimal for power consumption. AutoML algorithm's goal is to isolate machine learning (ML) practitioners from the design cycle performing end-to-end optimization, ranging from data preprocessing to hyperparameter selection.

In this thesis, 5 complementary contributions were developed in order to further push the field toward achieving feasible embedded DL. Our contributions fall in the broader categories of pruning, quantization, dynamic execution and efficient architectures.

Initial steps in this direction have already been taken. In the recent years, ranging from low rank matrix decompositions [14, 15, 88], pruning [16–18] and sparsification

[19, 19–21], quantization [22–25], architecture search [26–29], efficient submodules [30–33], weight sharing [34] and more, methods for compressing DNNs have appeared in all forms. However, as observed by [13], 90% of papers from the 2018 Annual Meeting of the Association for Computational Linguistics, 80% from the 2018 Conference on Neurological Information Processing Systems (NeurIPS), and 75% from the 2019 Conference on Computer Vision and Pattern Recognition prioritized accuracy over efficiency.

Several approaches have been proposed to speed up neural networks by imposing constraints on the number of parameters, the precision of the parameters and the number of FLOPs. Some are decades-old like pruning [19, 20], while more recent ones include automatic ML (AutoML) such as Neural Architecture Search (NAS) [26, 28, 89, 90]. Below we describe three approaches that are used in the development of this thesis.

3.2.1 Pruning

Pruning consists in removing sections of the networks that are redundant or that have little impact or even harmful impact in the accuracy. Formally, given a NN $f(\theta)$ with parameters θ, the task of pruning consists in finding a binary matrix $\boldsymbol{M} \in \{0,1\}^{|\theta|}$ with the highest number of zero entries that will retain or improve the accuracy of $f(\theta \odot \boldsymbol{W})$. In practice, once the model is deployed the mask should be dropped and the corresponding entries should be removed from θ.

There are two types of pruning: sparsifying [21, 91, 92], which consists in removing individual parameters of the weight matrices, and structured pruning [17, 93], which consists in removing entire filters. Both have advantages and disadvantages:

- Sparsifying relies on sparse structures and sparse operations, which are not commonly supported by hardware platforms. Sparse routines can be implemented by software libraries, nevertheless, only a fraction of the expected improvements are expected. Additionally, even if specialized hardware is built for sparse operations, it still will not necessarily translate into the theoretical expected savings given that the details of the implementations have to be taken into account. Structure pruning, on the other hand, translates directly into savings, both in terms of operations and parameters, without any additional overhead given that entire filters are simply dropped. Nevertheless, Application Specific Integrated Circuits

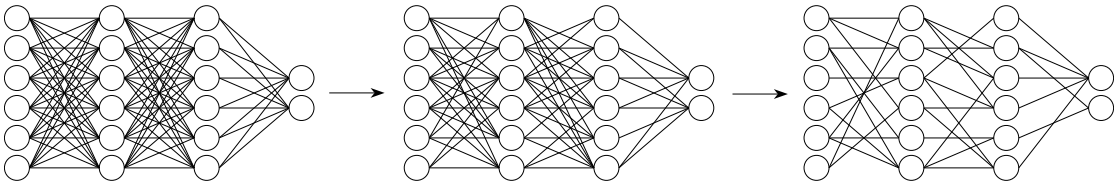


FIGURE 3.3: Stages of sparse pruning

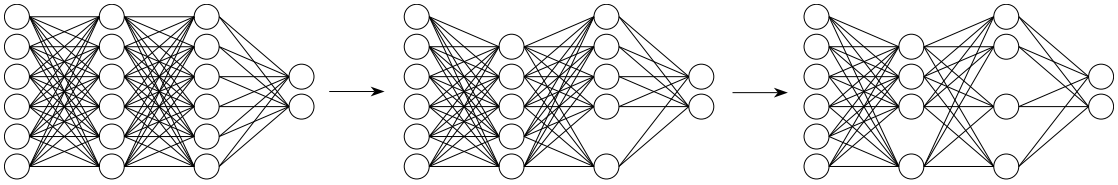


FIGURE 3.4: Stages of structured pruning

(ASICs) built for DL are being designed considering sparsity requirements, and will likely support these structures natively.

- Sparsifying achieves very high levels of compression compared with structured pruning.
- Sparse networks are more robust to noise and interference [94].

Sparse and structured pruning are depicted in Figure 3.3 and Figure 3.4.

State-of-the-art pruning works include [95–97]. A more recent approach consists in pruning at initialization [98]. For an update on the current state of neural pruning see [99].

3.2.2 Quantization

Neural Network quantization consists in reducing the precision (bitwidth) of the parameters and alternatively of the internal representations of the network [100]. In the extreme case, only binary values can be used [22–24], and the multiply-accumulate (MAC) operations can be replaced with efficient logical operations such as XNOR gates. When only reduced bitwidth is used, such as 2 or 4 bits, fixed point multiplication and additions can be performed instead of their floating point counterpart.

The decision of picking fixed-point over floating-point depends on a number of factors. Fixed point store numbers with a fixed number of digits before and after the decimal point. On the other hand, in floating point representations, the decimal point can

move relative to the digits. Thus, floating point representations can support a much wider range of values, with the ability to represent very small numbers and very large ones. This reflects in the rounding and truncating noise, which can be more pronounced in fixed-point representations, as such, floating-point yields higher precision. Another factor is the processor cost. Fixed-point processors are used in a larger number of applications, therefore, making them cheaper than floating-point ones due to the large scale production. Additionally, ease of development is another factor. Floating-point development is usually faster than fixed-point due to corrections required to compensate for quantization noise. Performance is the last of the considerations. As can be observed in Table 1.1, fixed-point representations are more efficient in terms of energy consumption.

Formally, k-bit quantization consists in, given predefined or learned *codebook* $\mathbb{Q} := q_1, ..., q_n$ (the set of permissible values, $|n| = 2^k$), assigning each of the parameters of the network to a member of the codebook:

$$\arg\min_{\theta \in \mathbb{Q}} L(\theta) \tag{3.8}$$

Some codebooks have particular advantages:

1. Binary weights. This codebook has the particularity that multiplications between weights and activations can be replaced by the simple addition of the activations.

2. Binary weights and activations. Similar to the aforementioned case, if the activations are additionally binary, dot products can be performed with logical operations, namely XNOR and POPCOUNT.

3. Binary weights and ternary activations. Ternary networks $\{-1, 0, 1\}$ or $\{-\alpha, 0, \alpha\}$ have the disadvantage that 2 bits are required for storage wasting both VRAM and memory accesses. However, if only the activations are extended to ternary, the overhead is not so terrible plus it offers the advantage that operations can still be performed with boolean logic as investigated by Wan et al. [101].

4. Ternary weights and ternary activations. As mentioned above, these networks require the same storage and memory accesses than a 2-bit model but have lower representation capacity. Nevertheless, they have the advantage that the operations can be executed completely by logical operations. Therefore, Chen et al. [102] took

on the task of operating directly on the codec representation rather than on the codebook values reducing the computational complexity.

5. Powers-of-two. In this codebook, multiplications can be performed by bit shifts.

In practice, a quantized network training procedure involves storing a real-valued version of the network, W. During inference, the real W is quantized using a predetermined pointwise quantization function $\text{Quant}_\text{w} : \mathbb{R} \mapsto \mathbb{Q}_\text{w}$. The weights are updated during the optimization process by estimating the gradients w.r.t. the real-valued copy. Additionally, internal network activations can be optionally quantized with their own bitwidths taken from codebook $\mathbb{Q}_a$ by $\text{Quant}_\text{a} : \mathbb{R} \mapsto \mathbb{Q}_\text{a}$. Given independent bitwidths $bits_w$ and $bits_a$ for the network weights and activations, $|\mathbb{Q}_w| = 2^{bits_w}$ and $|\mathbb{Q}_a| = 2^{bits_a}$. Finally, a quantized convolutional layer with K filters is computed as follows:

$$\mathbf{O}_k = \sum_{i=1}^{I} \text{Quant}_\text{a}(\mathbf{A}_{\text{i},:,:}) * \text{Quant}_\text{w}(\mathbf{F}_{\text{k,i},:,:}), \tag{3.9}$$

where $\mathbf{F}_k \in \mathbb{R}^{I \times h_f \times w_f}$, $\mathbf{F}_k \subset W$ is the k-th convolutional filter. I, h_f and w_f denote the number of input channels, height and width of the filters, respectively. $\mathbf{A}_i \in \mathbb{R}^{h_{in} \times w_{in}}$ and $\mathbf{O}_k \in \mathbb{R}^{h_{out} \times w_{out}}$ denote the input activations and output pre-activations of the filter, where h_{in}, w_{in}, h_{out}, w_{out} represent the height and width of the input and output feature maps, respectively.

This quantization is a hard decision and thus is non-differentiable, and gradients can't be backpropagated through it. However, Bengio et al. [103] proposed a heuristic workaround which empirically has demonstrated to be highly effective. This heuristic consists in defining a quantization function, usually rounding$(\cdot)$, and transporting the gradient unmodified through the non-differentiable quantization function.

The field of quantization is very active, with every other publication closing the gap between quantized and real networks. Recently ReActNet [104], a fully binarized network, managed to surpass 70% accuracy on ImageNet using a Resnet-18 architecture [34], which is comparable to its full precision counterpart.

Figure 3.6 depicts a visual illustration of quantization. A neural layer is quantized with three different bitwidths.

3.2.2.1 Straight-Through gradient Estimator (STE)

Here we elaborate on the aforementioned heuristic. The STE proposed in [103] allows to estimate gradients through the non-differentiable functions round and sign, making the k-bit quantizers employed compatible with the backpropagation algorithm. The STE operates by passing the output gradients unaltered to the input, it is equivalent to the derivative of an identity mapping on the inputs. For an easier understanding of the STE, consider a regular floating point network. If we pay close attention, we can see that although these networks have full precision (32 and 64 bits in today's processors), the precision is not infinite and therefore errors are introduced from truncation and rounding. Nevertheless, when backpropagating gradients through the network, these errors are simply ignored, no compensation of any kind is performed on the gradients. The STE used in quantization is simply magnifying this very same concept to larger rounding errors in few-bit quantization.

In the particular case of activations, the gradients for inputs outside of the range $[-1, 1]$ are suppressed and the derivative of the quantization function becomes equivalent to the derivative of the hardtanh function, denoted *clipped ReLU STE*. Formally the STE is defined as:

$$\frac{d\boldsymbol{L}}{dx} \approx \frac{d\boldsymbol{L}}{dQ} 1_{|x|\leq 1}. \tag{3.10}$$

STE as Lazy projection. As an observation, the STE is equivalent to the dual-averaging method, or lazy projected SGD [105]. In this case, lazy projected SGD proceeds as follows:

$$\hat{\boldsymbol{W}} = Proj_Q(\boldsymbol{W}) = Q(\boldsymbol{W}) \tag{3.11}$$

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta \hat{\nabla} \boldsymbol{L}(\hat{\boldsymbol{W}}) \tag{3.12}$$

For a network with parameters $\boldsymbol{W}$. Written compactly, $\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta \hat{\nabla} \boldsymbol{L}(\hat{\boldsymbol{W}})|_{\boldsymbol{W}=Q(\boldsymbol{W}_t)}$, which we can observe is the straight through estimator.

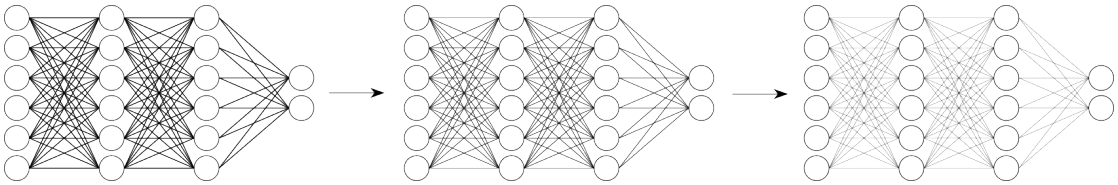


FIGURE 3.5: Neural networks quantization. The figure depicts the same network quantized at three different resolutions. Thinned connections indicate lower bitwidth.

3.2.3 Distillation

Neural network distillation consists in training a small model by using a larger model as a teacher [106]. Colloquially defined as extracting the "dark knowledge" learned by a large capable network by using as guidance signal both the available labels and the features of the teacher network given a provided input. In order to distill a pretrained network, two approaches can be considered: isomorphic weaker architectures or non-isomorphic architectures. Isomorphic architecture approaches commonly simply try to match the intermediate representations, whereas non-isomorphic architectures usually focus on matching the output logits.

Knowledge distillation is performed for two reasons:

1. Expedite the training of small models. Training with additional supervision reduces the required number of training iterations.

2. Train more capable smaller models. Larger models are able to disentangle more complex data patterns and subsequently transfer this learned knowledge to smaller models incapable to figure out complex relationships on their own.

Student capacity. Designing shallow student networks that can learn effectively from deeper teacher networks has been studied by Ba and Caruna [107]. More recently, [108] found that the student model can learn little from teacher supervision when the capacity gap is too large.

Teacher capacity. Regarding the specifications of the teacher networks and training regimes, it is still unclear what the optimal strategy is. See Section 3.2.5 for details relating compute budgets.

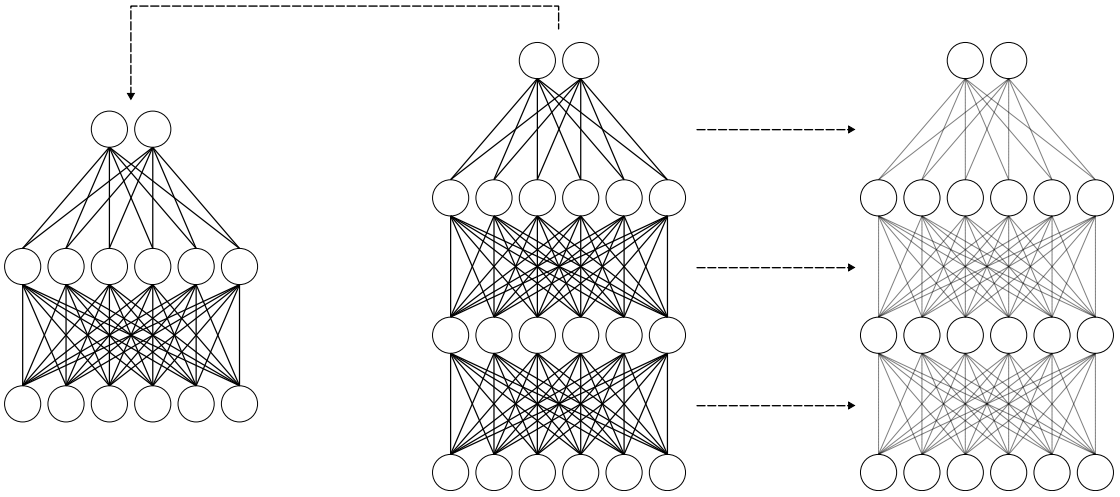


FIGURE 3.6: Left: Distillation of logits in non-isomorphic architectures. Right: Intermediate feature distillation in weaker isomorphic architectures

3.2.4 Efficient architectures

Since the beginning of the DL era, several architectures targeted for image classification have appeared. The first deep architecture was AlexNet [2], it had 61M parameters, it introduced the group convolution in order to split the network in separate GPUs and it achieved remarkable results in the ILSVRC 2012. It was followed by VGG [37] with 138M parameters, the VGG architecture consists of convolutional layers with progressively increasing capacity. Later Inception (GoogLeNet) [109] came out, it used multiple branches with different kernel sizes and 1x1 convolutions and it had 23M parameters. It was followed by ResNet [34] that introduced residual connections, with 25 million parameters, and finally DenseNet [110] with 15M parameters, which extended the idea of residual connections to dense connectivity.

All these architectures were designed with the goal of increasing classification accuracy, disregarding the notion of efficiency. With this target in mind, several works in literature have proposed different neural building blocks. An initial attempt was the depthwise separable convolutions presented in the Inception series [111, 112] and Mobilenets [30, 32, 113], followed by SqueezeNet's fire module [31]. Group convolutions were utilized in ResNeXt [114] along with channel shuffling in ShuffleNet [33, 52]. The linear bottleneck was introduced in MobileNetV2 [32]. Finally, residual connections [6, 115], although initially conceptualized to deal with vanishing gradients, have been utilized as means to obtain increases in performance with little overhead as in the sandglass block [116].

3.2.5 Training budgets.

Regarding the specifications of the teacher networks and training regimes, it is still unclear what the optimal strategy is. It is common knowledge that increasing the capacity of networks allows for disentangling more complex patterns in the data [117]. Yehuda et al. [118] observed in a controlled setting when using the correct number of parameters for a network, it shows linear convergence. However, overparameterization can hurt training by showing exponential convergence [119]. On the other hand, overparameterization can also annihilate some spurious local minima [120]. Therefore, overparameterizing neural networks can have both positive and negative implications when done carelessly.

Jared et al. [121] empirically studied the scaling laws for neural networks performance with respect to model size, dataset size, amount of compute and architecture. They found that there is a power-law relationship between the first three of them and the loss, when none of them is bottlenecked by the others. Conversely, they also found that network structure, namely width or depth, is not a critical factor. They also found that larger-models are more sampling efficient. Therefore, an optimal training routine would involve very large models with relatively modest data. As a result of these findings, the community started training larger and larger models quickly reaching hundreds of billions parameters [122, 123]. Nevertheless, "Chinchilla scaling laws" [117] revisited the problem of optimal parameter/training-tokens allocation and observed that for a fixed training budget, for every increase in parameters, there must be an equal increase in data. The authors were able to outperformed much larger models with their Chinchilla model by simply increasing the dataset.

Sparse Mixture of Experts (SMoEs) . A popular approach for scaling predesigned primitive architecture blocks is to stack multiple of them in parallel with intermediate learnable routing layers [9]. Not to be confused with ensembling which combines independent weak learners with data partitioning methods such as bagging and boosting [124].

SMoEs have surged as a cost-effective solution to scaling commercial large models [125, 126] as they allow to have the capacity of a very large model (commonly 8 times more parameters) but at similar training budgets, effectively offloading compute to the end-user's device. The reason for this is that the whole model can fit inside the VRAM of

large GPUs used for training. Nevertheless, routing layers allow to select sparse experts layers at each stage, rather than executing the whole model, and thus behaving as a small model in terms of computation and convergence. During inference, commonly small devices with limited VRAM, will store the model in the systems' RAM and the selected experts at each stage will have to be transferred to the parallel processor for execution, consequently switching workload from the training's compute budget into inference's communication cost.

3.2.6 Pareto frontier

Pareto efficiency or optimality is a situation where no individual criterion can be better off without making at least another criterion worse. For example, making a neural network more efficient by quantizing it, results in turn in a reduction in accuracy.

Given an initial situation, a Pareto improvement is a new situation where some indicators will improve and no indicator will get worse. Ideally, when a new efficient method is published in the literature, it is expected to push the Pareto frontier. In other words, it is expected to either accomplish better results with the same resources, or the same results with fewer resources.

The Pareto frontier is the set of all Pareto efficient allocations. In the context of efficient neural networks, Pareto frontier graphs are commonly used to represent different trade-offs such as latency vs FLOPs, or parameters vs energy consumption. They are also frequently employed to plot the scaling behavior of neural architectures. This is accomplished by freezing the pre-designed architecture and increasing the number of parameters.

Chapter 4

Automatic Pruning for Quantized Neural Networks

4.1 Introduction

Neural network quantization and pruning, briefly covered in Sections 3.2.1 and 3.2.2, are two techniques commonly used to reduce the computational complexity and memory footprint of these models for deployment. Quantization allows to deploy reduced-precision models on devices manufactured to take advantage of different data-types (e.g. TF32, BFloat16, Int8, Int4), reducing computation as well as storage requirements. Pruning refers to removing redundant modules in hand-crafted architectures to produce a new compact sub-network that is deployed for inference. Despite significant progress has been achieved by the previously described strategies, they are not mutually exclusive and little effort has been done towards combining their strengths. Tung and Mori [127] previously considered this challenge, however, they did it in a fine-grained fashion, this is, by pruning individual connections, also known as sparsification, and subsequently quantizing. To the best of our knowledge, the problem of pruning quantized structures was investigated for the first time by Xu *et al.* [128], where the authors approached it using a learning-based approach. Learning-based approaches commonly operate by partially [21] or completely removing channels during training and progressively identifying the most discriminative ones. However, as observed in [129], removing channels affects the internal statistics of the network and consequently the real impact of removing such

channels can only be indirectly measured in the loss. Thus, rule-based approaches are still actively investigated [17, 18]. In this paper, we focus on the problem of pruning structures for quantized networks using a rule-based approach. Our strategy focuses on removing entire structures (*i.e.* kernels and filters), making networks simpler to implement by avoiding sparse operations. In particular, we propose a Quantized Neural Network (QNN) specific pruning approach based on the geometry of the QNN weight vectors and the interactions between consecutive layers.

In our experiments, we prune redundant structures in already compact models without a high degradation in accuracy. Pruning QNNs can help improve generalization and yield highly light-weight yet accurate DNNs with potential applications on real-time mobile and embedded devices.

Furthermore, we leverage Bayesian optimization to efficiently determine the pruning ratio for each layer. We conduct extensive experiments on CIFAR-10 and ImageNet with various architectures and precisions. In particular, for ResNet-18 on ImageNet, we prune 26.12% of the model size with Binarized Neural Network quantization, achieving a top-1 classification accuracy of 47.32% in a model of 2.47 MB and 59.30% with a 2-bit DoReFa-Net in 4.36 MB.

4.2 Related work

In this section, we cover major generic pruning approaches and explain how they are related to our method or how they could be adapted to QNNs.

Commonly, pruning approaches rely on heuristic metrics to determine the importance of filters based on either the weights magnitude [93], batch normalization scalings [129] or gradients magnitude [98, 130]. Weights-magnitude-based pruning can be implemented either on the underlying real-valued weights or directly in the quantized weights. This method is related to our approach since it takes into account the Euclidean distance of the underlying weights to the quantized ones. The reason being that the underlying weights usually have zero mean normal distributions. Therefore, the Euclidean distance of the weight vectors to the quantized vectors is inversely related to the weight vector norms. In fact, our approach improves over this idea by additionally adding the angle

between these two vectors given that it is a more significant distance metric in the context of dot products.

Pruning based on the magnitude of the batch-normalization scalings as well as on the magnitude of the gradients are compatible with the quantization framework and are used as baselines for comparison Table 4.6.

Other approaches, such as the broader field of AutoML, rely on reinforcement learning to automatically prune channels [131]. However, they will not be covered in this thesis, and we leave it as a possible research direction.

4.3 Method

Given an L-layer QNN, let $\mathbf{F}_k \in \mathbb{R}^{C \times h_f \times w_f}$ be the k-th convolutional filter in the l-th layer, where C, h_f and w_f denote the input channels, height and width of the filter respectively. Let $\mathbf{I} \in \mathbb{R}^{C \times h_{in} \times w_{in}}$ and $\mathbf{O}_k \in \mathbb{R}^{h_{out} \times w_{out}}$ be the input and output feature maps of the filter, where h_{in}, w_{in}, h_{out}, w_{out} represent the height and width of the input and output feature maps, respectively. The output feature map of the k-th filter, $\mathbf{O}_k$, is computed by convolving $\mathbf{I}$ with $\mathbf{F}_k$:

$$\mathbf{O}_k = \sum_{c=1}^{C} \mathbf{I}_{c,:,:} * \mathbf{F}_{c,:,:}. \tag{4.1}$$

Training a QNN involves storing a shadow real-valued copy of the network which is updated during the optimization process by estimating the gradients w.r.t. the real weights. During inference, the real-valued network is quantized using a predetermined pointwise quantization function θ and the full-precision weights can be disposed of. Given a pre-trained QNN, let the set of K real-valued filters for the l-th layer be $\mathbf{W}_L = \{\mathbf{F}_1, ..., \mathbf{F}_K\}$, our goal is to prune non-informative or even harmful filters aiming to save computational resources and reduce latency. In order to choose those filters, we propose a rule-based method to rank them by importance based on the interaction of each filter with the next layer (see Figure 4.2). We will elaborate on how to measure the importance in Sec. 4.3.1. Then we will describe the pruning methods on individual kernels in Sec. 4.3.2 and whole filters in Sec. 4.3.3. Finally, we introduce how to automate the pruning process in Sec. 4.3.4.

4.3.1 Pruning metrics

Common quantization sets, also known as codebooks, do not contain a *zero* value, forcing each real weight to land on an available quantization bin, implicitly injecting noise to the inference process. Intuitively, a real-value weight which lies near any quantization boundary will land on either quantization bin during stochastic quantization with approximately the same probability [22]. Moreover, using deterministic quantization, these weights can easily be flipped into another bin during a gradient descent update step, harming the convergence of the optimization process.

Filters with multiple weights near the quantization boundaries will relate to a higher distance between the real-valued and the quantized versions. Additionally, a scaled version of a filter will have a zero angle w.r.t to the original one, thus a proportional dot-product. In particular, two metrics are considered to measure the distance between the real representation of a vector and the quantized one. A subset of filters is selected in the l-th layer given a quantization function θ, a metric and a threshold th:

$$\hat{\mathbf{W}}_L = \{\mathbf{F}_k | distance(\mathbf{F}_k, \theta(\mathbf{F}_k)) < th\}. \tag{4.2}$$

We re-arrange the k-th filter $\mathbf{F}_k$ into a 1-dimensional vector $\mathbf{v}$. The first ranking metric used is the angle, which can be computed from the cosine distance between the real-valued and the quantized vectors defined as

$$\phi = \frac{\mathbf{v} \cdot \theta(\mathbf{v})}{\|\mathbf{v}\| \cdot \|\theta(\mathbf{v})\|}. \tag{4.3}$$

For the particular binary case it can efficiently computed as $\phi = \frac{\sum_i |v_i|}{\sqrt{\sum_i v_i^2} * \sqrt{n}}$.

Additionally, as second ranking metric, let the Euclidean distance between the two vectors be defined as

$$d = \sqrt{\sum_i (v_i - \theta(v_i))^2}. \tag{4.4}$$

4.3.2 Pruning individual kernels

Every convolutional layer is comprised by K filters with C input channels as shown in Figure 4.2, thus it contains $C \times K$ individual kernels. As an initial experiment, we

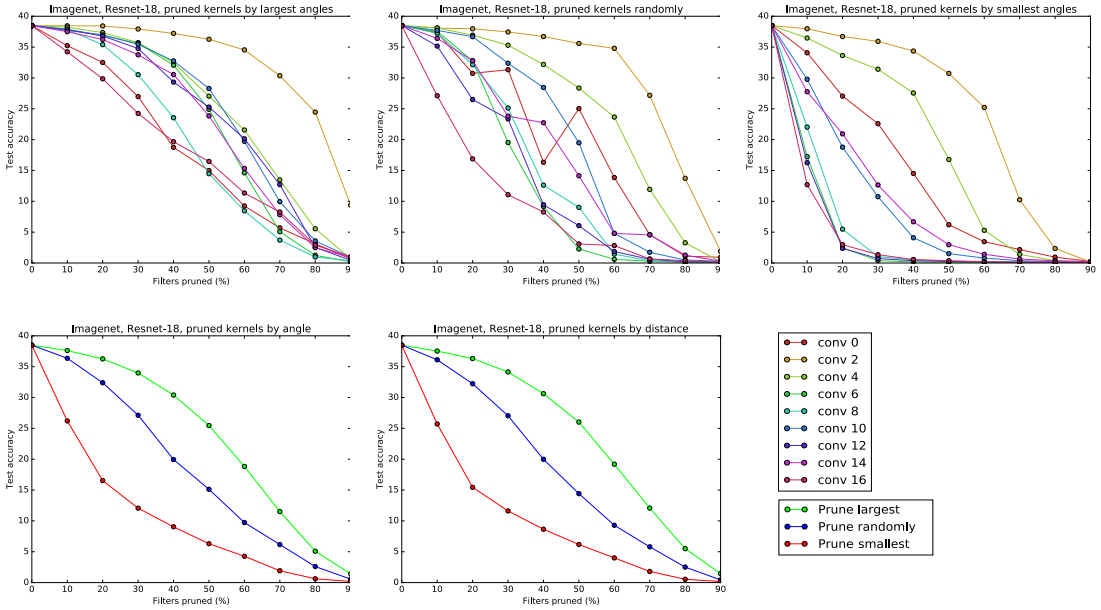


FIGURE 4.1: Pruning individual kernels layer-wise in a BNN ResNet-18 trained on ImageNet. Each layer is pruned at the time, keeping the remaining layers unmodified. (a): Sorting by largest angle first and then pruning. (b): Pruning randomly. (c): Sorting by smallest angle first and then pruning. (d): Accuracy change during pruning averaged across all the layers in order to provide a cleaner visualization. It uses angle distance as metric. (e): Same as (d) but using Euclidean distance as metric.

first rank and prune the individual kernels in the l-th layer computing the distances kernel-wise. In Figure 4.1, kernels were sorted by distance according to equation 6.15 and equation 4.4 and pruned in descendent, random and ascending order, it can be observed that both metrics are useful for ranking the kernels and yield similar results. Additionally, it can also be observed that some layers are more sensitive to pruning than others. Automatically sampling a near optimal pruning threshold will be addressed in Sec. 4.3.4.

4.3.3 Pruning filters

Pruning kernels using the proposed metrics proved to be an effective technique; however, in order to avoid working with sparse matrices, it is desirable to prune whole filters at once, hence we extend the method to filter-level. Unlike kernel-level pruning, filter-level pruning removes the corresponding output feature maps produced by the pruned filters in the l-th layer as well as the related filters in $(l+1)$-th layer. A straightforward way to rank the $\mathbf{F}_k$ filter is to compute filter-wise distances, either by vectorizing the whole filter and computing the distance or by averaging the kernel-wise distances across the

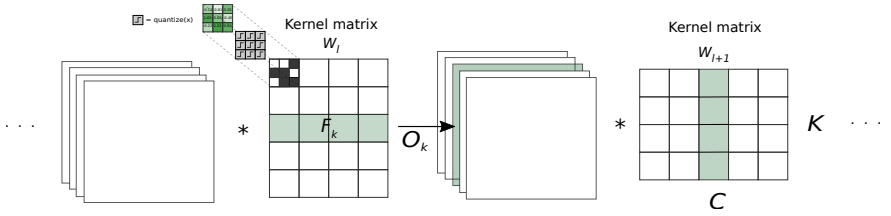


FIGURE 4.2: Kernel pruning during training. The l-th layer is conformed by $K \times C$ kernels. Kernel-wise distances can be averaged across the output channels in the l-th layer or across the input channels in the $(l+1)$-th layer.

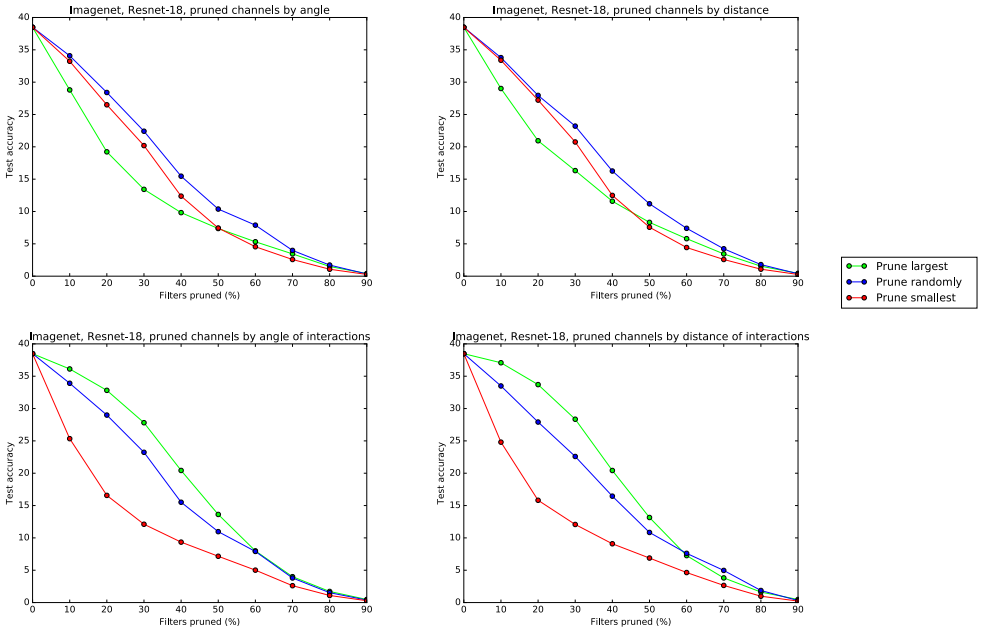


FIGURE 4.3: Pruning complete filters in a BNN ResNet-18 trained on ImageNet. Accuracy during pruning averaged across all the layers using: (a) Angle, (b) Euclidean distance, (c) Angle of the interactions and (d) Euclidean distance of the interactions.

output channel (shaded region in the left side of Figure 4.2). Unexpectedly, this strategy led to poor filter selection as can be observed in Figure 4.3. Alternatively, we analyze the interaction of the l-th layer with the $(l+1)$-th layer. In specific, we rank the $\mathbf{F}_k$ filter in the l-th layer by averaging the kernel-wise distances across the input channels in the $(l+1)$-th layer (shaded region in the right side of Figure 4.2). This method led to a good filter selection criteria. We conjecture that if the feature map $\mathbf{O}_k$ produced by the k-th filter is not a meaningful representation, the related kernels in the next layer will not converge.

4.3.4 Determining per-layer pruning ratio

In order to automatically determine a near optimal pruning ratio for each layer, we leverage Bayesian optimization as proposed by Snoek *et al.* [132], for more on GPs we refer the reader to Rasmussen and Williams [133].

Let the objective function for the l-th layer be defined as

$$y(x) = \mathcal{L}_c(x) + \alpha_1 \mathcal{L}_{params}(x) + \alpha_2 \mathcal{L}_{size}(x), \tag{4.5}$$

where $\mathcal{L}_c$, $\mathcal{L}_{params} = \sum_L |\hat{\mathbf{W}}_L|$ and $\mathcal{L}_{size} = \sum_L [|\hat{\mathbf{W}}_L| \times L_{bit}]$ denote the loss terms incurred by the classification error, number of parameters and network size given all in percentages, and x is the pruning ratio to be estimated. The second and third terms, $\mathcal{L}_{params}$ and $\mathcal{L}_{size}$, are only apparently redundant as will be explained at the end of the section.

Bayesian optimization allows to optimize non-convex, expensive to evaluate, black-box functions by assuming the optimization function was sampled from a Gaussian Process (GP). In this process, multiple pruning ratios are tested for each layer, the evaluation is done in a small subset of the data. In order to pick the pruning ratios to test, an acquisition function can be computed in closed form based on the previous observations.

GPs define distributions over functions of the form $GP : \chi \to \mathbb{R}$. Any given point $x \in \chi$ corresponds to a single Gaussian random variable defined by it's mean and covariance functions:

$$m_x = \mathbb{E}\{x\}. \tag{4.6}$$

$$c_{x_1 x_2} = \mathbb{E}\{(x_1 - m_{x_1})(x_2 - m_{x_2})\}. \tag{4.7}$$

The Gaussian Process Upper Confidence Bound (UCB) will be our choice of acquisition function. Alternative acquisition functions include Probability of Improvement (PI) and Expected Improvement (EI) [132]. We define a set of N pruning ratio observations $X = \{x_n\}_{n=1}^N$, with their respective total loss evaluations $Y = \{y_n\}_{n=1}^N$ as defined by equation 4.5. Finally, the UCB depends on the predictive mean and predictive variance functions:

$$\mu(\hat{x}) = m_{\hat{x}} + c_{\hat{x}X} C_N^{-1} (Y - m_X), \tag{4.8}$$

$$\sigma^2(\hat{x}) = c_{\hat{x}\hat{x}} - c_{\hat{x}X} C_N^{-1} c_{X\hat{x}}, \tag{4.9}$$

where C_N^{-1} denotes the inverse covariance matrix of the observations or *information matrix*, and $\hat{x}$ denotes the unexplored candidates. Finally, the UCB has the form

$$UCB(\hat{x}) = \mu(\hat{x}) - \kappa\sigma(\hat{x}), \tag{4.10}$$

where the parameter κ can be tuned to favor exploration over exploitation. Given χ is a bounded region, the acquisition function determines the next observation to be made

$$\hat{x}_{next} = \arg\max_{\hat{x}} UCB(\hat{x}). \tag{4.11}$$

Given a limited budget of exploration steps, the $\hat{x}$ that provides the minimum loss according to Eq. (4.5), will be used as the pruning ratio for the l-th layer.

Compound loss function in equation 4.5. Conventionally in QNNs, the first and last layers remain with full precision in order to avoid excessive loss of information [24]. In our experiments we noticed the second last layer is highly sensitive to pruning, particularly in Xnor-Net and DoReFa quantization schemes, as shown in Table 4.1, therefore, the third loss term is to encourage more aggressive pruning in the full precision weights (*i.e.*, in the first and second last layers) by taking into account their bitwidths. Removing the second term will cause the classification loss to overwhelm the pruning loss in the intermediate layers and no pruning will occur. Furthermore, the opposite situation will occur in the first and second last layers, where the pruning loss will overwhelm the classification loss degrading the accuracy to an unrecoverable state. In practice, we found $\alpha_2 = \alpha_1/4$ to be a good setting.

Pruning full-precision layers. In order to prune the first layer, we use the angle (or distance accordingly) of the interactions in the second layer. Similarly, the second last layer will always use the sub-optimal metric based on the current layer.

4.4 Experiments

For our experiments we tested 2 different achitectures, VGG [37] and ResNet [6], for the task of image classification on the CIFAR-10 [134] and ImageNet (ILSVRC-2012) [3] datasets with 4 different quantization schemes, BinaryConnect (full-precision activations

TABLE 4.1: Layerwise pruning ratio for Xnor-Net ResNet-18 on Imagenet. Original accuracy: 52.23, Pruned accuracy: 49.48 (⋆ Pruning ratio for downsample residual connection will be the same as the contiguous layer).

Layer	Params	Size(MB)	Ratio(%)	Layer	Params	Size(MB)	Ratio(%)
conv1	9K	0.035	0	res2	32K	0.003	⋆
conv2	36K	0.004	0	conv11	589K	0.070	1.1
conv3	36K	0.004	0	conv12	589K	0.070	25.53
conv4	36K	0.004	0	conv13	589K	0.070	9.13
conv5	36K	0.004	0	conv14	1179K	0.140	26.29
conv6	73K	0.008	0	res3	131K	0.015	⋆
res1	8K	0.001	⋆	conv15	2359K	0.281	1.8
conv7	147K	0.017	0	conv16	2359K	0.281	40.0
conv8	147K	0.017	1.41	conv17	2359K	0.281	0.5
conv9	147K	0.017	1.46	fc	513K	1.956	0
conv10	294K	0.035	0				

TABLE 4.2: CIFAR-10 accuracy on VGG-11.

	Original Acc.(%)	Retrain Acc.(%)	Pruned Ratio(%)	Memory(MB)	Speedup
BinaryConnect	87.60	86.53	34.44	0.75	1.4x
BNN	82.07	82.31	43.82	0.64	1.4x
BNN (all)	84.02	83.09	48.59	0.58	1.5x
Xnor-Net	78.80	74.01	26.41	0.88	1.1x
DoReFa-Net	87.27	86.30	53.73	0.94	1.4x

TABLE 4.3: CIFAR-10 accuracy on ResNet-14.

	Original Acc.(%)	Retrain Acc.(%)	Pruned Ratio(%)	Memory(MB)	Speedup
BinaryConnect	92.73	91.76	21.79	0.43	1.4x
BNN	89.47	88.76	26.79	0.40	1.1x
BNN (all)	88.26	87.58	31.20	0.37	1.4x
Xnor-Net	86.59	83.90	21.61	0.43	1.3x
DoReFa-Net	90.58	89.84	31.34	0.73	1.2x

and binary weights) [22], Binarized Neural Networks (binary activations and binary weights) [23], Xnor-Net (binary scaled activations and binary scaled weights) [24] and DoReFa-Net (2-bit activations and 2-bit weights for our particular experiments) [135].

Our QNNs implementation was done using the Pytorch framework and the Bayesian optimization using [132] public Bayesian optimization package. To implement our algorithm every convolutional, fully-connected (FC) and Batch Normalization (BN) [136] layer is assigned two boolean masks, one is used to index the subset of neurons to compute and a second one to index the subset of inputs to use. Residual connections were pruned accordingly where needed. To pre-train our baselines, we trained from scratch for CIFAR-10. For ImageNet we use a full-precision pre-trained model and fine-tune for

TABLE 4.4: CIFAR-10 accuracy on ResNet-14.

Pruning filters by angle of the filter					
	Original Acc.(%)	Retrain Acc.(%)	Pruned Ratio(%)	Memory(MB)	Speedup
BNN	89.47	87.71	28.17	0.39	1.4x
BNN (all)	88.26	87.89	29.37	0.38	1.4x
DoReFa-Net	90.58	89.23	34.48	0.69	1.3x
Pruning filters by euclidean distance of the filter					
	Original Acc.(%)	Retrain Acc.(%)	Pruned Ratio(%)	Memory(MB)	Speedup
BNN	89.47	87.98	29.02	0.39	1.4x
BNN (all)	88.26	87.70	37.17	0.33	1.4x
DoReFa-Net	90.58	89.40	23.38	0.81	1.1x
Pruning filters by angle of the interactions					
	Original Acc.(%)	Retrain Acc.(%)	Pruned Ratio(%)	Memory(MB)	Speedup
BNN	89.47	88.76	26.79	0.40	1.1x
BNN (all)	88.26	87.58	31.20	0.37	1.4x
DoReFa-Net	90.58	89.84	31.34	0.73	1.2x
Pruning filters by euclidean distance of the interactions					
	Original Acc.(%)	Retrain Acc.(%)	Pruned Ratio(%)	Memory(MB)	Speedup
BNN	89.47	88.50	31.70	0.38	1.4x
BNN (all)	88.26	87.56	40.51	0.32	1.5x
DoReFa-Net	90.58	89.15	22.10	0.83	1.1x

TABLE 4.5: Overall results for ResNet-18 on ImageNet using the angle of the interactions.

	Orig. Acc.(%)	Ret. Acc.(%)	P. Ratio(%)	Memory(MB)	Speedup
BinaryConnect	61.23	58.38	20.66	2.66	1.4x
BNN	51.92	50.10	16.69	2.73	1.4x
BNN (all)	35.80	31.58	10.27	1.29	1x
Xnor-Net	52.23	49.48	10.63	3.01	1.2x
Xnor-Net [128]	49.98	50.13	8.39	3.85	1.09x
DoReFa-Net	60.03	59.30	6.46	4.36	1.4x

TABLE 4.6: Comparison with chosen baseline pruning methods on a quantized ResNet-18 on ImageNet using BinaryConnect quantization.

	Orig. Acc.(%)	Ret. Acc.(%)	P. Ratio(%)
Weights base (Ours)	61.55	58.38	**20.66**
Batch-norm based baseline [129]	61.55	**59.18**	20.11
Gradients based baseline [18]	61.55	58.12	20.36

every specific quantization scheme. We used the standard pre-processing and augmentation as reported by [6]. For training with the pre-trained full-precision model, we use an initial learning rate of 0.005 and decrease it by a factor of 10 every 10 epochs during 60 epochs.

For the pruning strategy, we tested a layer-wise bottom-up and top-down approach with no noticeable difference. Every time a layer is pruned a non-trivial amount (for our experiments we use above 5%), the model is fine-tuned for 5 epochs to allow the weights

to adjust to the new architecture and recover from the accuracy drop. It is worth noting that most of the layers are not pruned, commonly only the later layers will be pruned as can be verified in Table 4.1, which is reasonable since the input channels increase with the depth of the network, therefore the reward for pruning a deeper layer will be larger. It also suggests that filters are more redundant as the depth increases. For fine-tuning we use the same learning rate schedule used during training but decreasing every epoch. Finally, at the end of the pruning process, additional fine-tuning for another 10 epochs is performed and the best validation value is reported. No guidance loss was used in order to isolate the effect of our method.

In our experiments we report the original accuracy, the accuracy after pruning and fine-tuning, the pruned ratio (the pruned ratio refers to model size in bits, not the number of parameters) and average speedup computed from 10 forward passes. To compute the size of the model we used 1 bit per connection for BNNs and 2 bits for DoReFa-Net, 32 bits for biases and other full-precision parameters (*e.g.*, Xnor-Net scalings, BN and PRelu parameters).

All binary network. As mentioned in Sec. 4.3.4, it is common practice in quantized networks to keep full-precision weights in the first and last layers. By quantizing only the weights in the first layer, it will incur into a small accuracy degradation. In the binary case, hence, the dot product can be efficiently computed exclusively with additions. Similarly, in our experiments we quantized the last layer and placed a BN layer in-between this and the softmax layer. Once again, we noticed it did not incur into a large accuracy degradation. By quantizing both the input and last layers, an extremely light and efficient network can be attained with a tolerable accuracy degradation, and it can be further pruned with our proposed method. We refer to this configuration as All weights Binarized Neural Network.

Details on Bayesian optimization. For our layer-wise pruning ratio optimization, we use a bounded region of $[0, 40]\%$ and hyperparameter $\kappa = 2.5$. The process performs 5 observations and 5 exploration steps.

Experiments on CIFAR-10. The CIFAR-10 dataset consists of 50,000 training images plus 10,000 images for validation of size 32×32 distributed across 10 categories for classification. VGG and ResNet architectures were designed for the ImageNet dataset, where commonly the images are cropped to size 224×224, which is considerably larger

than CIFAR-10. Therefore, in order to make the pruning task challenging, we utilized compact versions of both. Our VGG-11 network is identical to the original configuration in [37] with the exception that our implementation only has one FC layer. Similarly, our ResNet-14 implementation is a compact version of the original ResNet-18 [6] where the last stage (a set of 4 *Basicblock* layers) was omitted. The remaining layers were slightly widened by an inflation ratio of 1.25x. For this section we used the pruning loss parameter $\alpha_1 = 1$, except for all BNN where we used $\alpha_1 = 2$. Our resulting VGG-11 network is still highly overparameterized for the task at hand, thus it is more prunable than ResNet-14. As shown in Table 4.2 and Table 4.3, a BNN VGG-11 can be further compressed 43% with no loss in accuracy, however a ResNet-14 BNN can achieve higher accuracy with approximately one third less the size.

In order to compare with different metrics, we performed tests with ResNet-14 using three different quantizations: BNN, full BNN and DoReFa. As reported by Hubara *et al.* [137] and Leroux *et al.* [138], our BNN and Xnor-Net implementations achieved comparable results, frequently BNN outperforming Xnor-Net, therefore we will use BNN quantization for our comparison of the different metrics due to its simplicity and efficiency. As can be observed in Table 4.4 for a small dataset like CIFAR, all the distance metrics performed reasonably well.

Experiments on ImageNet. The ImageNet dataset [3] contains 1.28M labeled images for training and 50K for validation for the task of object classification, spread across 1,000 classes. For the experiments in this section, we used a standard ResNet-18 architecture with no inflation. PRelu layers were placed after the quantized convolutional layers which provided a small increase in accuracy. We pre-train a model using the described procedure and afterwards we proceed to the pruning stage. For this section, we used the pruning loss parameter $\alpha_1 = 8$.

Table 4.5 shows the results of our pruning strategy using the angle of the interactions for ranking. We did not find significant differences when using the angles and the Euclidean distance of the interactions, therefore the decision of using either metric is arbitrary. We compare our method only to [128] on pruning Xnor-Net given that there is no baseline for the rest of the quantization schemes. Xu *et al.* [128] achieved 21.40% parameter pruning corresponding to 8.39% of the total model size with a top-1 accuracy of 50.13%. Our method achieved 25.5% parameter pruning equivalent to 10.63% of the total size

and top-1 accuracy of 49.48%. It is worth noting that our original network is lighter, therefore, although our pruning ratio is only moderately higher, our pruned network is considerably lighter (3.85 MB - 3.01 MB).

We additionally pruned a BNN to achieve an even lighter network and DoReFa-Net to demonstrate the scalability of our method to generic quantization. In our experiments, we noticed that DoReFa-Net is slightly more sensitive to pruning than the other quantization schemes. Finally, we compared against chosen mainstream quantization approaches not tested in the context of quantized networks, and found that batch-norm based approaches perform slightly better than our proposed approach.

4.5 Conclusion

In this chapter, we have proposed to prune quantized neural networks to yield highly compressed yet accurate models for embedded devices. In specific, we have used the shadow full-precision model along with its quantized version in order to accurately approximate full-precision convolutions. To measure the importance of each filter, we have proposed two metrics, including cosine distance and Euclidean distance. During pruning, we leverage the interactions between two consecutive layers. We have conducted extensive experiments on CIFAR-10 and ImageNet classification tasks and observed that conventional quantized architectures can be further pruned with low degradation.

Chapter 5

Switchable Precision Neural Networks

5.1 Introduction

Commonly DNN compression strategies have been exclusively focused on improving the performance and efficiency of static networks. Dynamically routed networks have provided improvements using an alternative approach. By performing computations conditioned on the inputs, the networks are capable of saving resources by executing just the sufficient amount of operations required to map the input to the desired output. Popular strategies include skipping convolutional layers based on the input data complexity [75, 139] and early classifiers [140].

A new approach of instantaneous and on-demand accuracy-efficiency trade-off has been recently explored in the context of neural networks slimming [141]. Following on the spirit of slimmable neural networks, we propose a dynamic quantization strategy, termed Switchable Precision neural Networks (SP-Nets), to train a shared network capable of operating at multiple quantization levels both on their parameters and activations. Our proposed approach aims at providing on-demand computation rather than input dependant. At runtime, the network can adjust its precision on the fly according to instant memory, latency, power consumption and accuracy demands, permitting instantaneous accuracy-efficiency trade-offs. In other words, the user selects the amount of desired

computation resources akin to "power-saving" mode. For example, by fixing the network weights to 1-bit with switchable precision activations, our shared network spans from BinaryConnect to Binarized Neural Network, allowing to perform dot-products using only summations or bit operations, respectively. In addition, a self-distillation scheme is presented in order to increase the performance of the quantized switches. Our proposed approach, SP-Nets yields higher accuracy than individually trained quantized networks on some tasks.

We would like to point out that at the time of development, the tasks of dynamic on-demand quantization had not been approached before. Nevertheless, competing approaches have appeared since then, including Adabits [142], MEBQAT [143] and Quantizable DNNS [144]. In Table 5.4, we compare our approach to those mentioned.

Different precision switches are difficult to optimize as a whole. We summarize the reasons in two parts. On one hand, during training, batch normalization (BN) layers use the current batch statistic to perform intermediate feature maps normalization while estimating the global statistics by accumulating a running mean and running variance, which are used as the replacement during testing. However, the batch statistics of each precision switch are different. As a result, the discrepancy of the features means and variances across different switches leads to inaccurate accumulated statistics in the BN layers. To circumvent this problem, we follow the procedure in [141] by using independent BN parameters for each switch, named *switchable batch normalization (S-BN)*.

On the other hand, we conjecture that simultaneously optimizing multiple quantized switches will reflect in the loss manifold by progressively quantizing the loss surface and the higher precision switches will assist the lower precision ones to achieve a less noisy and smoother convergence, potentially leading them to a better minima. Conversely, the network will converge to only minima which perform well at all the bitwidths potentially harming the performance of some of the switches, particularly the higher precision ones.

During training of SP-Nets, the gradients of each switch are combined before running an optimizer step. However, there is no explicit mechanism impeding the individual switches from moving in distinct directions. In order to encourage the different switches to move in approximately the same direction, we adapt a self-distillation strategy, where

the full precision switch provides a guiding signal for the rest of the switches. Specifically, only the teacher full-precision switch sees the ground-truth while the student low-precision switches are trained by distilling knowledge from the full-precision teacher.

In order to increase the flexibility of our model, in addition to switchable precision representations, we equip our network with slimming (width switchable) capability.

We demonstrate the performance of SP-Nets against independently trained quantized models in the tasks of image classification and semantic segmentation on the ImageNet and Pascal VOC datasets using ResNet and MobileNet architectures.

5.2 Preliminaries

The optimization problem of traditional Quantized Neural Networks (QNNs) aims at minimizing an objective function $\boldsymbol{L}$ given a set of trainable weights W, which take bitwidth values from C_w, a predefined discrete set typically referred as *codebook*. A common QNN training procedure involves storing a real-valued version of W. During inference, the real W is quantized using a predetermined pointwise quantization function $\mathrm{Quant_w} : \mathbb{R} \mapsto \mathrm{C_w}$. The weights are updated during the optimization process by estimating the gradients w.r.t. the real-valued copy. Additionally, internal network activations can be optionally quantized with their own bitwidths taken from codebook C_a by $\mathrm{Quant_a} : \mathbb{R} \mapsto \mathrm{C_a}$. Given independent bitwidths $bits_w$ and $bits_a$ for the network weights and activations, $|C_w| = 2^{bits_w}$ and $|C_a| = 2^{bits_a}$. Finally, a quantized convolutional layer with K filters is computed as follows:

$$\mathbf{O}_k = \sum_{i=1}^{I} \mathrm{Quant_a}(\mathbf{A}_{\mathrm{i},:,:}) * \mathrm{Quant_w}(\mathbf{F}_{\mathrm{k,i},:,:}), \tag{5.1}$$

where $\mathbf{F}_k \in \mathbb{R}^{I \times h_f \times w_f}$, $\mathbf{F}_k \subset W$ is the k-th convolutional filter. I, h_f and w_f denote the number of input channels, height and width of the filters, respectively. $\mathbf{A}_i \in \mathbb{R}^{h_{in} \times w_{in}}$ and $\mathbf{O}_k \in \mathbb{R}^{h_{out} \times w_{out}}$ denote the input activations and output pre-activations of the filter, where h_{in}, w_{in}, h_{out}, w_{out} represent the height and width of the input and output feature maps, respectively.

In the context of QNNs, multiple quantizers and gradient estimators have been proposed in the literature [103, 135, 145–147]. For our SP-Net, we stick to common ones, which

will be described in Sec. 3.2.2.1. The motivation to use each quantizer will be explained in each corresponding subsection.

5.2.1 Quantizers

The k-bit quantizers described in the next sections share the same base quantization function, $Q(\cdot)$:

$$Q(x) = \frac{1}{2^k - 1}\text{round}((2^k - 1)x). \tag{5.2}$$

During backpropagation, we use the STE:

$$\frac{d\boldsymbol{L}}{dx} \approx \frac{d\boldsymbol{L}}{dQ}1_{|x|\leq 1}. \tag{5.3}$$

5.2.1.1 Tanh-Based Quantizer.

DoReFa and half-wave Gaussian quantization [135, 146] proposed to use different quantizers for weights and activations. Weight quantizers approximate the hyperbolic tangent function (tanh), constraining the weights to $[-1, 1]$. However, tanh is associated with the vanishing gradient problem, thus, it is attractive for activations quantizers to approximate the popular ReLU activation function as described in the next section, constraining the activations to the positive range.

In DoReFa [135], a $\tanh(\cdot)$ function is first used to project the input to range $[-1, 1]$ in order to reduce the impact of large values. The quantizer is defined as follows:

$$\text{TanhQuant}(x) = 2 \cdot Q(\frac{\tanh(x)}{2 \cdot \max(|\tanh(x)|)} + \frac{1}{2}) - 1, \tag{5.4}$$

where $Q(\cdot)$ hereafter is defined in Eq. (6.5).

It is appealing to train a network with activations switchable down to 1 bit with permissible values $\{-1, 1\}$ since convolutions can be performed using xnor and popcount operations assuming that weights are 1-bit as well. Therefore, when training these type of networks, TanhQuant quantizer is used on both weights and activations to allow the activations of the intermediate switches to lie in the range $[-1, 1]$. For this same reason, the layers are re-ordered as described in Xnor-Net [24] (typical

layers ordering is QuantConv→BN →ReLU→QuantConv, while layers re-ordered are QuantConv→ReLU→BN→QuantConv).

5.2.1.2 ReLU-Based Quantizer.

As mentioned in the previous section, [135, 146] employ a ReLU approximation quantizer for the activations with no layer re-ordering. The method proposed by DoReFa [135] will be employed here, which consists of simply clipping the activations followed by the base quantizer:

$$\text{ReLUQuant}(x) = Q(\text{clip}(x, 0, 1)). \tag{5.5}$$

The ReLU-based quantizer will be used in our SP-Net activations whenever a network is not required to have activations switchable down to 1 bit.

In particular, [146] proposed both uniform and non-uniform spacing between the codebook elements. They first clipped the activations to the range $[0, v]$, where v is some predetermined value, and the codebook for both cases, uniform and non-uniform, is obtained from the network internal statistics. In our tests with ReLUQuant quantization, we simply constrain the activations to the range $[0, 1]$ with uniform quantization. In the logarithmic quantizer described in the next section, non-uniform quantization is used.

5.2.1.3 Logarithmic Quantizer

In full-precision neural networks, the weights and activations have non-uniform distributions [147]. Taking advantage of that fact, the authors used logarithmic representations, both in weights and activations, achieving higher classification accuracy at the same resolution than uniform quantization schemes at the expense of higher implementation and computation complexity. In the original paper, their LogQuant layer contains a global Full Scale Range (FSR) parameter which is reported in the paper. Additionally, each layer has its own FSR. In our SP-Net, we use a variation of their LogQuant layer in order to avoid the FSR parameter. Our modified LogQuant quantizer is defined as follows:

$$\text{LogQuant}(x) = \begin{cases} 0 & \text{if } x = 0 \\ 2^{\widehat{x}} \cdot x & \text{otherwise} \end{cases}, \tag{5.6}$$

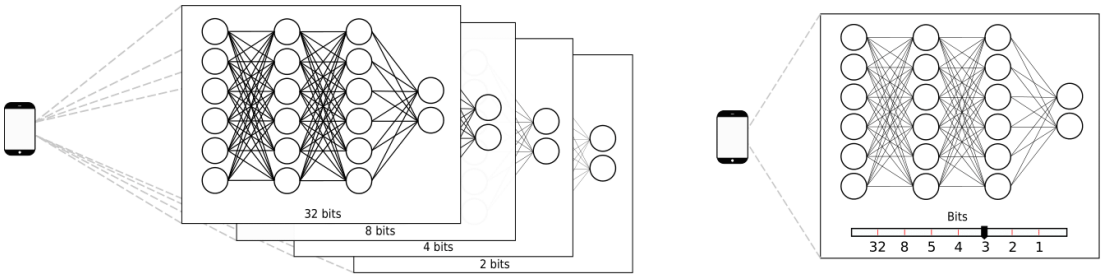


FIGURE 5.1: Overview of the proposed approach. Left: Traditional QNNs are trained to operate at an arbitrary predefined bitwidth. Right: In SP-Nets multiple precision switches share a common architecture, making it capable of adjusting the precision of their representations on the fly, granting devices and end-users real-time control over the network performance.

where

$$\hat{x} = \text{rescale}(Q(\text{normalize}(|\log(x)|))), \tag{5.7}$$

$$\text{normalize}(x) = (x - \min(x))/(\max(x) - \min(x)), \tag{5.8}$$

$$\text{rescale}(x) = x \cdot (\max(x) - \min(x)) + \min(x), \tag{5.9}$$

$$s(x) = \text{sign}(x). \tag{5.10}$$

Similarly to Sec. 5.2.1.1, two-sided logarithmic quantization (positive and negative) can be used in order to have activations switchable down to 1-bit. Depending on the choice of activations quantization (one-sided or two-sided), layer re-ordering should be taken into account. In our experiments, logarithm base-2 was used, however, base-$\sqrt{2}$ could provide higher accuracy.

5.3 Method

SP-Nets generalize QNNs in the sense that the learnable weights W are simultaneously optimized for multiple codebooks $C_w[n]$ and $C_a[n]$ of variable cardinality, where $n = 1, ..., N$. The permissible values of the N codebooks are determined by the choice of quantizers Quant_w and Quant_a, while their cardinality is determined by the bitwidths $bits_w[n]$ and $bits_a[n]$.

The training procedure for SP-Nets proceeds as follows: The first tuple of codebooks $(C_w[0], C_a[0])$ is selected as the active switch and a forward and backward pass of the

network is executed. The gradients through the non-differentiable quantization function are estimated using the STE defined in section 7.2. The gradients are stored and subsequently the following switch is selected with two new codebooks. The steps are repeated until the N codebooks have been exhausted and the new gradients are aggregated to the previously estimated gradients at every iteration, $\nabla W = \sum_{i=1}^{N} \nabla W_i$. Finally, an optimizer step is performed using the total accumulated gradient.

From a geometric perspective, this is equivalent to moving in the average direction of the gradient vectors of all the switches present in the shared network.

Nevertheless, the aforementioned procedure performs poorly in practice as empirically demonstrated in section 5.4.4. Therefore, we resort to S-BN [141], a technique used to train slimmable neural networks, as described in detail in section 5.3.1. In section 5.3.2, we extend our SP-Net to mixed slimmable SP-Net. Finally in section 5.3.3 we adapt a self-distillation technique to improve the performance of our approach.

5.3.1 Switchable Batch Normalization for Dynamic Quantization

DNNs at their current state are not naturally dynamically switchable in precision due to the inconsistent behavior of batch normalization layers during training and inference. During training, BN layers utilize the current batch x_b mean $\mathrm{E}_b[x_b]$ and variance $\mathrm{Var}_b[x_b]$ to perform intermediate feature maps normalization while accumulating a running mean u and running variance σ^2, which is used as replacement during test.

In a naive SP-Net implementation, the mini-batch internal statistics of each quantization switch will be different, however, all the switches will contribute to the accumulated global mean u and variance σ^2, thus, enabling the network to train properly, but resulting in inaccurate test inference.

In order to address this issue, we resort to Switchable Batch Normalization (S-BN) layers [141], which equip regular BN layers with private $u[n]$, $\sigma^2[n]$, $\beta[n]$ and $\gamma[n]$ for each of the N switches. This allows the internal representations to have different distributions depending on the active bitwidths. The overhead of the additional private parameters is negligible since BN parameters account for a minimal portion of the total amount of parameters. It is also negligible in terms of run-time complexity since they involve no additional operations. Additionally, BN layers count with two learnable parameters β

and γ used to provide the layer with the capacity of performing an identity mapping of the input. Unlike u and σ^2, β and γ can be updated by all the switches. Providing with private versions of them is not as crucial, however doing so yields an additional accuracy boost [148]. Furthermore, they generate no additional overhead since they can be merged with $u[n]$ and $\sigma^2[n]$ after training.

In Algorithm 3, we illustrate the use of S-BN in one SP-Net training iteration.

Algorithm 3: One SP-Net iteration using S-BN

Require: SP-Net model M.
Require: Lists of codebooks for weights and activations C_w, C_a.
Get mini-batch data x and ground-truth y;
for *c_w in C_w* **do**
  for *c_a in C_a* **do**
    Set c_w, c_a in M as the active switch;
    Switch S-BN layer for current switch;
    Forward pass using input $M(x)$;
    Compute the loss w.r.t. the output y;
    Compute parameters gradients using the STE;
  end
end
Update weights using accumulated gradients;

5.3.2 Slimmable SP-Net

Network slimming [141] relies on S-BN in order to allow each layer of the network to operate at different widths. Given the current width multiplier $width \in [0, 1]$, a slimmable convolutional layer with K filters computes only the first $\lceil K * width \rceil$ ones.

Network slimming and quantization are complementary techniques and effortlessly work along without technical complications. Therefore, we can train a single shared network with switchable width and precision to increase the flexibility. A single slimmable/SP network can be trained in the cloud and distribute particular switches to different deployment systems based on their specific hardware capabilities, where they can be further slimmed and quantized instantaneously on demand. In Sec. 5.4.2, we provide a comparison of a slimmable SP-Net with the corresponding individually trained switches.

Although the benefits of a SP-Net in terms of power and speed improvements are evident by performing dot-product operations on quantized vectors, the benefits on memory savings may not be so apparent, given that the full precision weights must be stored on

non-volatile memory at all time regardless of the active quantization switch. However, during operation, a network clone is stored in the RAM of the processor in order to provide quick access, therefore weight quantization only takes place once every time a new switch is requested and the quantized clone is kept in volatile memory. By quantizing activations, additional RAM savings can be obtained. In other words, there are no savings in memory footprint for storage but there are in RAM during operation.

5.3.3 Self-Distillation

Knowledge distillation is a common strategy used to provide a stronger training signal, typically from a large network to a smaller one in a teacher-student scheme. In a similar fashion to knowledge distillation, by simultaneously optimizing multiple quantized switches, implicitly the high precision switches are providing guidance to the noisier low precision updates. However, this behavior is not explicitly encouraged. Therefore, in this section we present a distillation mechanism denominated *self-distillation*, simultaneously developed by Zhang et al. [108], where only the full-precision switch sees the ground truth, while the low bitwidth switches try to match the internal representation as well as the output distribution of the full-precision switch. The strategy formulated allows the full-precision switch to guide the optimization process with a significant amount of information flow across all the switches.

In US-Net [148] gradients are prevented from flowing from the sub-networks to the largest width switch. Formally, to mimic the outputs, similarly to US-Net, we use the Kullback–Leibler divergence as distance measure on the output distributions p_r and p_q of the full-precision switch and the quantized active switch. Let $SG(\cdot)$ denote the stop-gradient function, the output mimic loss is:

$$\boldsymbol{L}_{out} = D_{KL}(SG(p_r)\|p_q). \tag{5.11}$$

Additionally, to create the guidance signal, Zhuang *et al.* [149] proposes a hint-based training strategy by comparing the intermediate feature maps between the full-precision teacher and the low-precision student. Similarly, let f_r and f_q denote the internal feature maps (i.e., pre-activations) of the full-precision switch and the quantized active switch,

the guidance loss for internal representations becomes:

$$\boldsymbol{L}_f = \|f_r - f_q\|_2^2. \tag{5.12}$$

In this case, we do not stop the gradients flowing to f_r and quantize f_r.

Finally, the loss for the quantized switches is their aggregation, while for the full precision switch is simply the regular cross-entropy classification loss:

$$\boldsymbol{L}_q = \alpha_1 \boldsymbol{L}_{out} + \alpha_2 \boldsymbol{L}_f, \tag{5.13}$$

$$\boldsymbol{L}_r = \boldsymbol{L}_{cross-entropy}(p_r, y), \tag{5.14}$$

where y denotes the ground-truth labels, and α_1 and α_2 are the scaling coefficients to balance $\boldsymbol{L}_{out}$ and $\boldsymbol{L}_f$, respectively.

5.4 Experiments

In this section, we test 2 different architectures, ResNet [6] and MobileNet [30], for the task of image classification on the ImageNet (ILSVRC-2012) [3] and Tiny ImageNet datasets with 2 different quantizers: Tanh-based quantization [135], ReLU-based quantization [135, 146].

We implement our SP-Nets using Pytorch. For our ImageNet experiments, we use a regular (*i.e.*, 1 single switch) full-precision pre-trained model taken from the official Pytorch zoo, we then replicate the BN parameters across all the S-BN switches and fine-tune the SP-Net model. We use the standard pre-processing and augmentation as reported by [6]. For training with the pre-trained full-precision model, we use Adam [65] optimizer with an initial learning rate of $1e^{-4}$ and decrease it by a factor of 10 at epochs 15 and 20 with a total of 25 epochs. For our Tiny ImageNet experiments, all the networks are learned from scratch using SGD optimizer with an initial learning rate of 0.1 and decreased by a factor of 10 every 30 epochs during 100 epochs. Batch size of 128 was used for all experiments. As in common practice, the first and last layers are not quantized [24].

The independent (multiple or not shared) quantized networks were fine-tuned from the original full precision pretrained network using the same quantization strategy.

5.4.1 Evaluation on ImageNet

Evaluation on ResNet-18. In Table 5.1, we compare the Top-1 accuracy for different bitwidths in both weights and activations for a ResNet-18 architecture. We combined 3 C_w and 3 C_a codebooks to form 9 different switches. Our SP-net achieved very close accuracy to the independently trained QNNs with the full-precision switch being the most affected, while providing increased efficiency-accuracy flexibility.

TABLE 5.1: Top-1 accuracy (%) for ResNet-18 on ImageNet. A single SP-Net was trained for all the combinations

Weights bitwidth	Activations bitwidth	SP-Net (single)	Independent (multiple)
2	2	62.4	**62.6**
2	4	64.6	**64.9**
2	32	65.3	**65.8**
3	2	63.9	**64.0**
3	4	66.3	**66.5**
3	32	66.8	**67.4**
32	2	65.1	**65.3**
32	4	67.0	**67.3**
32	32	68.1	**69.5**

We additionally investigate the effect of the bitwidth gap in SP-Nets. In Table 5.2, we present results with a shorter gap, using bitwidths of a more practical scenario. In essence, we repeated the previous experiment but we did not fine-tuned the full precision switch. Consistent with the aforementioned experiment, our SP-net achieved very similar results to independently trained QNNs, and the bitwidth gap did not have an impact in the accuracy.

TABLE 5.2: Top-1 accuracy (%) with a short gap between switches for Resnet-18 on ImageNet.

Weight	Activation	SP-Net (shared)	Independent (multiple)
2	2	62.5	**62.6**
2	4	64.6	**64.9**
3	2	63.9	**64.0**
3	4	66.3	**66.5**

Evaluation on MobileNet. The MobileNet architecture makes use of the depthwise separable and pointwise convolutions without residual connections drastically reducing the number of parameters without a major decrease in accuracy. However, it is very

sensitive to quantization, especially on the activations. Sheng *et al.* [150] re-designs the MobileNet architecture to be more quantization friendly. For our MobileNet SP-Net, we follow their architecture. The re-design involves simple layers replacement and re-ordering. In Table 5.3, we report the results of different bitwidths for weights and activations using our SP-Net MobileNet and independent networks with the modified architecture. As can be observed, the impact of switchable precisions on MobileNet is higher than on ResNet-18.

TABLE 5.3: Top-1 accuracy (%) for MobileNet on ImageNet.

Weights bitwidth	Activations bitwidth	SP-Net (single)	Independent (multiple)
8	8	59.5	**63.5**
8	32	66.0	**69.0**
32	8	59.6	**63.6**
32	32	66.1	**69.7**

Comparison with recent related approaches. At the time of production of our work, there were not related approaches. However, similar works have appeared since then. Therefore, here we attempt to compare them. Both Adabits [142] and Quantizable DNNs [144] present a very similar core methodology to ours with slight variations. Adabits additionally proposes a specific quantization to avoid storing the 32 bit full precision network, and instead store only the largest switch. Similar to ours, Quantizable DNNs allows to quantize weights all the way down to 1 bit. Quantizable DNNs, does not provide accuracy results with activations quantization. Finally, unlike both works, we presented a self-distillation strategy to improve the accuracy of the lower bitwidths.

TABLE 5.4: Comparison with related approaches

Method	Architecture	Weights bitwidths	Activations bitwidths	Acc(%)
SP-Net (ours)	ResNet-18	[32, 3, 2]	[32, 32, 32]]	[69.5, 67.4, 65.8]
Quantizable DNN [144]		[4,3,2]	[32, 32, 32]	[66.9, 66.2, 62.9]
SP-Net (ours)	MobileNetV1	[32, 8]	[32, 32]	[69.7, 69.0]
Adabits [142]		[8, 6, 5]	[32, 32, 32]	[72.4, 72.4, 72.1]

5.4.2 Evaluation on Tiny ImageNet

The Tiny ImageNet dataset is a downsampled ImageNet version (64×64) with 100K images for training and 10K for validation, spread across 200 classes. For the experiments on Tiny ImageNet, we used the same ResNet-18 architecture as that for ImageNet except for the first layer, whose filter size is $3 \times 3 \times 64$ with $stride = 1$ and $padding = 1$.

TABLE 5.5: Tiny ImageNet accuracy for a 1-bit SP-Net ResNet-18.

Weights bitwidth	Activations bitwidth	SP-Net (single)	Independent (multiple)
1	1	**43.5**	40.0
1	3	**48.8**	46.3
1	8	**48.8**	47.0
1	32	49.0	**49.1**

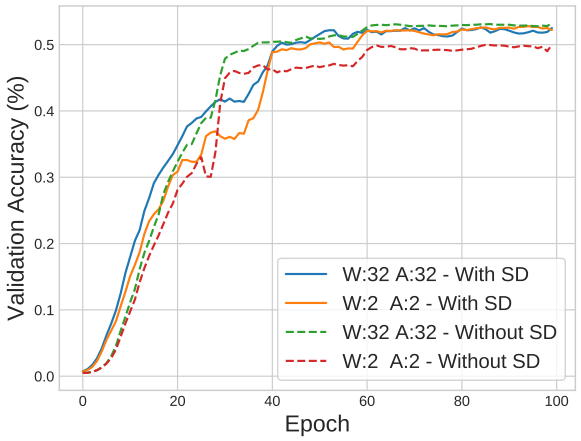


FIGURE 5.2: Tiny-ImageNet accuracy for SP-Net with and without self-distillation

Comparison of Different Quantizers.

TABLE 5.6: Comparison of different quantizers on a SP-Net for ResNet-18 on Tiny ImageNet. The networks were slimmed with a factor of 0.25 to magnify the effect of the quantizers. [1] ReLU-based: TanhQuant quantizer on weights, ReLUQuant on activations. [2] Tanh-based: TanhQuant quantizer on both weights and activations *with layer re-ordering*. [3]Logarithmic 1: TanhQuant quantizer on weights and LogQuant quantizer on activations. [4]Logarithmic 2: LogQuant quantizer on both on weights and activations.

Width	Bitwidth		Quantizer Accuracy(%)			
	Weight	Activation	ReLU-based[1]	Tanh-based[2]	Logarithmic 1[3]	Logarithmic 2[4]
0.25	2	2	**32.5**	29.9	29.1	28.9
0.25	2	4	**34.5**	32.7	34.0	33.9
0.25	2	8	**34.8**	32.8	34.1	34.1
0.25	2	32	34.3	32.8	34.4	**35.5**
0.25	4	2	**34.1**	33.7	33.3	32.5
0.25	4	4	**37.4**	36.8	37.3	36.7
0.25	4	8	37.4	36.6	**37.5**	37.0
0.25	4	32	37.5	36.6	37.6	**37.7**
0.25	8	2	**34.4**	33.0	33.9	32.6
0.25	8	4	**37.3**	36.5	37.2	36.4
0.25	8	8	37.3	36.9	**37.4**	36.9
0.25	8	32	37.5	37.0	37.6	**37.7**
0.25	32	2	**36.3**	33.6	33.5	32.8
0.25	32	4	**37.3**	36.5	37.2	36.8
0.25	32	8	37.2	37.0	**37.4**	36.9
0.25	32	32	**37.7**	37.0	37.5	37.5

As explained in Sec. 3.2.2.1, different quantizers can be used in different scenarios. ReLUQuant quantizer is used in activations in general. When it is desirable to have activations quatizable to 1-bit, TanhQuant quantizer is used. LogQuant quantizer is chosen when higher accuracy is desired with the same bitwidth, at an expense of higher complexity.

In Table 5.6, 4 quantizer configurations were tested for SP-Nets with multiple weight and activation switches. In our results, Tanh-based configuration obtained the lowest accuracy across all the switches and ReLU-based configuration frequently obtained the best results. Logarithmic based configurations occasionally performed better, particularly at higher activations bitwidths. Logarithmic based configurations were expected to produce the best results, however our FSR-free modified version and logarithm base choice may have harmed their performance. All the networks were slimmed by a factor of 0.25 to magnify the effect of the quantizers.

1-bit SP-Net. We trained a network with a single bit per weight and switchable precision activations. Our network spans from the popular BinaryConnect, where dot products are computed using only summation, to Binarized Neural Networks (BNNs), where activations are quantized to 1-bit and dot products are computed using xnor and popcount operations.

Training a network with precision switchable down to 1-bit per activation is particularly challenging since it causes a significant decrease in performance across all the bitwidths when using a ReLU-based quantization. Therefore, networks with activations switchable to 1-bit are trained with Tanh-based quantization with layer re-ordering. Weights are trained by simply using the sign function with STE.

The results in Table 5.5 show that our 1-bit SP-Net surpasses the independently trained counterparts by a large margin when the activations are in extremely low-precision.

Slimmable SP-Net. Our results for slimmable SP-Net in Table 5.7 demonstrate the flexibility of our network, however they show no clear accuracy advantage over independently trained networks. We tested widths of 0.25, 0.5 and 1.0, with slimmable SP-Net and independently trained networks constantly outperforming each other. However, it is worth noting that our slimmable SP-Net network can simultaneously switch width and bitwidth on demand.

TABLE 5.7: Tiny ImageNet accuracy for a SP-Net ResNet-18.

Width	Weights bitwidth	Activations bitwidth	Slim/SP-Net	Independent
0.25	2	2	**30.3**	26.5
0.25	2	32	**36.5**	34.2
0.25	32	2	**35.6**	32.8
0.25	32	32	**39.7**	37.8
0.5	2	2	40.9	**41.3**
0.5	2	32	44.8	**49.7**
0.5	32	2	**46.1**	45.8
0.5	32	32	49.0	**52.8**
1.0	2	2	**47.8**	47.4
1.0	2	32	50.5	**54.4**
1.0	32	2	**51.8**	50.7
1.0	32	32	53.0	**56.4**

Self-Distillation. Our self-distillation method in Sec. 5.3.3 proved to be very effective, by improving the accuracy across all the switches, as can be observed in Table 5.8. Additionally, it can be observed that the validation performance of intermediate switches is superior to the performance of the full-precision one. However, we credit this to over-fitting, since the training accuracy is higher for the full precision switches. We also compared the performance of the distillation losses, confirming our feature maps distillation loss $\boldsymbol{L}_f$ is indeed benefiting the overall performance of the network. The value of $\boldsymbol{L}_f$ is expected to be several orders of magnitude larger than $\boldsymbol{L}_{out}$, since it is dictated by the number of layers and the size of the intermediate feature maps. We set the hyper-parameters $\alpha_1 = 1$ and $\alpha_2 = 1e^{-7}$ to bring $\boldsymbol{L}_f$ to the same order of magnitude with $\boldsymbol{L}_{out}$. In Figure 5.2, we plot the accuracy curves during training of our SP-Net with and without self-distillation.

TABLE 5.8: Top-1 accuracy for a SP-Net network with and without self-distillation for ResNet-18 on Tiny ImageNet.

Weights bitwidth	Activations bitwidth	Vanilla SP-Net	SP-Net+L_{out}	SP-Net+$L_{out} + L_f$
2	2	50.1	53.0	**53.3**
2	3	50.5	53.8	**53.9**
2	32	51.2	**54.2**	53.8
3	2	50.8	53.4	**53.9**
3	3	51.5	54.0	**54.4**
3	32	52.3	54.1	**54.5**
32	2	51.4	52.6	**53.3**
32	3	52.1	53.4	**53.9**
32	32	52.9	**52.9**	52.8

5.4.3 Evaluation on Pascal VOC

The Pascal VOC 2012 dataset [151] contains dense image annotations for the task of semantic segmentation. It contains 1.4k images for training and 1.4k for validation. It was extended with extra 5.6k training images by the Semantic Boundaries Dataset (SBD) [152]. It contains annotations per pixel for 20 classes plus a background class. The mean Intersection Over Union (mIOU) is used as performance measurement. A FCN-Resnet18-32s architecture pretrained on ImageNet was used and fine-tuned with weight decay of $1e^{-6}$ and initial learning rate of $1e^{-4}$ decreased by a factor of 0.1 at iterations 30 and 60 for 90 epochs. As previously explained, the batch-norm parameters from the pretrained network were used to initialize the batch-norm parameters of all the switches. All the layers were quantized except the last one, which in this case is a convolutional rather than a fully connected layer. The images were to 224×224 and batch size of 16 was used both for training and validation. We used weight bidwiths of $\{2, 3, 32\}$ and activations bitwidths of $\{2, 32\}$. Our results are reported in Table 5.9.

TABLE 5.9: Pascal VOC mIOU for a SP-Net FCN-ResNet-18-32s.

Weights bitwidth	Activations bitwidth	SP-Net (single)	Independent (multiple)
2	2	**56.1**	54.0
2	32	**58.4**	58.2
3	2	57.0	**57.8**
3	32	60.4	**61.7**
32	2	59.1	**60.2**
32	32	62.0	**62.9**

5.4.4 Ablation Studies

Non-Switchable Batch Normalization. We replaced our S-BN layers with standard BN ones to verify and demonstrate that privatizing all BN layers for each switch is essential to successfully perform inference. Figure 5.3 shows the validation plots for a SP-Net network with 4 switches with and without S-BN. The switches of the network with S-BN learn at different rates, with the high precision ones learning faster and achieving higher accuracy than the low precision ones, while the accuracy of the switches without S-BN quickly stagnates and does not recover.

Impact of Multiple Switches. The impact of increasing the number of switches was investigated. In Table 5.10, we present the results of SP-Net with 4 and 16 switches

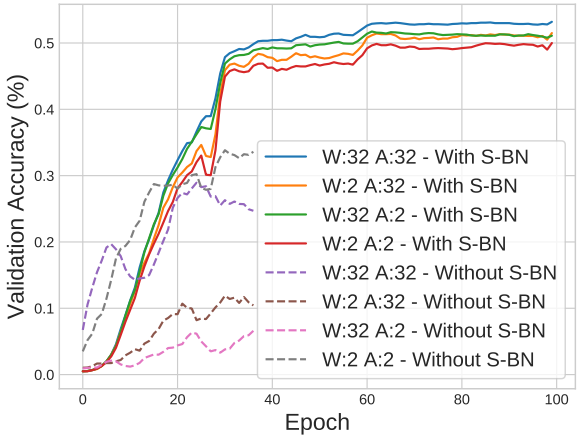


FIGURE 5.3: Top-1 accuracy for SP-Net ResNet-18 with and without switchable batch normalization (S-BN) on Tiny ImageNet.

slimmed with a factor of 0.5. The first remark that we notice is that with an increasing number of switches, the initial learning rate should be lowered. For the network with 16 switches, we set the initial learning rate to 0.03. The accuracy obtained using fewer switches is higher, we conjecture this is due to the additional constraints; however, it indicates an interesting direction of research.

TABLE 5.10: Top-1 accuracy (%) of SP-Nets on Tiny ImageNet with 4 and 16 switches slimmed with a factor of 0.5.

		Weights bitwidth				Weights bitwidth			
		2	4	8	32	2	4	8	32
Activation bitwidht	2	**41.3**	-	-	**43.8**	37.7	41.3	41.4	41.4
	4	-	-	-	-	38.6	42.3	42.9	42.4
	8	-	-	-	-	38.4	42.5	42.7	42.5
	32	**43.3**	-	-	**45.2**	39.6	42.7	43.2	43.1

Switchable Precision in Weights vs Activations. Binary ensemble neural network [153] found that QNNs activations are more sensitive to quantization than weights. Our observations on a SP-Net MobileNet confirm their findings. However, for our SP-Net ResNet-18, we observe that weights and activations are about equally sensitive. For example, by freezing the weights to 2-bit and varying the bitwidths of activations in $\{2, 4, 8, 32\}$, we can observe that the accuracy gap is 1.8% with top accuracy of 34.3% for the ReLU-based quantizer. Similarly, by freezing the activations to 2-bit and varying the bitwidths of weights, the gap for ReLU-based quantizer is 3.8% with top accuracy of 36.3%.

5.5 Conclusion

In this chapter we have proposed a DNN capable of operating at variable precisions on demand. With this approach, we grant devices and end-users real-time control over the performance of the DNNs powering inference algorithms. Our approach is lightweight and does not require altering the model, making it compatible with connectivity-free and limited memory devices. An additional virtue of our proposed network lies in the ability to train a single network that can be distributed to different devices based on their capabilities. We have demonstrated the flexibility of our method with multiple quantization functions and a slimming complementary strategy. Moreover, we have proposed a training procedure to increase the accuracy of our network across the available precisions. Finally, we performed multiple ablation studies to analyze the performance of our approach in different scenarios.

Following the spirit of Universally Slimmable Networks [148], a continuously quantizable SP-Net would be an interesting research direction as well as mixed precision SP-Net, where each layer is quantized independently on-demand. This method could be used to find the optimal layer-wise bitwidth for weights and activations.

Chapter 6

Angle Penalized Neural Network Scaled Quantization

6.1 Introduction

Dot-product computations in convolutional layers of quantized neural networks can be efficiently performed using any scaled version of the quantized weight vectors by factorizing out the multiplicative factor and scaling the output correspondingly. However, common weight quantization strategies frequently disregard this opportunity. Of particular interest to us is Binary Weight Networks (BWN) and Xnor-Nets [24], for binary weights and binary activations, respectively. By introducing scaling factors in their binary convolutional filters, the authors were able to better approximate full-precision representations. The scaling factors for the weights are computed analytically as the mean of the full-precision filters and a compensation to the gradient is performed after backpropagation in order to take into account the effect of the scalings. However, this adds considerable overhead to the training process and additionally only a scaling factor per output channel is considered.

Anderson and Berg [154] and Banner *et al.* [155] studied the training of Quantized Neural Networks (QNNs) with the Straight Through Estimator (STE) and asserted that the effective training of QNNs results from the *angle preservation* property which states that the quantized vectors in high dimensions have a relatively small angle with respect to the real vectors that generated them. Additionally, it is known that in linear operations

such as convolution, proportional weight filters generate proportional activations. In other words, weight vectors with zero angle, or equivalently unitary cosine distance, perform proportional transformations.

Building on top of these findings, we introduce an approach to better approximate the real weights in QNNs while jointly optimizing the scaling factors. We develop a quantization strategy focused on the structure of quantized representations as a whole, rather than on their individual values, to develop a training strategy capable of exploiting the aforementioned opportunity. Our approach, Angle Penalized neural network Scaled Quantization (APSQ) jointly introduces both the quantization and implicitly the scaling into the network optimization procedure. Moreover, by formulating the quantization as a regularization problem, our method allows for continuous gradient propagation, making it more stable than competing approaches based on gradient estimations. In our approach, angles, rather than distances, between pairs of corresponding real and quantized vectors are penalized. Given that quantized vectors are fixed, this regularization encourages the real vectors to lie near any scaled version of the quantization set. By constraining our problem using APSQ, the allowed region of exploration reduces to only areas near the diagonals in Figure 6.1, whereas independently computed scalings as done by Xnor-Nets [156] allow to explore the whole real vector space, potentially converging to minima distant to the feasible regions. Moreover, our work can be seen as a generalization of BWN with scaling factors progressively learned and it is agnostic to the quantization function used to map real vectors to the quantized ones.

We extend our approach to kernel-wise scalings in order to further increase the representation capacity of convolutional filters while maintaining efficient binary and fixed-point operations. In order to maintain a reasonable memory footprint we quantized the kernel-wise scalings without a major impact in accuracy.

Specifically, scaled binary kernels on a Resnet-18 architecture allow to achieve near full-precision accuracy on ImageNet classification, obtaining 67.0% (-2.3%) with an equivalent memory footprint of 2 bits per weight while using only summations.

6.2 Background and Motivation for Angle Penalization

Early DNN quantization strategies relied on STE [103], which allows to propagate gradients through non-differentiable rounding functions. The noisy surrogate gradients of this STE method converge to a critical point of the real estimated population loss [157] and we refer to this approach as lazy evaluation. Extensive analysis about STE has been published in the literature, and multiple weaknesses have been pointed out, including slow and sub-optimal convergence [83] and improved gradient estimators such as [145, 158] have been proposed. Recently, STE has been shown to be an implementation method of the mirror-descent algorithm for certain convex projections [159]. However, there is no theoretical justification for the use of STE for non-differentiable rounding functions (*e.g.*, sign [22]) which project the real vectors to nonconvex quantized sets.

On the other hand, non-lazy evaluation consists in starting with a full-precision DNN and progressively quantize it using a differentiable approach. This is achieved by including a penalization or driving factor that pushes the network parameters towards the quantized set while increasingly updating such driving factor using a hyper-parameter. Notable works in this direction include [83, 160–162] and relaxations have also been used extensively [163–165].

BWN [24] falls in the category of QNNs trained with lazy evaluation (*i.e.* STE), where the scalings are computed analytically using the noisy weights. In contrast, by formulating our approach as a regularization term, it allows for real gradient propagations and is compatible with both lazy and non-lazy approaches.

Furthermore, networks trained using the methodology in BWN are not restricted on their feasible exploration region potentially targeting minima reachable by the underlying real weights but unreachable to the quantized ones (*i.e.* far from the region of feasible quantized vectors) as can be seen in Figure 6.1 left; which motivates the need for angle penalization as will be addressed in section 6.5. In the left figure, the underlying full precision weight vector is initialized randomly in the parameter space $\mathbb{R}^2$. Subsequently, it is quantized and the gradient is evaluated on the quantized version. Then the gradient is transplanted to the real vector and a corresponding gradient descent update is performed moving the real vector in the direction of the minimum unreachable by the quantized one. As the process is repeated the quantized vector starts to converge to a

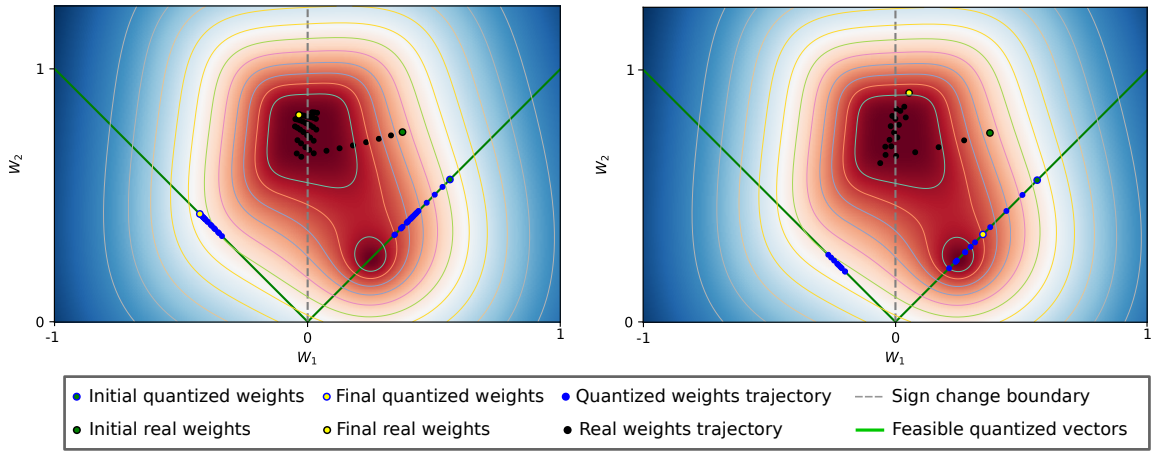


FIGURE 6.1: Scaled quantized neural network trained with the straight through estimator fails to converge to a reachable minima in the feasible region while being dragged by an unreachable minimum located in the middle of two quantization bins next to the sign change boundary. Left: A QNN trained using the procedure in BWN. Right: A QNN using learnable scalings.

reachable local minimum in the diagonal; however, the real vector approached the sign change boundary. Before converging to the reachable minimum a sign change occurred in W_1 and the network is now evaluated on the opposite diagonal. Finally, the network enters a stage of constant sign switching around an unreachable minimum in the middle of the quantization bins.

Alternative approaches like BNN+ [158] and Xnor++ [156] are closely related to our end goal of learning scalings for QNNs. These schemes simply include the scalings as free learnable parameters in the optimization graph, however the inherent drawback is still present as depicted in Figure 6.1 right. The latent real weights are not aware of the existence of the scalings, thus freely move in the parameter space eventually being dragged to an unreachable minima. Identical settings were used in both experiments, however it can be observed in the right side experiment that larger steps are taken, this is due to the independently learned scalings detached from the real weights. In this occasion the quantized vector reaches the minimum in the diagonal in a few steps; however, similarly to the first example, the real vector already lies next to the sign change boundary, therefore any further steps results in a sign change. Once the quantized vector is located on the opposite diagonal, the network aims towards the unreachable minimum in the center. Finally, the network enters a stage where both minima are simultaneously targeted depending on the sign of W_1 causing the quantized vector to oscillate around the reachable minimum.

6.3 Problem Definition

Given a DNN and a set of data points $\boldsymbol{D}$, the task of network scaled quantization consists in jointly optimizing the network parameters $\boldsymbol{\alpha} \in \mathbb{R}^j$ and $\boldsymbol{W} \in \boldsymbol{C}^n$ that minimize a loss function $\boldsymbol{L}$, where n is the number of weights and $\boldsymbol{C}$ is a small discrete set of values, denominated as *codebook*, determined both by the *quantizer* and the *bitwidth*. Additionally, $\boldsymbol{\alpha}$ denotes weight groups scaling parameters, which can be layer-wise, channel-wise or kernel-wise.

Formally, scaled quantized DNN optimization is defined as:

$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{W}} \quad & \boldsymbol{L}(\boldsymbol{D};\boldsymbol{\alpha},\boldsymbol{W}) \\ \text{s.t.} \quad & \boldsymbol{W} \in \boldsymbol{C}^n, \boldsymbol{\alpha} \in \mathbb{R}^j. \end{aligned} \tag{6.1}$$

Depending on the choice of weight grouping, the number of j scaling parameters in $\boldsymbol{\alpha}$ per layer will be: 1 for layer-wise grouping, K for channel-wise and $I \times K$ for kernel-wise, where I and K denote the number of input and output channels, respectively. Hereafter, kernel-wise grouping will be used for illustration. A kernel-wise scaled quantized convolutional layer with K output channels is defined as follows:

$$\mathbf{O}_k = \sum_{i=1}^{I} \mathbf{A}_{i,:,:} * (\alpha_i \cdot \mathbf{F}_{k,i,:,:}), \tag{6.2}$$

where the $*$ operator denotes convolution. $\mathbf{F}_k \in \boldsymbol{C}^{I \times h_f \times w_f}$, $\mathbf{F}_k \subset \boldsymbol{W}$ is the k-th convolutional filter. h_f and w_f denote the height and width of the filters, respectively. $\mathbf{A} \in \mathbb{R}^{I \times h_{in} \times w_{in}}$ and $\mathbf{O}_k \in \mathbb{R}^{h_{out} \times w_{out}}$ denote the input activations and output pre-activations of the filter, where h_{in}, w_{in}, h_{out}, w_{out} represent the height and width of the input and output feature maps, respectively.

6.4 Angle Penalized Scaled Quantization

Existing weight quantization methods commonly neglect the potential increase in representation capacity offered by scaling parameters [22, 23, 137], do not include them into the optimization process [166] or rely on coarse approximations of $\boldsymbol{W}$ [24] to obtain them.

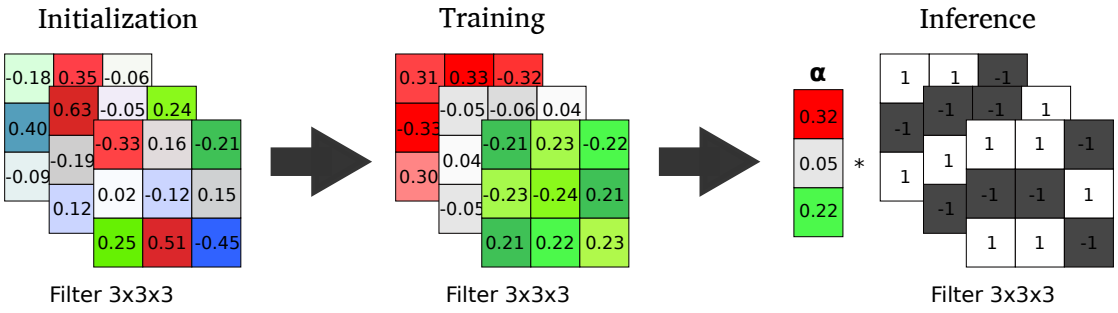


FIGURE 6.2: Stages of kernel-wise APSQ.

In order to simultaneously optimize the quantized weights and scaling parameters of our network, we formulate the quantization problem as a regularization term. However, unlike [83], rather than reducing the distance to the quantized set, we penalize the angle with respect to the quantized weight vector version, effectively reducing the distance to any scaled version of such set and providing our algorithm with additional flexibility and faster convergence. Moreover, in contrast to [24], our method allows for computation of real gradients.

The quantizer employed in this paper is described in Section 6.4.1 and the formulation of our approach is provided in Section 6.4.2. Lazy and non-lazy versions of our method are discussed in Section 6.4.3. In Sections 6.4.4 and 6.4.5 we describe the implementation details of APSQ. In Section 6.5 we provide a comparison with L_1 and L_2 penalizations and finally in Section 6.6 we provide an analysis on the advantages and drawbacks of adding real valued scalings to the quantized kernels.

6.4.1 Weight Quantization Function Overview

Common regularization-based quantization approaches such as Proxquant [83] and [160] operate by computing a quantized version of the weight matrix $\boldsymbol{W}$ using a quantizer denoted Quant$(\cdot)$, and penalizing the network based on the distance to the quantized version. In this paper we take a similar approach, however, rather than minimizing the distance to the closest quantized vector, we minimize the angle, effectively reducing the distance toward the optimal scaled version of the quantized vector. Finally, the quantized $\hat{\boldsymbol{W}}$ is factorized into a representative quantized vector for such angle obtained from the quantization function Quant$(\cdot)$, multiplied by a scalar.

One virtue of our method lies in the fact that it is readily compatible with any quantization function. For general uniform k-bit weight quantization we will use the quantizer proposed in DoReFa [135], however it can be applied to non-uniform quantizers such as [146, 147]. For 1-bit networks we will simply use the sign function. Therefore, the quantizer is defined as:

$$\mathrm{q}(\mathbf{x}) = \frac{1}{2^k - 1}\mathrm{round}((2^k - 1)\mathbf{x}), \tag{6.3}$$

$$\mathrm{DoReFa}(\mathbf{x}) = 2 \cdot \mathrm{q}(\frac{\tanh(\mathbf{x})}{2 \cdot \max(|\tanh(\mathbf{x})|)} + \frac{1}{2}) - 1, \tag{6.4}$$

$$\mathrm{Quant}(\mathbf{v}) = \begin{cases} \mathrm{sign}(\mathbf{v}) & \text{if k} = 1 \\ \mathrm{DoReFa}(\mathbf{v}) & \text{if k} > 1 \end{cases}. \tag{6.5}$$

where k denotes the quantized weights bitwidth while $\mathbf{x}$ and $\mathbf{v}$ real weight vectors. In equation 8.5 the maximum is taken over the absolute values of the vector. The tanh, round and sign operations are permformed elementwise.

Note that the function $\mathrm{Quant}(\dot{})$ is not differentiable, however this is not an issue since gradients need not be propagated through it. The reason is because the quantized weight vector serves only as a static reference vector for the real vector as will be explained subsequently.

6.4.2 Angle Penalization

In order to optimize (equation 6.1), we re-formulate DNN scaled quantization as a regularization term in the network task-specific loss function:

$$\boldsymbol{L}(\boldsymbol{D}, \boldsymbol{\alpha}, \hat{\boldsymbol{W}}) = \boldsymbol{L}_{task}(\boldsymbol{D}, \boldsymbol{\alpha}, \hat{\boldsymbol{W}}) + \beta R_{APSQ}(\hat{\boldsymbol{W}}), \tag{6.6}$$

where $\boldsymbol{W} \in \boldsymbol{C}$ is replaced with $\hat{\boldsymbol{W}} \in \mathbb{R}^n$ and $\beta > 0$ is a quantization enforcing hyper-parameter.

We re-arrange the i-th kernel of a convolutional filter $\mathbf{F}$ into a 1-dimensional vector $\mathbf{v}$. In order to minimize the angle between two vectors, we can equivalently maximize their cosine distance:

$$R_{APSQ} = \frac{1}{N}\sum_{n=0}^{N}\left[1 - \frac{\mathbf{v}_n \cdot \mathrm{Quant}(\mathbf{v}_n)}{\|\mathbf{v}_n\|_2 \cdot \|\mathrm{Quant}(\mathbf{v}_n)\|_2}\right], \tag{6.7}$$

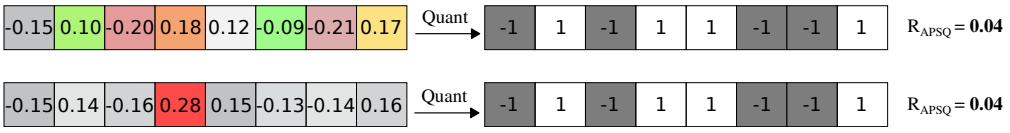


FIGURE 6.3: Angle based quantization penalizes whole structures, allowing individual weights to quantize at their own rate. In the top vector, all real elements deviate from a quantized representation by a considerable amount. In the bottom one, all real elements are very close to a quantized representation with the exception of one that deviates by a large amount. However, both vectors have the same angle with respect to the quantized vector and are penalized equally.

where N is the number of kernels across the whole network and $\|\cdot\|_2$ denotes the L_2 norm. Note that $\text{Quant}(\mathbf{v}_n)$ is a reference vector and should be not included into the optimization graph, therefore gradients should not be propagated through it. For the particular binary weights case, it can be efficiently computed as:

$$R_{APSQ} = \frac{1}{N}\sum_{n=0}^{N}\left[1 - \frac{\|\mathbf{v}_n\|_1}{\|\mathbf{v}_n\|_2 \cdot \sqrt{numel}}\right], \tag{6.8}$$

where $numel$ is the number of elements in $\mathbf{v}_n$

The Euclidean distance between a randomly initialized vector and its quantized representation will be typically larger than a scaled version of the same, potentially speeding up the convergence. Additionally, angle penalization restricts the exploration region to feasible quantized vectors, addressing the inherent problem of vanilla scaled QNNs (*e.g.* BWN) depicted in Section 6.2. Figure 6.5 illustrates our solution as well as a Euclidean distance based alternative.

Progressively increasing the parameter β encourages the network to slowly converge to a scaled quantized representation. Multi-step and exponentially increasing β schedulers were tested with both schedulers providing similar results.

In Figure 6.4, we plot the cosine distance during training against validation accuracy using a pretrained model. Kernel-wise scalings and both lazy and non-lazy projections were used as explained in the next section. As the angle between the real and quantized vectors decreases, the validation accuracy increases which further validates the usefulness of angle penalization.

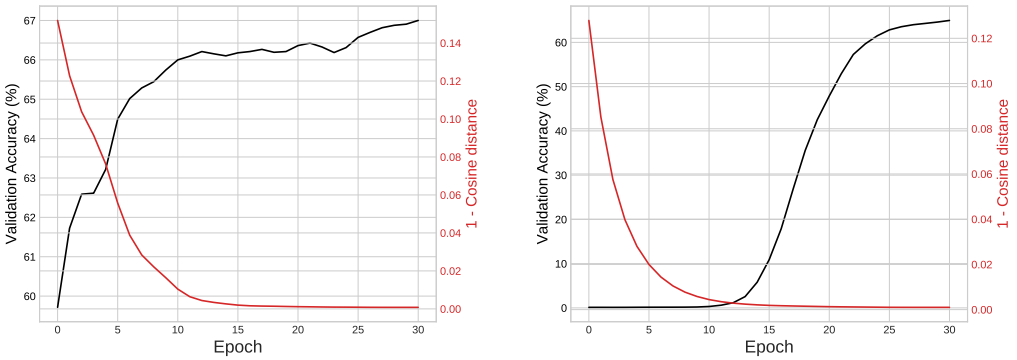


FIGURE 6.4: Cosine distance during fine-tuning along with Top-1 ImageNet validation accuracy using lazy (left) and non-lazy (right) APSQ. As the angle between the real and quantized vectors decreases, the validation accuracy increases.

6.4.3 APSQ in Lazy and Non-Lazy Context

As pointed out by Proxquant [83], the typical QNNs with STE training procedure described in [22], can be seen as a regularized optimization with regularization parameter $\beta = \infty$ using a lazy projection strategy, this is, using an auxiliary variable to project onto a constrained set during evaluation. This is equivalent to Nesterov's dual-averaging method, where the subgradients are accumulated during stochastic learning updates and the weights are projected onto a closed convex set $\boldsymbol{C}$ at every iteration [167], in this case a quantization set. Using our previous notation, lazy evaluation is defined:

$$\hat{\boldsymbol{W}}_{t+1} = \hat{\boldsymbol{W}}_t - \eta_t \nabla \boldsymbol{L}(\boldsymbol{W}_t)|_{\boldsymbol{W}_t = \text{Quant}(\hat{\boldsymbol{W}}_t)}. \tag{6.9}$$

where ∇ denotes the gradient computed using the quantized weights and η_t the learning rate.

Using a soft projection on the closed quantization set, by setting the initial value $\beta_0 = 0$ and $\beta \to \infty$, would yield the method in Binaryrelax [168]. If additionally a non-lazy projection is used, it yields the methods in Proxquant [83] and ELQ [166]. Rather than adding a regularization term in the loss function to encourage quantizedness, these methods proceed by taking a weight update step using any optimizer and subsequently projecting:

$$\hat{\boldsymbol{W}}_{t+1} = \text{SoftProj}_{\text{Q}}(\hat{\boldsymbol{W}}_t - \eta_t \hat{\nabla} \boldsymbol{L}(\hat{\boldsymbol{W}}_t)), \tag{6.10}$$

where $\text{SoftProj}_{\text{Q}}(\cdot)$ denotes a soft projection onto the quantized set and $\hat{\boldsymbol{W}}$ the gradient computed using the real underlying weights.

BNN+ [158] and [169] consider a combination of both lazy and non-lazy approaches. They follow the regular QNN with STE training scheme and additionally regularize the auxiliary weights using L_1 and L_2 distances to the quantized set:

$$\begin{aligned}\hat{\boldsymbol{W}}_{t+1} =& \hat{\boldsymbol{W}}_t - \eta_t \nabla \boldsymbol{L}_{task}(\boldsymbol{W}_t) - \\ & \eta_t \hat{\nabla} \beta R_Q(\hat{\boldsymbol{W}}_t)|_{\boldsymbol{W}_t = \text{Quant}(\hat{\boldsymbol{W}}_t)},\end{aligned} \tag{6.11}$$

where R_Q denotes a differentiable regularization term encouraging the weights to become quantized.

Finally, we define a vanilla non-lazy approach, where the quantization is achieved uniquely by a regularization parameter:

$$\hat{\boldsymbol{W}}_{t+1} = \hat{\boldsymbol{W}}_t - \eta_t \hat{\nabla} \boldsymbol{L}(\hat{\boldsymbol{W}}_t)|_{\boldsymbol{L} = \boldsymbol{L}_{task} + \beta R_Q}. \tag{6.12}$$

Note that during validation, both lazy and non-lazy methods, perform hard quantization in order to measure the performance of the quantized version.

In our experiments we will use equation 6.12 and equation 6.11 for the non-lazy and lazy comparison, respectively, with $R_Q = R_{APSQ}$. Our results indicate that lazy projection is superior in this context.

6.4.4 Inference Using APSQ

In the previous sections we described the training mechanism for APSQ neural networks. In order to perform inference during validation, a hard quantization must be utilized for both lazy and non-lazy training procedures. In the binary case, we binarize the kernels and we use the kernel-wise means as the scaling factors as described by Xnor-Net [24] since it represents the shortest Euclidean distance to a quantized vector. For general k-bit quantization we use the kernel-wise maximum values in order to make use of the whole real range as follows:

$$\alpha_i = \begin{cases} \text{mean}(\hat{\mathbf{F}}_{k,i,:,:}) & \text{if k} = 1 \\ \max(\hat{\mathbf{F}}_{k,i,:,:}) & \text{if k} > 1 \end{cases}, \tag{6.13}$$

$$\mathbf{F}_{k,i,:,:} = \begin{cases} \text{Quant}(\hat{\mathbf{F}}_{k,i,:,:}) & \text{if k} = 1 \\ \text{Quant}(\frac{\hat{\mathbf{F}}_{k,i,:,:}}{\alpha_i}) & \text{if k} > 1 \end{cases}, \tag{6.14}$$

where $\hat{\mathbf{F}} \in \mathbb{R}^{I \times h_f \times w_f}$, $\hat{\mathbf{F}} \subset \hat{\boldsymbol{W}}$. This process is illustrated in Figure 6.2 using a single $3 \times 3 \times 3$ convolutional filter.

6.4.5 Post-Training Fine-Tuning

In the non-lazy setting, there is a discrepancy between the training and validation behaviors. Due to this, once a model has been selected using the validation set, additional fine-tuning is performed by freezing the quantized weights and scalings at their final value as explained in Section 6.4.4 and fine-tuning the batch-normalization parameters.

6.5 Comparison with L1 and L2 norms

A competing approach to our APSQ consists in having learnable scalings and using L_1 or L_2 norms to the quantized set as explored by [158]:

$$R_L = \|\mathbf{v} - \boldsymbol{\alpha} \cdot \text{Quant}(\mathbf{v})\|_p \,, \tag{6.15}$$

where $\|\cdot\|_p$ denotes a p-norm and $\mathbf{v}$ denotes a convolutional kernel re-arranged into a 1-dimensional vector.

Nonetheless, these norms are unbounded (unless weight clipping is used) and depend on the magnitude of the full-precision auxiliary weights. Therefore, small penalization factors must be selected to avoid overshooting; thus, slow and unstable convergence is observed as depicted in Figure 6.5 left. As the quantized vector moves towards the minima, the full-precision underlying weight vector moves towards the sign change boundary without crossing it and then slowly starts to close the Euclidean distance with the quantized vector.

In contrast, our angle penalization is bounded and does not depend on the magnitude of the weights, therefore a larger penalization can be selected. As depicted in Figure 6.5 right, the real underlying vector closes the distance with the feasible quantized set in

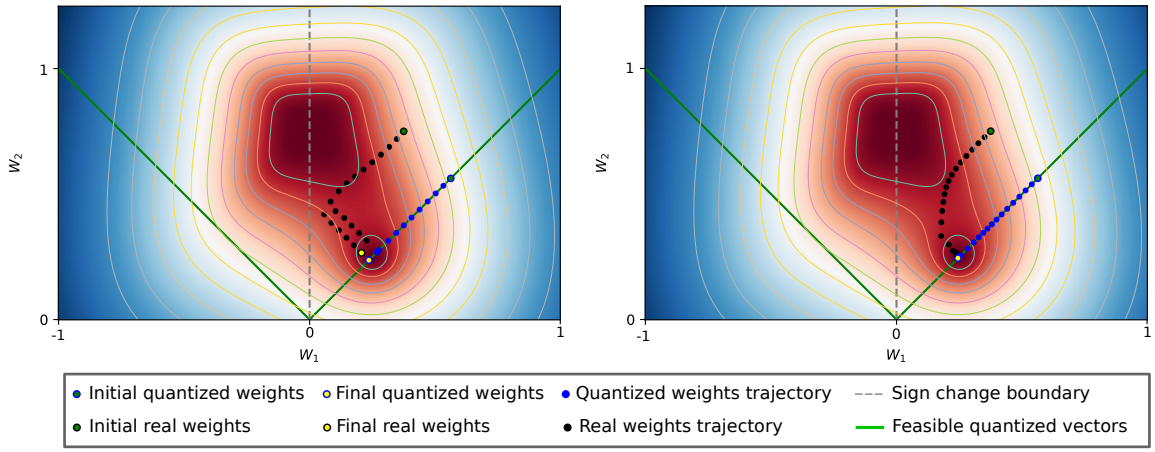


FIGURE 6.5: Scaled QNN trained with STE converges to reachable minima in both cases. Left: L_2 penalization with learnable scalings. L_2 norm depends on the magnitude of the weights and it is unbounded; therefore, care must be taken when selecting the penalization hyperparameter. Right: APSQ (ours) is independent of the weights magnitude and a higher penalization hyperparameter can be used. Additionally, ASPQ quantizes structures as a whole rather than independent weights.

a timely manner without overshooting. In practice we start with a low penalization to grant the weights enough flexibility to switch sign and progressively increasing it using a linear scheduler.

In fact, in the binary case, our angle penalization is inversely proportional to the ratio L_1/L_2 as can be seen in equation 6.8. Meaning that in the $\mathbb{R}^2$ space, the largest penalization occurs at 0° and 90° (the sign change boundaries) where $L_1 = L_2$, and the smallest penalization at the diagonals where $L_1 > L_2$.

p-norm based progressive quantization quantizes all the weights at the same rate using a weight penalization parameter. The reason is the gradient per weight depends exclusively on such weight. Let $\boldsymbol{\lambda} = \mathbf{v} - \boldsymbol{\alpha} \cdot \text{Quant}(\mathbf{v})$, the derivative of the p-norm w.r.t. to $\mathbf{v}$ is given by:

$$\frac{\partial \|\boldsymbol{\lambda}\|_p}{\partial \mathbf{v}} = \frac{\partial \|\boldsymbol{\lambda}\|_p}{\partial \boldsymbol{\lambda}} \frac{\partial \boldsymbol{\lambda}}{\partial \mathbf{v}}. \tag{6.16}$$

Given that $\text{Quant}(\mathbf{v})$ is treated as a constant, $\frac{\partial \boldsymbol{\lambda}}{\partial \mathbf{v}} = \mathbf{1}$.

In contrast, angle-based regularization quantizes structures as a whole; thus, it has the advantage that each weight gets quantized at its own rate using a kernel penalization parameter. In this case, the gradient is affected by both the individual weights and

the norm of the whole weight vector. Let $\text{CosDis}(\mathbf{v}, \text{Quant}(\mathbf{v})) = \frac{\mathbf{v} \cdot \text{Quant}(\mathbf{v})}{\|\mathbf{v}\|_2 \cdot \|\text{Quant}(\mathbf{v})\|_2}$, the derivative w.r.t. $\mathbf{v}$ is given by:

$$\frac{\partial \text{CosDis}(\mathbf{v}, \text{Quant}(\mathbf{v}))}{\partial \mathbf{v}} = \frac{\text{Quant}(\mathbf{v})}{\|\mathbf{v}\|_2 \|\text{Quant}(\mathbf{v})\|_2} - \text{CosDis}(\mathbf{v}, \text{Quant}(\mathbf{v})) \frac{\mathbf{v}}{\|\mathbf{v}\|_2^2} \tag{6.17}$$

6.6 Accuracy, Computation vs Memory Footprint Trade-Off

Kernel-wise $\boldsymbol{\alpha}$ parameters improve the representation capacity of QNNs with a small increase in computational complexity at the expense of memory size trade-off.

Using the notation from Section 6.3, during a convolution operation, for every element of an input feature map, $h_f \times w_f$ low-bidwith multiplication are executed. Only one additional full-precision multiplication per element is required for the scalings. Similarly, $I \times K \times L$ additional floating point variables are required, where L denotes the number of layers. For instance, in a 3×3 kernel, 9 low-bidwidth plus 1 full-precision multiplications would be required per element and additional $\frac{32}{9} = 3.5$ bits per weight if using 32 floating point variables.

Evidently, the memory size is heavily impacted by the use of kernel-wise scalings. In order to mitigate this sacrifice, we quantize the scalings $\boldsymbol{\alpha}$ during training by first clipping to the range $[0, 1]$ and using $\text{q}(\boldsymbol{\alpha})$ defined in equation 6.3. Note that the scalings are not included into the optimization graph. Thus, once again, there is no need to backpropagate through the non-differentiable quantization function. After convergence, we additionally froze the weights and fine-tuned the batch-norm parameters.

We quantized $\boldsymbol{\alpha}$ using different bitwidths and reported our results in the ablation studies in Section 6.7.1. In order to have exactly an additional 1 bit per weight, we quantized $\boldsymbol{\alpha}$ using $h_f \times w_f$ bits (*e.g.* 9 bits for 3×3 kernels) and compared against a 2-bit QNN, which is equivalent in terms of memory but without the benefit of avoiding fixed-point multiplications. In Table 6.4 we reported the effect of quantization on $\boldsymbol{\alpha}$.

Specifically, for 1×1 convolutional layers used in downsample connections in Resnet architectures [6] and extensively used in Network-in-Network [170] and Mobilenets [6, 30], kernel-wise scalings would be redundant compared to full precision layers and therefore are replaced with channel-wise scalings.

6.7 Experiments

In this section, we tested our approach on Resnet-18 and Resnet-50 [6] architectures, on the task of image classification on a large-scale dataset, ImageNet (ILSVRC-2012) [2]. Note that, our objective is to demonstrate the effectiveness of angle penalization in comparison to distance based regularization and our approach can be used in conjunction with any quantization method. To this end, we use BinaryConnect (BC) [22] as the base algorithm and compare against BWN [156] as well as learnable scaling methods, however, any recent method [83, 159, 162, 168] could be employed.

We implemented our APSQ training scheme using the Pytorch framework. We tested our approach against our implementations of BWN and BinaryConnect (BC) with straightforwardly added learnable scalings. BNN++ can be thought of as BC with learnable scalings, however BNN++ additionally has a different training methodology using a soft quantization. A pretrained ImageNet network was used for all the cases. The learnable scalings were initialized to the kernel means or channel means, depending the case. As a common convention, the last layer (the fully-connected) was not quantized.

In all our experiments we reported our results for APSQ using lazy optimization with STE as per equation 6.11. Biases were kept as full precision.

6.7.1 Evaluation on ImageNet

As in previous sections, the ImageNet 2012 dataset was used. Only the Top-1 is used to compute the loss.

The training regimes used were dependent on the mode. For APSQ, we used SGD with an initial learning rate of $1e^{-3}$ and weight decay of $1e^{-4}$ with angle penalization hyperparameter β scheduler, with learning rate update by a factor of 0.1 at epoch 25. For the modes of BWN, BC + learnable scalings and BC + learnable scalings + L_2,

TABLE 6.1: Top-1 validation accuracy (%) for a 1-bit APSQ ResNet-18 on ImageNet. LS = Learnable Scalings

Mode	Channel-wise		Kernel-wise	
	Top1	Top 5	Top1	Top5
BWN	62.0	84.0	65.5	86.5
BC + LS	61.4	83.7	65.5	86.5
BC + LS + L_2	61.8	84.2	65.9	86.9
BC + LS + angle (ours)	60.5	83.1	65.4	86.4
APSQ (ours)	**62.7**	**84.8**	**67.1**	**87.9**

TABLE 6.2: Top-1 validation accuracy (%) for a 1-bit APSQ ResNet-50 on ImageNet. LS = Learnable Scalings

Mode	Channel-wise		Kernel-wise	
	Top1	Top5	Top1	Top5
BWN [24]	70.4	89.7	70.8	89.8
BC + LS	69.2	88.8	69.6	89.3
APSQ (ours)	**70.7**	**89.0**	**71.8**	**90.6**

Adam optimizer with an initial learning rate of $1e^{-5}$ and weight decay $1e^{-6}$ was used, with learning rate adjustments by a factor 0.1 at iterations 15 and 25. BC + Learnable scalings + angle is extremely sensitive to the angle penalization, therefore we used $\beta = 1$ across the whole training. All networks were trained for a total of 30 epochs.

In Tables 7.1 and 6.2 we report our ImageNet results for full-precision activations 1-bit weight APSQ on Resnet-18 and Resnet-50 architectures. Our approach outperformed straight-forward learnable scalings and BWN both in channel-wise and kernel-wise strategies. BWN and BC have been endowed with the same scalings, thus the memory footprint is the same. Additionally, our 1-bit APSQ Resnet-18 is only 2.1% below its full-precision counterpart using only summations and with $(h_k \times w_k)$-fold memory size reduction per layer.

For multi-bit quantization, BWN does not provide a formulation for computing the scalings, therefore only DoReFa-Net [135] with learnable scalings is compared against our approach. A comparison for 2-bit scaled weights and 2-bit activations is provided in Table 6.3. For quantizing activations, regular DoReFa quantization is used with STE for backpropagating gradients. A pretrained network was used in all cases and fine-tuned like in the previous case. The same weight scheduler as before was used. Likewise to BWN, the last layer of our network was not quantized.

Quantizing the Scalings. Although our kernel-wise approach improves the representation capacity of QNNs, it is at the expense of significant memory increase. Thus, in

TABLE 6.3: Top-1 validation accuracy (%) for a 2-bit weights and 2-bit activations APSQ ResNet-18 on ImageNet. LS = Learnable Scalings

Mode	Channel-wise		Kernel-wise	
	Top1	Top5	Top1	Top5
DoReFa (no scalings)	61.4	83.9	-	-
DoReFa + LS	60.5	83.3	61.6	84.0
DoReFa + LS + L_2	60.7	83.3	61.8	84.3
DoReFa + LS + angle (ours)	**61.9**	**84.1**	62.6	84.8
APSQ (ours)	61.3	83.6	**64.6**	**86.0**

this section we investigate the effect of quantizing the $\boldsymbol{\alpha}$ scalings. The quantization and fine-tuning details are included in Section 6.6. By quantization the scalings $\boldsymbol{\alpha}$ using bitwidths of $\{9, 8, 7, 6\}$, our method occupies the equivalent of $\{2, 1.8, 1.7, 1.6\}$ bits per weight for 3×3 binary convolutional kernels. The results are reported in Table 6.4 and compared against a 2 bit network. In the rest of the experiment section, full precision $\boldsymbol{\alpha}$ were used.

TABLE 6.4: Top-1 validation accuracy (%) for a 1-bit APSQ ResNet-18 on ImageNet with scalings quantization and post-training fine-tuning. A network trained with bitwidth of 2 and full-precision activations (Top-1 64.3%, Top-5 85.7%) was used for reference.

Scaling bitwidth	Equivalent weight bitwidth	Kernel-wise		Δ (2 bit)	
		Top1	Top5	Top1	Top5
6	1.6	64.9	86.3	+0.6	+0.6
7	1.7	66.4	87.1	+2.1	+1.4
8	1.8	66.9	87.2	+2.6	+1.5
9	2	67.0	87.4	+2.7	+1.7
32	4.5	67.1	87.5	+2.8	+1.8

In Figure 6.6 we plotted training curves for networks trained using different precision for the kernel-wise scalings.

6.8 Conclusion

In this chapter we have pointed an issue inherent in QNNs with scalings factors and a solution for such issue based on the minimization of the angle between pairs of corresponding real and quantized vectors. We trained a scaled QNN using our approach and tested its performance on large-scale image classification achieving competitive results. Furthermore, we have proposed additionally performing quantization on the learned scalings in order to provide a reasonable trade-off between memory footprint and computational complexity. We believe our approach provides a feasible solution to the

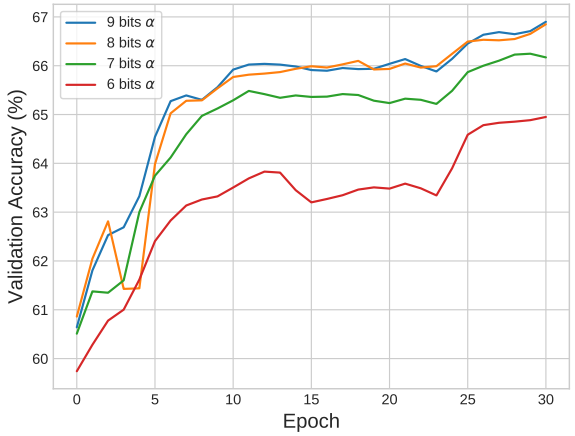


FIGURE 6.6: ImageNet accuracy with quantized $\boldsymbol{\alpha}$ scalings on 1-bit APSQ

implementation of QNNs on embedded devices given that no fixed point multiplications are required during convolutions while maintaining a close gap with a real-valued counterpart. Furthermore, our approach is complementary and compatible with any hard quantization strategy.

Chapter 7

Training 1-bit Networks on a Sphere: A Geometric Approach

7.1 Introduction

Weight binarization offers a promising alternative towards building highly efficient Deep Neural Networks (DNNs) that can be deployed in low-power, constrained devices. However, given their discrete nature, training 1-bit DNNs is not a straightforward or uniquely defined process and several strategies have been proposed to address this issue yielding every time closer performance to their full-precision counterparts across different tasks.

Optimizing them is equivalent to solving a combinatorial problem with d variables, where d, the number of weights in the network, is a value in the order of millions. Exploring the space of solutions results infeasible with traditional methods; thus, numerous approaches have been proposed in the context of DNNs including the widely accepted procedure based on the Straight Through Estimator [103] originally presented in BinaryConnect [22], and later followed by works and variants such as [23, 24, 145], proximal methods [83, 162, 171] and relaxations [165, 168].

In this work we attack the problem from the perspective of geometric optimization, introduced in Section 3.1.2 and refer the reader to [82] for further insights. This work parts from noticing that, given a static DNN with d binary weights, all the possible weight configurations lie on a $(d-1)$-dimensional hypersphere $\mathbb{S}^{d-1} = \{x \in \mathbb{R}^n : x^T x = 1\}$ residing inside a d-dimensional space. Spherical manifolds belong the class of Riemannian

manifolds; therefore, it is possible to leverage concepts of differential geometry in order to stick only to valid solutions at every step of the training process, this is, constraining the parameters of our network to a unit sphere.

Our approach offers a principled solution; nevertheless, in practice we found that simply constraining the norm of the underlying auxiliary network works just as effectively.

Additionally, our method presents improved generalization, we conjecture that, by tying the weights together, our optimization is less susceptible to large changes in a particular direction caused by single stochastic updates and consequently to over-fitting. Finally, we observe that by enforcing a unit norm on the network parameters, our network explores a space of well-conditioned matrices. In order to further comprehend the inner operation of our method, we provide an analysis of the conditioning number of the layers of our network during training [172]. We observe that constraining the magnitude of the weights leads to better convergence by improving the conditioning number of each layer. We draw a relation between the L_2, spectral and Lipschitz norms previously used to improve generalization and convergence [173, 174]. Our method additionally acts as a direct replacement to techniques such as weight decay [175] reducing the number of design choices.

Finally, we complement our approach with the scheme proposed by Liu *et al.* [176]. In a similar line to our work, Minimal Hyperspherical Energy (MHE) [176] aims at enforcing filters' diversity by projecting the filters of each layer on a hypersphere and evenly distributing them on it. This is popularly known as the Thompson/Thames problem.

7.1.1 Neural Networks as Quotient Manifolds

Manifolds in general, may be embedded in higher dimensional manifolds. For example, in the case of an $(n-1)$-dimensional hypersphere $\mathbb{S}^{n-1} = \{x \in \mathbb{R}^n : x^T x = 1\}$ residing in an n-dimensional space, and are therefore denominated sub-manifolds. This allows one to conveniently describe these entities by the use of equivalence relations. This is done by linking equivalent elements of a manifold under an action or relation. The resulting manifold is denominated a quotient manifold of the original one and is more efficiently optimized as there is no need to explore the equivalent elements but one. In this paper we notice that normalization layers determine the scalings and shifting

of networks filters, inducing an equivalence relation $\sim$ over the filter-wise manifolds $\boldsymbol{M}_i \in \mathbb{R}^n$ under the actions of scalar multiplication and addition. Thus, scaled and shifted versions of a filter conform an equivalence class. The resulting quotient space $\hat{\boldsymbol{M}}_i = \boldsymbol{M}_i/\sim := \{[x] : x \in \boldsymbol{M}_i\}$ (*i.e.* the set of equivalence classes) admits a unique manifold structure $\mathbb{S}^{n-1}$ and therefore the most efficient way of traversing it is on angular steps.

7.2 Preliminaries: Optimization on Riemannian manifolds

Consider the problem of optimizing a real-value DNN, but constraining the L_2 norm of the weights to a unit hypersphere $w \subset \mathbb{S}^{d-1}$, or to the more general case of an arbitrary Riemannian manifold $w \subset \boldsymbol{M}^{d-1}$. This is feasible due to the concepts of gradient projection and parameters retraction which will be explained below.

7.2.1 Quotient manifolds

In topology, a quotient or identification space is the result of "stitching together" certain points of a given topological space under an equivalence relationship $\sim$. Formally, let the set $\overline{\boldsymbol{M}} = \boldsymbol{M}/\sim$ be the quotient of the manifold $\boldsymbol{M}$ by $\sim$. In other words, the quotient manifold $\overline{\boldsymbol{M}}$ groups all elements of $\boldsymbol{M}$ in the same equivalence class as a single point given under the relation $\sim$. Let π be the natural projection that associates to each $y \in \boldsymbol{M}$ its equivalence class $\pi(y) = [y] = \overline{y} \in \boldsymbol{M}$. These three notations for equivalence class are interchangeable.

Let $\langle \cdot, \cdot \rangle$ be the Riemannian metric on the tangent space $T_x\boldsymbol{M}$ of the embedding space $\boldsymbol{M}$. The quotient $\overline{\boldsymbol{M}} = \boldsymbol{M}/\sim$ admits a Riemannian structure for the induces Riemannian metric if the metric is compatible with the equivalence relationship $\sim$, if it does not depend on the chosen representative of the equivalence class. The original tangent space $t_x\boldsymbol{M}$ can be decomposed into a vertical $V_x\boldsymbol{M}$ and a horizontal space $H_x\boldsymbol{M}$. The vertical space $V_x\boldsymbol{M}$ at $x \in \boldsymbol{M}$ is the set of all the tangent vectors in $T_x\boldsymbol{M}$ that are tangent to the equivalence class $\overline{x}$. It can also be seen as the set of tangent vectors or movements along which keep x in its equivalence class $\overline{x}$. The horizontal space $H_x\boldsymbol{M}$ at $x \in \boldsymbol{M}$ is the orthogonal complement of the vertical space $V_x\boldsymbol{M}$. It can also be seen as the set of tangent vectors or movements which move x across equivalence classes.

Generally speaking, the field of Quotient manifolds is applicable to any entity that can be described as a set of elements $\boldsymbol{M}$, where it allows us to define a subset $\overline{\boldsymbol{M}}$ with the relationship between the two sets is defined by $\overline{\boldsymbol{M}} = \boldsymbol{M}/\sim$. The operator $\sim$ provides the equivalence relationship between the two sets; and is described by the set operation of *onto* mapping $f : \boldsymbol{M} \twoheadrightarrow \overline{\boldsymbol{M}}$ where a subgroup $\boldsymbol{M}^* \subset \boldsymbol{M}$ is collapsed to a single element in $\overline{\boldsymbol{M}}$. Within the scope of this work, we restrict $\boldsymbol{M} \in \mathcal{R}^n$ and $\overline{\boldsymbol{M}} \in \mathcal{S}^{n-1}$ (the unit hypersphere). The necessary mathematical tools for efficiently realizing operations on Quotient manifolds will be discussed in the following section where we describe Riemannian geometry.

7.3 Training 1-bit DNNs on a Sphere

Given a training dataset X, consisting typically of samples with their corresponding targets drawn in minibatches of n elements, the task of training a 1-bit DNN with binary weights $\mathbf{b} \in \{1, -1\}^d$, is defined as follows:

$$\min_{\mathbf{b}\in\{-1,1\}^d} \boldsymbol{L}(\boldsymbol{X}, \mathbf{b}). \tag{7.1}$$

Optimizing them corresponds to solving an outrageous combinatorial problem with millions of variables. However, it is easy to notice that all the possible weight configurations lie on a $(d-1)$-dimensional sphere. Thus, the obliged question is: *Is it possible to leverage this information to address the problem?* We find the answer in the field of Riemannian optimization, which subsumes hyperspheres optimization along with multiple other differentiable manifolds.

In section 7.2, the concepts of gradient projection, retraction and steps for traversing Riemannian manifolds were explained. In this section we will extend them to the particular case of spheres starting with the tangent space.

The tangent space to S^{d-1} at x is the set of orthogonal vectors to this same point, $T_x S^{d-1} = \{\gamma \in \mathbb{R}^d : x^T\gamma = 0\}$. The orthogonal projection onto the tangent plane is defined as:

$$\mathrm{Proj_x}(\xi) = \xi - \mathrm{xx^T}\xi, \qquad \xi \in \mathbb{R}^\mathrm{d} \tag{7.2}$$

For the retraction there are two options which we compare in our experiments section. A simple choice is normalization to unit norm:

$$\mathrm{Ret}_{\mathrm{x}}(\Delta) = \frac{\mathrm{x} + \Delta}{\|\mathrm{x} + \Delta\|}, \qquad \Delta \in \mathbb{R}^{\mathrm{d}} \tag{7.3}$$

where $\|\cdot\|$ denotes the Euclidean norm in $\mathbb{R}^n$. A second possibility is

$$\mathrm{Ret}_{\mathrm{x}}(\Delta) = \mathrm{x}\cos\|\Delta\| + \frac{\Delta}{\|\Delta\|}\sin\|\Delta\|, \qquad \Delta \in \mathbb{R}^{\mathrm{d}} \tag{7.4}$$

which corresponds to the *exponential mapping*.

Nevertheless, so far we have ignored the second constraint, which is the discreteness of the solution space on the sphere, this is, $\mathbf{b}$ is only allowed to take values in $\{-1, 1\}^d$, making it incompatible with SGD given it relies on infinitesimal steps over smooth surfaces. In order to address this issue and simultaneously to evaluate only feasible network candidates, we can resort to a number of techniques like incremental quantization [164, 177], reformulating the problem as a hashing problem [178] or using proximal operators [83].

In this work we rely on the widely accepted Straight Through Estimator [103] originally employed by [22–24]. At each iteration the network is projected to its nearest valid neighbor by means of the sign$(\cdot)$ function. The loss is evaluated at this point and the gradient is computed. Subsequently the gradient vector is translated to the original location w and an angular update step is taken. The whole process, STE and angular step is depicted in Algorithm 4.

7.3.1 Projecting Filters, Layers and Entire Networks

We tested our approach on filter-wise, layer-wise and network-wise unit norm spheres. Given the large number of parameters in the layer-wise and network-wise cases, the magnitudes of the weights, especially in the deeper layers, will be very small. However unlike techniques like weight-decay, they are bounded on how much they can shrink, and additionally there is no need for hyperparameters. In our experiments we observed that this constraint increased the generalization of the network in the filter-wise and layer-wise cases, but harmed it in the network-wise case. We investigated the conditioning of the network to acquire intuition on this behavior. In section 7.4 we describe the procedure followed to analyze the conditioning and generalization of the network.

Algorithm 4: One training epoch of 1-bit DNN on a sphere

Require: 1-bit DNN weights w
Require: Loss function $\boldsymbol{L}$, epoch t and optimizer
Get epoch learning rate α_t;
for *length(mini-batches)* **do**
  Get mini-batch data x
  Get nearest valid network: $\hat{w} = \text{sign(w)}$
  Forward pass using input x
  Compute loss $\boldsymbol{L}(x, \hat{\mathbf{w}})$
  Compute gradients $\mathbf{g}$ of $\boldsymbol{L}$ w.r.t. $\hat{\mathbf{w}}$
  Transport gradients from $\hat{\mathbf{g}}$ to w
  Add momentum term to $\mathbf{g}$
  if *optimizer == Adam* **then**
    Update gradients moments
    Normalize gradients
  end
  Project gradients $\mathbf{g}$ onto tangent plane of $\mathbf{w}$ using $\text{Proj}_{\mathbf{w}}(\mathbf{g})$
  Compute deltas $\Delta\mathbf{w} = \alpha_t \mathbf{g}$
  Update $\mathbf{w} = \mathbf{w} - \Delta\mathbf{w}$
  Retract $\mathbf{w}$ back onto the sphere using $\text{Ret}_{\mathbf{w}}(\Delta\mathbf{w})$;
end

7.3.2 Relation to Spectral and Lipschitz Normalizations

Our work saves a close relation to spectral normalization presented in [173, 174]. Yoshida and Miyato [173] demonstrated that penalizing the spectral norm of weight matrices in networks reduces the sensitivity to perturbations and subsequently exhibits higher generalizability. The spectral norm a weight matrix $\sigma(\mathbf{M})$ is defined as:

$$\sigma(\mathbf{M}) = \max_{\mathbf{h}:\mathbf{h}\neq\mathbf{0}} \frac{\|\mathbf{M}\mathbf{h}\|}{\|\mathbf{h}\|} \tag{7.5}$$

which corresponds to the largest singular value of $\mathbf{M}$. For any matrix norm $\|\cdot\|$, the spectral norm or spectral radius $\sigma(\mathbf{M})$ of this matrix has as upper bound:

$$\sigma(\mathbf{M}) \leq \|\mathbf{M}\| . \tag{7.6}$$

Therefore, constraining the L_2 norm of $\mathbf{M}$ implicitly constrains the spectral norm. As an extension to [173], Miyato *et al.* [174] applied this technique to stabilize the training of the discriminator in Generative Adversarial Networks (GANs) [179]. This is accomplished by controlling the Lipschitz norm $\|\mathbf{L}\|_{Lip}$ of each of the discriminator's layers $\mathbf{L}$.

In order to control $\|\mathbf{L}\|_{Lip}$, it is sufficient to control the spectral norm of the weight matrix $\sigma(\mathbf{L})$. Thus, normalizing every layer $\mathbf{L}$ by its largest singular value $\mathbf{L}_{SN} = \mathbf{L}/\sigma(\mathbf{L})$ consequently constrains the entire Lipschitz norm of the model.

7.3.3 Discussion

Although the weights in our network are projected onto a unit sphere, the batchnorm layers [136] commonly used in DNNs remain in place, scaling and translating the per channel spheres, providing extra degrees of flexibility to the optimization.

Finally, although the same approach can be applied to the activations of the network, we found no benefit in doing so. Therefore, the activations are not considered in the experiments.

7.3.4 Compatible approaches

Drawing inspiration from the Thomson or Tammes problems (depending on the restrictions), also known as spherical coding or packing, Liu *et al.* [176] investigated the formulation of this problem in the context of DNNs motivated by the task of increasing the discrimination capacity of the networks filters. The approach is formulated as a regularization, termed Minimum Hyperspherical Energy (MHE). Additing a MHE term on the loss encourages the projections onto the sphere of the channel-wise weight vectors to be as evenly spaced as possible. However, using the enterity of the sphere will cause half of the channels to have a negatively correlated or reflected counterpart. Therefore, the authors extended the formulation to operate uniquely on half of the sphere, termed Half-space MHE.

Particularly, in the context of the Thomson problem, the MHE regularization is defined as:

$$R_{MHE} = \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} \frac{1}{\left\| \arccos\left(\hat{\mathbf{F}}_i^T \hat{\mathbf{F}}_j\right) \right\|} \tag{7.7}$$

where $\hat{\mathbf{F}} = \frac{\mathbf{F}}{\|\mathbf{F}\|}$. Note that the MHE loss term does not push the network filters to lie on the sphere. In fact, it only penalizes the angular distance of their projections in order to encourage diversity, which supports our rationale on angular learning encoding semantic information.

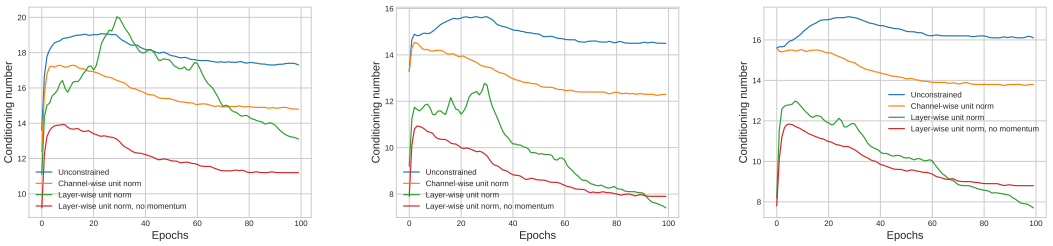


FIGURE 7.1: Conditioning number for layers 3, 8 and 13 across training epochs on a Resnet-18 network trained on Tiny-Imagenet.

This formulation is naturally complementary to our approach; thus, in order to explicitly prevent filter redundancy we extend our scheme based on this principle. We modify our task loss function to encourage the channels to be evenly spaced, while strictly constraining each vector to lie on the unit sphere unlike MHE [176]. Formally, we redefine the loss in equation 3.3 to include this component with weighting η:

$$\boldsymbol{L}(\boldsymbol{X}, \mathbf{b}, \mathbf{b}_{aux}) = \boldsymbol{L}(\boldsymbol{X}, \mathbf{b}) + \eta R_{MHE}(\mathbf{b}_{aux}), \tag{7.8}$$

where $\mathbf{b}_{aux} \in \{1, -1\}^d$ is the background full-precision weights of the binary network.

7.4 Conditioning Analysis

In the optimization field the condition number is commonly measured by taking the ratio and between the largest and smallest singular values of the Hessian matrix: $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$ of the function being optimized, with $H = \mathbb{E}[J^T J]$ and J, the Jacobian matrix, is computed per example in a batch of samples, and subsequently the Hessian matrix is computed by taking the mean over the samples. Note this is an approximation of the Hessian, where the high-order values are neglected and we focus on the cross-product of the first order derivatives known as the Gauss-Newton matrix.

The condition number provides a good estimate for the density of pathological curvatures which exist in the loss surface. These can also have the interpretation of erratic local minimas which cause effects such as vanishing gradients. Therefore, imposing a well-conditioned Hessian matrix on the loss surface behaves as a smoothing process which eliminates local minima and mitigates common optimization issues that arise when optimizing deep neural networks. We modify the computation of the condition number

by following the method proposed in [172]. We initially compute the spectrum of the Hessian matrix as:

$$\rho_H(\lambda) = \frac{1}{n}\sum_{i=1}^{n}\delta(\lambda - \lambda_i(H)) \tag{7.9}$$

following, we compute the conditioning of the Hessian as the ratio between the second and first moments of the spectrum $\rho_H(\lambda)$ of the Hessian as, $\kappa = \frac{m_2}{m_1^2}$. This form of computing the condition number is more robust than the standard approach as it does not suffer from Hessian matrices that are populated with many low singular values. It provides the density measure of the spectrum and effectively measures the amount of erratic directions in which the loss surface diverges from the mean direction, the global minimum of the loss surface.

In Figure 7.1, we show the conditioning numbers across training for layers 3, 8 and 13 of a ResNet-18 network trained on Tiny-ImageNet. All the methods start from randomly sampled weights using standard initialization schemes. During the initial iterations they all bounce and subsequently stabilize. Layer-wise unit spheres appear to be very sensitive to momentum as can be appreciated. Both modes of our spherical constraints, channel-wise and layer-wise, improve the conditioning of the layers significantly. Notably layer-wise with no momentum provided the lowest and most stable curves, particularly in the early layers.

7.5 Experiments

In this section, we tested our approach on a ResNet-18 [6] architecture in the task of image classification on a large-scale dataset, ImageNet (ILSVRC-2012) [2]. We projected the auxiliary weights onto channel-wise spheres and for binarization we used the procedure in BinaryConnect (BC) [22] as the base algorithm.

We implemented our training scheme using the Pytorch framework. A pretrained ImageNet network taken from the official Pytorch model zoo was used as initialization for the full-precision weights in all the cases. As common convention, the first and last layers were not quantized.

7.5.1 Evaluation on ImageNet

The ImageNet 2012 dataset contains 1.2M training and 50k validation labeled images spread across 1000 categories. Top-1 and Top-5 predicted classes in descending order of confidence are provided. Only the Top-1 is used to compute the loss.

TABLE 7.1: Top-1 validation accuracy (%) for ResNet-18 on ImageNet. LS = Learnable Scalings

Method	Bitwidth(W/A)	Top1	Top 5	Width(x)	Top1	Top5
BWN [24]	1/32	60.8	83.0	-	-	-
TWN [180]	2/32	61.8	84.2	-	-	-
BC [22] + LS	1/32	61.4	83.7	-	-	-
BC [22] + LS + L_2	1/32	61.8	84.2	-	-	-
BC [22] + LS + angle (ours)	1/32	60.5	83.1	-	-	-
1-bit-per-weight [181]	1/32	-	-	2.5	68.97	88.49
Spherical projection (ours)	1/32	61.2	83.9	2.5	-	-
ASQ (ours)	1/32	62.7	84.8	2.5	-	-
ASQ + MHE (ours)	1/32	60.2	82.6	2.5	-	-
Spherical projection + ASQ (ours)	1/32	62.7	84.8	2.5	-	-
Spherical projection + MHE (ours)	1/32	61.5	83.8	2.5	-	-
Spherical projection + ASQ + MHE (ours)	1/32	**62.9**	**84.9**	2.5	-	-

We used different training regimes depending on whether ASQ was used or not. For the networks trained with ASQ, we used SGD with initial learning rate of $1e^{-3}$ and weight decay of $1e^{-4}$ with angle penalization hyperparameter $\beta = 5$, and multiplied by a factor of 1.3 at every epoch. The learning rate is updated by a factor of 0.1 at epochs 20, 25 and 30. For networks without ASQ we used ADAM [65] with an initial learning rate of $1e^{-5}$ and weight decay of $1e^{-6}$.

In Table 7.1 we report our ImageNet results for 1-bit weights and full-precision activations on Resnet-18. We experimented using channel-wise learnable scalings and replacing our angle penalization with a L_2 distance penalization on BC. Our approach outperformed BC and BWN. Additionally, we performed ablations for the spherical normalization, angle penalization, or ASQ, and minimum hyperspherical energy. Finally, we performed tests in a WideResnet [182] as done by McDonnell in [181].

7.6 Conclusion

In this chapter a novel geometric perspective on 1-bit DNNs optimization was presented. By projecting the auxiliary weights onto channel-wise spheres and formulating

the problem of quantization as an angle based regularization, we were able to improve the performance of these networks. We additionally integrated our approach with a complementary scheme based on the Thomson problem. Finally, we delved into the operation of the network, particularly observing a superior conditioning number of the weight matrices attained by our method, leading to improved training. We validate our results on the large-scale dataset Imagenet obtaining competitive results against already well established baselines. We performed ablation studies with the combinations of the proposed ideas and obtained 62.7% accuracy on our best performing 1-bit DNN.

Chapter 8

Flynet

8.1 Introduction

In this chapter we present a new efficient Convolutional Neural Network (CNN) architecture targeted for tiny machine learning (TinyML) on vision tasks. Through a series of architectural improvements to state-of-the-art mobile networks, we are able to significantly reduce the number of parameters while maintaining a reasonable number of multiply-accumulate operations. We switch the load from expensive pointwise convolutions into lighter multihead-depthwise convolutions with non-linear Max-out aggregation. By incorporating our contributions to a MobileNetV3 backbone, we achieved comparable accuracy with up to 0.5x reduction in parameters in the ImageNet dataset, and achieved 61% top-1 accuracy, matching MicroNet-M2, ShufflenetV2-0.5x and EfficientNet-B, with 1MB. We additionally reported results in the tasks of object detection in the COCO dataset. Finally, we performed ablation studies to demonstrate the effectiveness of our improvements.

8.2 Related Work

Efficient Building Blocks. With the intention of reducing computation and storage in CNNs, several works in literature have proposed different neural building blocks. An initial attempt was the depthwise separable convolutions presented in the Inception series [111, 112] and MobileNets [30, 32, 113], followed by SqueezeNet's fire module [31]. Group

convolutions were utilized in ResNeXt [114] along with channel shuffling in ShuffleNet [33, 52]. The linear bottleneck was introduced in MobileNetV2 [32]. Finally, residual connections [6, 115], although initially conceptualized to deal with vanishing gradients, have been utilized as means to obtain increases in performance with little overhead as in the sandglass block [116]. In this work a new block denoted *multihead-depthwise convolutions* is introduced, which we use extensively across FlyNet, along with dense lightweight residuals. Recently this block, also denoted generalized depthwise-separable convolution, along with concatenation was used for adversarially robust and efficient networks [183].

Dynamic Convolution and Mixture of Experts. Our method holds similarity to mixture of experts based models [184–186] where the outputs of different convolutional filters are fused by a gating or weighting mechanism with the main difference being that our fusing function is a differentiable non-linearity. In contrast, the more recently proposed dynamic convolutions [187, 188] adaptively adjust the convolutional filters prior to performing the convolution operation. Thus, the number of operations remains almost unchanged; however, the network grows considerably in size which is non-trivial given that typically memory accesses account for a considerable amount of power consumption [189]. Our approach differs from dynamic convolution in the sense that the filter responses are computed and aggregated via a Max-out activation function with squeeze-and-excitation [190] scalings. We only implement this strategy in depthwise convolutions, offloading the pointwise convolutions in MobileNet.

Efficient Attention. Squeeze-and-Excitation networks (SENet), introduced in [190], pioneered a whole set of attention based inexpensive architectural improvements including [191–196]. Recently [197] proposed replacing the multi-layer perceptron (MLP) in SENet for a single convolution. In this work we endow this lightweight channel attention with variance awareness, translating in accuracy gains at little extra overhead. Second-order statistics with channel-wise attention had been considered before in [198]; however, considerable complexity is added for similar gains to our approach as the authors consider the entire channels covariance matrix.

Improving Generalization by Injecting Noise. Over-fitting to the training data is an issue that has been extensively explored from different angles [69–71, 199, 200]. Dropout [67] has been modelled as a Gausssian noise injection process during the learning

process [76]. Batch-norm [136], originally believed to reduce the internal covariate shift was recently shown to be successful in part due to the same noise-injecting principle in the form of normal additive noise and scaled inverse Chi multiplicative noise which depend on the batch size [78]. Here, we present noise regularization based on features and parameters quantization. To the best of our knowledge, this is the first time quantization has been approached from a regularization perspective.

8.3 Motivation

Common approaches to tackle the problem of efficiency in neural networks involve reducing the amount of Multiply-Accumulate (MACs) operations [52, 201, 202]; however, reducing the number of parameters has, to some extent, been overlooked. Nevertheless, as described by Xu *et al.* [203], transmitting model parameters across different hierarchies of memories in an embedded device can account for up to 80% of the power consumption of an inference routine. For example, a commercial microcontroller might not fit a state-of-the-art efficient network in on-chip memory [54].

Similarly, a network designed for constant operation on an always-on device, and a network that will be loaded on demand by a cell-phone app to perform a single inference cycle, will have different energy profiles. The first one might be better suited for a low FLOPs model, whereas the second one for a low parameters model.

With respect to our architecture of choice, currently there is an ongoing paradigm switch in the field of deep learning migrating from once ubiquitous CNNs to Transformer-based [41] neural architectures [45, 204, 205] and MLP-only variants [206, 207]. Nevertheless, convolutional backbones are still leading the board in parameter efficiency [52, 201, 208].

8.4 FlyNet

In this work we propose four simple but highly effective architectural modifications to mobile networks that can be used as building blocks in further architecture exploration. Concretely, we switched the load from the expensive pointwise convolutions to the much lighter *multihead-depthwise convolutions* which generalizes depthwise convolutions. By additionally using a $\max(\cdot)$ activation function as feature aggregation, our

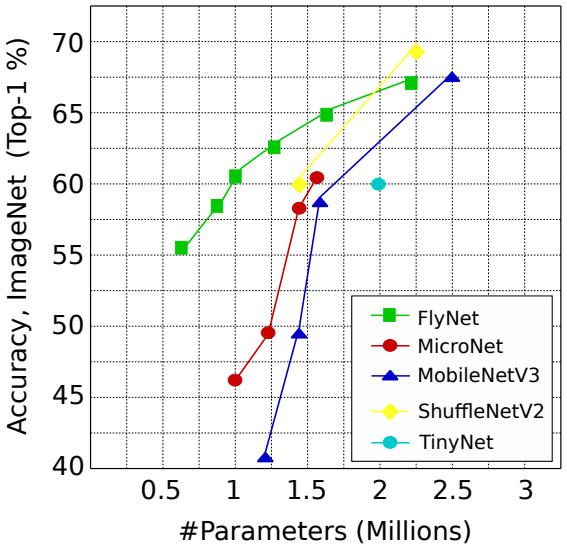


FIGURE 8.1: Comparison of FlyNet-h3 (3 MDW heads) architecture with different width multipliers against state-of-the-art models. FlyNet performs favorably in terms of parameter count against both popular and recently published works. Particularly, our method excels as the model shrinks, towards the extremely low parameter regime.

method is equivalent to a Maxout network [209] with depthwise convolutions. Additionally, we added efficient mean and variance aware channelwise attention. We leveraged dense residual connections, in contrast to previous MobileNet versions [32, 113] by simply cropping and padding where necessary, and finally we regularized our networks by injecting quantization noise to improve generalization. These contributions grant us significant accuracy gains over already well established mobile architectures pushing the barrier towards extremely low power and highly efficient CNNs for embedded devices. For reference see Figure 8.1.

The network developed in this chapter holds similarity to mixture of experts based models [184–186] where the outputs of different convolutional filters are fused by a gating or weighting mechanism with the main difference being that our fusing function is a differentiable non-linearity. In contrast, the more recently proposed dynamic convolutions [187, 188] adaptively adjust the convolutional filters prior to performing the convolution operation. Thus, the number of operations remains almost unchanged; however, the network grows considerably in size which is non-trivial given that typically memory accesses account for a considerable amount of power consumption [189]. Our approach differs from dynamic convolution in the sense that the filter responses are computed and aggregated via a Max-out activation function with squeeze-and-excitation [190] scalings.

We relied on MobileNetV3 [113] as base architecture for implementing our contributions

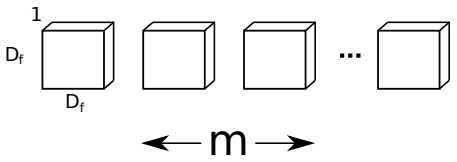


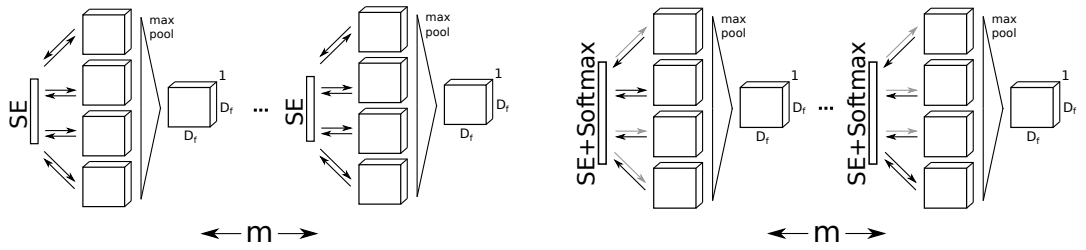
FIGURE 8.2: Standard depthwise features

described in the following subsection. Then we enlist our contributions and elaborate on each of them.

8.4.1 MobileNetV3 Backbone and Analysis

Popular compressed architectures such as MobilNetV3 and EfficientNet [202] based themselves on the MobilenetV2 [32] architecture, consist of a stack of depthwise separable convolutions and linear bottlenecks. Depthwise separable convolutions are comprised by depthwise and pointwise convolutions which is a form of factorization of a standard convolution. Depthwise convolutions, implemented as *3x3 groups=1* convolutions, have the function of finding spatial patterns but have no cross-channel communication. Pointwise convolutions, implemented by *1x1 groups=m* convolutions, have the function of mixing information across feature channels. The linear bottlenecks are implemented as two sequential pointwise convolutions that compress and expand the features with no activation function. Finally, inverted residuals connect the compressed representations in the linear bottlenecks allowing gradients to flow better throughout the network.

Borrowing notation from [32], a convolutional layer has the following parameters: D_k^2, D_f^2, m and n that denote the dimension of a convolutional kernel, dimension of the input feature map, number of input channels and number of output channels, respectively. Depthwise convolutions have a computational cost of $D_k^2 \cdot m \cdot D_f^2$, and require $D_k^2 \cdot m$ weights. Pointwise convolutions have a computational cost of $m \cdot n \cdot D_f^2$, and require $m \cdot n$ weights. It is easy to see that pointwise convolutions are far more expensive than depthwise ones, especially as m and n grow with the depth of the network. Based on these observations we propose *multihead-depthwise convolutions* to offload pointwise convolutions while attempting to maintain the representation capacity of the network.



(a) Multihead-depthwise (MDW) features with Max-out aggregation

(b) MDW with softmax excitation

FIGURE 8.3: (a)Act as feature map **mixer**. (b)Additionally to the traditional squeeze-and-excitation mechanism that scales the feature maps by the computed coefficients, here the coefficients are passed through a softmax before scaling. This operation can be thought of as a feature map **multiplexer**.

8.4.2 Architectural Contributions

Multihead-Depthwise Convolutions with Max-out Activation.

Multihead-depthwise (MDW) convolutions are a generalization of depthwise convolutions that leverage the over-looked output channels dimension (output depth). Whereas traditionally in a depthwise convolution each input channel gets convolved with only a single filter, in a MDW convolution it gets convolved with $h \geq 1$ filters. The resulting $m \cdot h$ output feature maps can subsequently be merged in several ways. A naive approach would involve using a pointwise convolution to shrink the dimension back down to the original dimension (denoted input depth in [30]). An appealing second alternative is to linearly combine each set of h heads produced by each input channel using a weighted combination followed by an activation function (*e.g.* relu$(\cdot)$). This approach incurs in fewer parameters and computation. However, in this work we propose to use the *Max-out* activation function proposed in [209], which is a parameter-free, non-linear aggregation, involves no Multiply-Add operations (MAdds) and achieves higher accuracy than a learned affine transformation. Refer to Table 8.1 for ablations on types of reductions and section 8.6.3 on the number of heads.

Formally, given an input features tensor $\mathbf{A} \in \mathbb{R}^{m \times f_1 \times f_2}$, MDW convolution convolves it with a h-heads kernel $\mathbf{W} \in \mathbb{R}^{m \cdot h \times k_1 \times k_2}$ to produce pre-activations $\mathbf{F}_{pre} \in \mathbb{R}^{m \times h \times f_1 \times f_2}$ and subsequently reduce them with $\max_{\text{dim}=1}(\mathbf{F})$ to produce post-activations $\mathbf{F}_{post} \in \mathbb{R}^{m \times f_1 \times f_2}$. This is illustrated in Figure 8.3. As in [210], in order to implement a *Funnel Max-out* activation, we additionally tested adding the input feature maps $\mathbf{A}$ as input to the Max-out activation: $\max_{\text{dim}=1}([\mathbf{A}, \mathbf{F}_{\text{pre}}]_{\text{dim}=1})$ with no success. $\max_{\text{dim}=\text{j}}(\cdot)$ and

$[\cdot]_{dim=j}$ denote max pooling and concatenation across dimension j with dimensions indexing starting at zero. Refer to section 8.4.2 for the results.

The computational cost of MDW convolution is simply h times that of a regular depthwise convolution. By transferring the load from the pointwise convolutions to MDW convolutions our network achieves high compression rates without incurring in a significant decrease in accuracy. We conjecture that our network makes better use of the redundant feature maps produced by the pointwise convolutions. A similar observation was done by Han *et al.* [211].

Efficient, Mean and Variance Aware Channel Attention. Squeeze-and-Excitation networks [190] (SENet) was a pioneer in proposing channel attention. The purpose of the SENet is to model interactions between channels of intermediate activations and re-scale them appropriately. Their approach involves adding an MLP per convolutional layer in order to compute conditional channel-wise scalings. The scalings $\boldsymbol{s} \in \mathbb{R}^n$ are obtained from the channel-wise means $\boldsymbol{\mu} \in \mathbb{R}^n$ in the following way:

$$\boldsymbol{s} = \text{MLP}(\boldsymbol{\mu}), \qquad \boldsymbol{\mu}_n = \frac{1}{f_1 f_2} \sum_{f_1} \sum_{f_2} \mathbf{A}_{n,f_1,f_2} \tag{8.1}$$

ECA-Net [197] replaced this MLP with a single 1D convolutional layer parameterized by $\boldsymbol{k} \in \mathbb{R}^z$: $\boldsymbol{s} = \boldsymbol{\mu} * \boldsymbol{k}$. $z \in \mathbb{Z}^+$ is the kernel size and good results can be obtained with kernels as small as $z = 3$. For our experiments, we empirically found that a kernel of size $z = 7$ provided the best results.

Given that the number of parameters is drastically reduced by replacing the MLP with a convolution, we augmented the ECA-Net formulation by adding the channel-wise variances $\boldsymbol{\sigma}^2 \in \mathbb{R}^n$ to the scalings computation which reuses the already computed means. We concatenate the mean and variance vectors on a new dimension and convolve with $\boldsymbol{k} \in \mathbb{R}^{z \times 2}$:

$$\boldsymbol{s} = [\boldsymbol{\mu}, \boldsymbol{\sigma}^2]_{dim=1} * \boldsymbol{k}, \qquad \boldsymbol{\sigma}^2_n = \frac{1}{f_1 f_2} \sum_{f_1} \sum_{f_2} (\mathbf{A}_{n,f_1,f_2} - \boldsymbol{\mu}_n)^2. \tag{8.2}$$

Finally, a non-linearity is applied on the computed scalars $\boldsymbol{s}$ prior to exciting the n output channels. As depicted in Figure 8.3 we tested both sigmoid (default in SENet) and softmax nonlinearities. The latter one is only applicable for MDW convolution

layers. The softmax is implemented by first reshaping $\boldsymbol{s} \in \mathbb{R}^n$ into $\boldsymbol{s} \in \mathbb{R}^{m \times h}$ where n denotes the number of output channels, and m denotes the number of input channels, with $n = m \times h$. Then the softmax is taken across the heads dimension:

$$\boldsymbol{s}_{sig} = \text{sigmoid}(\boldsymbol{s}), \qquad \boldsymbol{s}_{sof} = \text{softmax}(\boldsymbol{s}/T)_{dim=1}. \tag{8.3}$$

where T is the temperature hyperparameter. If used with MDW, the sigmoid case can be thought of as a channel mixer, while the softmax case can be thought of as a feature map selector or multiplexer.

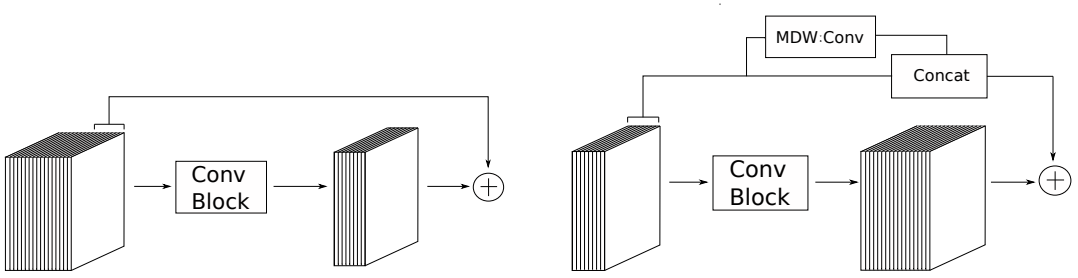
Including second-order information in the channel-wise attention has been previously explored in [198]; however, the authors compute a per-layer channels covariance matrix which is considerably more expensive than our approach.

TABLE 8.1: FlyNet ImageNet accuracy with different activations and channel reductions in the MDW convolutions. All the models have the same number of parameters and similar MAdds.

Model	Activation	Reduction	Top-1 Acc	Top-5 Acc
FlyNet-h3 0.5x	ECA+Sigmoid	Sum+Relu	57.6	80.5
FlyNet-h3 0.5x	ECA+Sigmoid	Funnel Max-out	58.8	80.9
FlyNet-h3 0.5x	ECA+Sigmoid	Max-out	58.9	81.0
FlyNet-h3 0.5x	ECA+Softmax(T=3)	Max-out	58.9	81.0
FlyNet-h3 0.5x	ECA+Softmax(T=10→ 1)	Max-out	**59.1**	**81.3**

In Table 8.1 we performed experiments on a network with $h = 3$ heads to compare the different combinations of excitation activations and feature map reductions. We can observe that Softmax with temperature annealing followed by Max-out reduction performed slightly better than the other strategies.

Dense Light Residuals. Unlike a traditional ResNet that uses residual connections [6, 115] as shortcuts between high dimensional representations, MobileNetV2 and V3 make use of them to connect its low dimensional features in the linear bottlenecks, and are thus named inverted residuals. Whenever there is a mismatch in the number of input and output channels, ResNet uses 1x1 convolutions to adjust the number of channels; however, this incurs in a significant number of weights and MAdds, and therefore are avoided in MobileNet by simply not placing them. Here, as depicted in Figure 8.4 we propose an alternative depending on whether the number of channels should increase or



(a) Residuals for channel decrease (b) Residuals for channel increase

FIGURE 8.4: Light residuals for channel mismatch. Unlike ResNet [6] which fully relies on 1x1 convolutions to adjust the number of channels in mismatching residual connections, we use a lighter version that compliments only for the channels shortfall. Both cases use Depthwise convolutions for spatial down-sampling when required.

decrease:

$$\mathbf{R} = \begin{cases} \mathbf{A}^{l-1}_{0:n,:,:} & if \quad m > n \\ [\mathbf{A}^{l-1}, \text{MDW}(\mathbf{A}^{l-1})]_{dim=0} & if \quad m < n \end{cases}, \tag{8.4}$$

where $\mathbf{R}$, the residual connection, now matches the dimension of $\mathbf{A}^{l-1}$, and l is the layer index. MDW is used to both fill the residuals shortfall and spatial downsampling.

In the case of $m < n$, analogously to the case $m > n$, we tested only bridging the available input channels with residual connections to the first n channels, however this strategy did not provide improvements.

TABLE 8.2: FlyNet ImageNet accuracy with light residual connections.

Backbone	Residual	Top-1 Acc	Top-5 Acc	Params	Madds
FlyNet-h3 0.4x	No Residual	54.5	78.0	0.657M	26.5M
FlyNet-h3 0.4x	Default in MobileNetV3	55.5	78.8	0.657M	26.5M
FlyNet-h3 0.4x	3x3 Conv	56.0	79.3	0.673M	30.9M
FlyNet-h3 0.4x	3x3 DW - PW	55.6	78.9	0.659M	27.5M
FlyNet-h3 0.4x	3x3 MDW - Concat	55.9	79.2	0.658M	27.0M

Following the analysis from the Sandglass block [116], we added dense residuals to the MobileNet architecture by bridging both the low dimensional representations and the high dimensional representations. As illustrated in the ablation studies detailed In Table 8.2, we achieved increased accuracy and generalization at practically no overhead.

8.5 Quantization as regularization

Over-fitting to the training data is an issue that has been extensively explored from different angles [69–71, 199, 200]. Dropout [67] has been modeled as a Gaussian noise injection process during the learning process [76]. Batch-norm [136], originally believed to reduce the internal covariate shift was recently shown to be successful in part due to the same noise-injecting principle in the form of normal additive noise and scaled inverse Chi multiplicative noise which depend on the batch size [78]. Here, we present noise regularization based on features and parameters quantization. To the best of our knowledge, this is the first time quantization has been approached from a regularization perspective.

Quantized networks have been observed to harm training accuracy at the expense of a reduction in computational resources, but present high generalization due to the highly restrictive permissible set of values, preventing them from over adjusting to the data. Therefore, we propose training with low quantization noise by manually tuning the bitwidth/resolution of the network.

Restrictive codebooks and clipping functions have been empirically observed to harm accuracy [212]. Here we use adaptive quantization, meaning that no clipping is performed and the codebook is dynamic. This is because no codebook will be used during inference; thus, it can vary across training iterations. Similarly, the quantization levels q do not have to be powers of 2.

Analogously to Dropout being modeled as multiplicative Gaussian noise on the weights [76], a q-level uniform quantization function $Q(\cdot)$ as follows:

$$Q(x) = \Delta \cdot (\text{round}(\frac{\text{x}}{\Delta})), \qquad \Delta = \frac{1}{\text{q} - 1} \tag{8.5}$$

can be modelled as additive noise $x + \epsilon$ with an uniform distribution $\epsilon \sim \mathbb{U}(-\Delta/2, \Delta/2)$, where x has been re-scaled to the range $[0, 1]$. Refer to Banner *et al.* [155] for details.

In order to back-propagate through the non-differentiable rounding function, we resort to the Straight Through Estimator (STE) originally proposed in [103], and employed ubiquitously across quantization works [22–24, 135]. The STE is defined as: $\frac{dL}{dx} \approx \frac{dL}{dQ(x)}$, where L denotes the loss function.

We inserted our quatization regularization as a drop-in replacement for Dropout in the MobileNetV3 backbone. As initialization, we used a pre-trained full-precision network. We swept different quantization levels in section 8.6.3.

8.6 Experiments

8.6.1 Evaluation on ImageNet

We conducted experiments in the ImageNet ILSVRC 2012 [2] dataset which consists of 1.2M labeled images for the task of classification with 50K test images spread across 1000 classes. The image resolution used for training and computing the number of MAdds is of 224^2. For testing we first resize the smallest size of the image to 266 pixels before doing a single center crop of 224^2.

We used the momentum SGD optimizer. For the learning rate we used a cosine schedule starting at $lr = 0.4$ and momentum=0.2 during 150 epochs with mini-batch of 64. We observed that both Dropout and quantization regularizations harmed the accuracy on small networks (*e.g.* FlyNet 0.5x), therefore we only applied our regularization on models above 1.6M parameters. Similarly, weight decay was set to $1e^{-5}$ for models with #Params<1.6M and $4e^{-5}$ for the remaining ones.

The results of FlyNet on ImageNet are listed in Table 8.3. We can see that FlyNet is very efficient towards the extremely low parameter regime, with FlyNet-h3 0.4x (3 heads and width multiplier of 0.4) widely outperforming MobileNetV3 0.15x and MicroNet-M0.

FlyNet effectively reduces the number of parameters while maintaining a reasonable number of MAdds in comparison to EtinyNet [203], since both memory accesses and operations can be expensive in terms of energy consumption as discussed in [189] and [22].

We compared against a downscaled version of EfficientNet [202] which does not perform very well. However, it is important to notice that EfficientNet was designed to be an efficient network in the high accuracy range (acc.>77%). This supports our intuition that networks designed with some constraints in mind do not necessarily perform well under different constraints, and there is a need for networks targeted specifically for

TABLE 8.3: FlyNet performance results in ImageNet.

Model	Top-1 Acc	Top-5 Acc	Params	MAdds
MobileNetV3 0.15x[201]	33.7	57.2	1.0M	4M
MicroNet-M0 [201]	46.6	70.6	1.0M	4M
FlyNet-h3 0.4x	**55.9**	**79.2**	0.65M	26M
MobileNetV3 0.2x[201]	41.1	65.2	1.2M	6M
MicroNet-M1#	49.4	72.9	1.2M	5M
MicroNet-M1	51.4	74.5	1.8M	6M
EfficientNet-B [202]	56.7	79.8	1.3M	24M
FlyNet-h3 0.5x	**59.1**	**81.3**	0.86M	34M
MobileNetV3 0.35x+BFT [213]	55.2	-	1.4M	15M
MobileNetV3 0.5x [113]	58.0	-	1.6M	21M
MicroNet-M2#	58.2	80.1	1.4M	11M
MicroNet-M2	59.4	80.9	2.4M	12M
TinyNet-E [214]	59.9	81.8	2.0M	24M
ShuffleNetV2 0.5x [52]	60.3	-	1.4M	41M
FlyNet-h3 0.6x	**61.5**	**83.4**	1M	46M
ShuffleNetV2 0.5x+BFT [213]	61.3	-	1.4M	41M
MicroNet-M3#	61.3	82.9	1.6M	20M
FlyNet-h3 0.7x	**63.3**	**84.5**	1.3M	54M
MicroNet-M3	62.5	83.1	2.6M	21M
Mobile-Former-26M [215]	64.0	-	3.2M	26M
EtinyNet [203]	65.5	86.2	0.98M	117M
FlyNet-h3 0.8x	**65.7**	**86.2**	1.6M	66M

high parameter compression. For example in the case of FlyNet, in Figure 8.1 we can observe that the FlyNet models draw a curve whose growth decreases with the number of parameters.

Finally we compared against ShuffleNetV2 [52], ShuffleNetV2 augmented with Butterfly transform [213], GhostNet [211] and other models aimed at relatively high accuracy, performing competitively in this ranges.

8.6.2 Evaluation on COCO Object Detection

COCO [216] object detection is large-scale dataset consisting of 80K training and 40K validation images with annotated boxes for 90 different classes.

We implemented our FlyNet backbone in the Faster-RCNN and RetinaNet frameworks compatible with MobileNetV3 without any modifications. We used pretrained backbones and fine-tuned for 26 epochs with multi-step learning rate starting at $2.5e^{-3}$, multiplied by a factor of 0.1 at epochs 16 and 22. We set weight decay to $4e^{-5}$ and momentum 0.9.

We reported Average Precision ($AP^{0.5}$) as well as the number of parameters and MAdds (computed using 224×224 images) of the backbones (ignoring the heads) used as drop-in

TABLE 8.4: FlyNet performance in COCO object detection. We reported the number of parameters and MAdds of the backbones used as drop-in replacement.

Backbone	DET Framework	Params	MAdds	mAP
FlyNet-h3 0.4x	RCNN	0.14M	26M	21.9
FlyNet-h3 0.5x		0.19M	34M	23.2
MicroNet-M2		0.58M	12M	22.7
FlyNet-h3 0.6x		0.24M	46M	24.4
MobileNetV3 1.0x		0.89M	56M	25.9
MicroNet-M3		0.69M	21M	26.2
FlyNet-h3 0.8x		0.41M	86M	**27.0**
FlyNet-h3 0.4x	RetinaNet	0.14M	26M	22.9
MicroNet-M2		0.58M	12M	22.6
FlyNet-h3 0.5x		0.19M	34M	23.7
MobileNetV3 1.0x		0.89M	56M	24.0
FlyNet-h3 0.6x		0.24M	46M	24.6
MicroNet-M3		0.69M	21M	25.4
FlyNet-h3 0.8x		0.41M	86M	**27.2**

replacement. FlyNet outperforms both MicroNet and MobileNetV3 as reported in [201] in terms of the number of parameters. In the low parameter regime, FlyNet-h3 0.5x performs almost 3 times as many MAdds as MicroNet-M2 but is 3 times smaller with higher accuracy.

8.6.3 Ablation Studies

In this section we test the individual contribution of our ideas. Some ablations have been included in the corresponding sections in the main body of the paper.

Number of MDW convolution heads. In Table 8.5 we investigate the impact of the additional heads in the MDW convolutions. We tested with heads in the range $[1, 5]$. We observed a monotonic increase in accuracy for all of them, with a total gain of more than 3% from its single head counterpart at almost no overhead in parameters and less than twice the operations. We also observed that the largest accuracy jump comes from the first additional head at very little cost.

TABLE 8.5: FlyNet ImageNet accuracy for different number of MDW heads.

Model	Heads	Top-1 Acc	Top-5 Acc	Params	MAdds
FlyNet 0.5x	1	56.7	79.6	0.80M	24M
FlyNet 0.5x	2	58.2	81.0	0.83M	29M
FlyNet 0.5x	3	59.1	81.4	0.86M	34M
FlyNet 0.5x	4	59.8	81.9	0.89M	39M
FlyNet 0.5x	5	**59.9**	**82.0**	0.92M	44M

Quantization levels. As common convention, we initiliazed the quantized networks with a pretrained full-precision network, and we set the momentum to 0.9. We did not test training from scratch. As previously mentioned, we found regularizing small networks harms their performance; therefore we tested our regularization on the largest version of FlyNet (1.0x).

TABLE 8.6: FlyNet ImageNet accuracy for different quantization levels (q-levels)

Model	q-levels	Top-1 Acc	Top-5 Acc
FlyNet-h3 1.0x	2^{32}	67.2	87.5
FlyNet-h3 1.0x	16	67.3	87.7
FlyNet-h3 1.0x	8	**67.4**	**87.8**
FlyNet-h3 1.0x	6	67.1	87.4
FlyNet-h3 1.0x	4	66.1	87.1

8.7 Conclusions

In this chapter we developed a series of simple but effective architectural modifications that can be integrated into any neural architecture to provide accuracy boosts at very little overhead. Particularly, our contributions are aimed at compressed networks in the extremely low parameter regime (sub 1M). We leveraged a MobileNetV3 backbone to devise the FlyNet architecture. We performed ablation studies and experiments on different large-scale benchmarks to analyze the impact of our contributions. We believe our research will set grounds for further automatic architecture search aimed at mobile and tiny machine learning.

Chapter 9

Conclusions and Future Research

In this thesis, we have described the state of deep learning for resource-constrained training and inference, along with works advancing the field published during the duration of my studies. Techniques such as pruning, quantization and architecture design were investigated in each of the chapters, both independently and in conjunction. Dynamic computation was also investigated in the context of on-demand processing.

As discussed in 4, these techniques are not mutually exclusive and their strengths could be aggregated in different ways. There is no such thing as an ideal lightweight network since the design is usually dictated by both the application and the hardware.

Throughout this thesis, the theoretical contributions have been highlighted while some implementation details have been omitted as not being part of the core idea. Nevertheless, it's important to point out that often idea and implementation go hand-to-hand. Here, we mention some considerations to be taken into account when attempting to obtain real-world measurable performance boosts from these compression techniques:

- Sparsification is not commonly supported natively in hardware. Therefore, pruning entire structures is the prevailing approach to obtain efficiency gains. Nevertheless, nowadays NVIDIA GPUs support 2:4 (50%) **sparsity pattern**. Research has been done to train networks with this configuration [217]. Additionally, this sparsification step is automatically performed when using the **TensorRT** library.
- The latest NVIDIA GPUs support different data types of reduced precision including bfloat16, a custom implementation of float16 with more bits for the exponent

and fewer bits for the mantissa. Additionally, Int8 and Int4 data types are supported. Similarly to the aforementioned mechanism, **integer quantization** is done automatically when using TensorRT. This type of quantization has not been attempted in this thesis. Nevertheless, it is compatible with the ideas presented, with the exception that a different quantizer is utilized [218].

- Custom codebooks, used extensively here by means of the DoReFa quantization, can be implemented with fixed-point arithmetic. **Fixed-point** numerical representations are an alternative to floating-point representations for storing and operating with real numbers. In fixed-point representations, the position of the decimal point is fixed. Therefore they are suited for applications where the range and precision requirements are known beforehand. Since fixed-point arithmetic is simpler, it is commonly used in embedded devices, signal processing and resource-constrained applications.

- Binary weights and binary activations networks efficiently implemented using only additions and logical operations, respectively, can be implemented using available open-source libraries such as **daBNN** [219].

Optimizing computational resource utilization in neural networks, both for training and inference, will continue to be a very active field of research with high industry demand as AI applications become ubiquitous and start experiencing backlash from improved **classical algorithms** and **tailored solutions**. Some examples are:

- For tabular data, classical tree boosting algorithms such as **XGBoost** [220] are still preferred over neural networks due to their computational efficiency and better performance on small datasets.

- Large Language Models (LLM) initially trained on huge datasets to act as holistic solutions, have been recently shifting to a new paradigm of information retriever and analyzer. This paradigm denoted Retrieval Augmented Generation (RAG) [221] consists of embedding data in latent vectors and storing them in **vector databases** powered by **K-NN**. When a new query is requested, a similarity search is performed in the vector database between the query and the stored vectors. Then the corresponding data sources are retrieved and analyzed by the

language model which provides answers grounded on these sources. This mechanism additionally addresses other concerns such as "hallucinations", which means that neural networks make up facts as they seem the plausible answer.

- A similar situation arises when the data paradigm is not well suited for the problem at hand. Such is the case of performing **arithmetic calculations**. In this case, networks that can make use of external tools such as calculators have been developed, such as the Toolformer [222].

- Finally, with the increase in LLMs performance experiencing diminishing returns with respect to increases in the model sizes, **expert small models** are re-surging as an efficient alternative [223].

Throughout my PhD only a handful of techniques have been explored but many more have appeared along the way. After all, five years are colloquially said to be a long time in "AI years". Interesting possible future research directions, both related to the ones explored here and novel ones include:

- Training on the edge is an important direction in the context of **federated learning and privacy**, which aims at training decentralized shared models built by combining heterogeneous models trained independently in the wild from multiple physical devices. In particular, training or even simply fine-tuning quantized networks, which is a very plausible scenario, further complicates the task since a full precision model needs to be stored during the weights updates.

- Deployment on the edge similarly continues to make steady steps. Currently, cloud computation through APIs is the dominating paradigm for practical reasons. However, at an overhead caused by communication routines, privacy concerns and network issues. For these reasons, edge inference will always be preferred over remote processing. Edge compute on the other hand requires highly custom code per platform for maximum resources utilization [54]. For example, our FlyNet publication reduces the number of parameters with the intention of speeding up the loading of networks from read-only memory. This would be useful in embedded devices with limited RAM. Additionally, serverless compute is starting to appear as an intermediate solution, providing decentralized cloud compute close to the end-user.

- Generative models are currently dominated by diffusion models [49] which require multiple passes to generate a single sample during inference. Adversarial networks on the other hand require a single pass, and thus continue to be investigated. Adversarial frameworks are unstable and have issues such as mode collapse. Maximum Likelihood training on the other hand yields high-quality samples, are stable and easy to train. Nevertheless, our L2-normalization in spherical networks helps stabilize the training of GANs making them more competitive against diffusion-based models. Consistency models [224] recently proposed aim to learn single-step prediction and are backed by a similar theory to diffusion networks.

- With the rise of LLMs, **training budgets**, as well as efficient inference costs, play a key role in the profitability of these commercial foundation models. **Sparse Mixture of Experts** (SMoEs), has surged as an alternative to dense layers for faster training convergence without sacrificing accuracy. 8 and 4-bit integer quantized versions are commonly released for these models. Additionally, derived from this is the field of **personalizing** and fine-tuning pre-trained large models for specific tasks such as Low-Rank adaptation (LoRA) [225].

- **Multimodal** learning offers a fresh field for efficiency, combining data from multiple domains into learned joint embeddings. Examples are CLIP [226], being a landmark work, followed by SLIP [227] and BLIP [228]. Among the proposed alternatives for efficient multimodal networks are those with shared cross-modality components [229].

- Another path consists in leveraging low-level parallel libraries such as CUDA or the more recently released Triton in order to develop **hardware aware layers**. Initiated by FlashAttention [230] which performs multihead attention with awareness of RAM hierarchies, bandwidths and sizes. Follow-up works include FlashConv [231] and Long Convolutions [232].

- Similarly, making use of newly developed hardware specific for deep learning such as photonics chips for fast memory transfer, thermodynamical [233] components for learning distributions and tensor processing units is an area of opportunity.

- **Collaborative** distributed training and deployments across multiple servers such as in those proposed by Petals and Hivemind [234].

- Finally, recently **State Space Models** [235] demonstrated competitive performance in modeling long sequences more efficiently than transformers, which also represents an interesting avenue for research.

Bibliography

[1] Richard Sutton. The bitter lesson, 2019. URL http://incompleteideas.net/IncIdeas/BitterLesson.html.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. Adv. Neural Inf. Process. Syst.*, pages 1097–1105, 2012.

[3] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *Int. J. Comp. Vis.*, 115(3):211–252, 2015.

[4] Sara Hooker. The hardware lottery. *arXiv preprint arXiv:2009.06489*, 2020.

[5] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 770–778, 2016.

[7] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[8] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[9] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270, 2022.

[10] Paras Jain, Ajay Jain, Aniruddha Nrusimha, Amir Gholami, Pieter Abbeel, Joseph Gonzalez, Kurt Keutzer, and Ion Stoica. Checkmate: Breaking the memory wall with optimal tensor rematerialization. *Proceedings of Machine Learning and Systems*, 2:497–511, 2020.

[11] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[12] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

[13] Payal Dhar. The carbon impact of artificial intelligence. *Nature Machine Intelligence*, 2(8):423–425, 2020.

[14] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Proc. Adv. Neural Inf. Process. Syst.*, pages 1269–1277, 2014.

[15] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[16] Yang He, Ping Liu, Ziwei Wang, and Yi Yang. Pruning filter via geometric median for deep convolutional neural networks acceleration. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.

[17] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *Proc. Int. Conf. Learn. Repren.*, 2017.

[18] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. In *Proc. Adv. Neural Inf. Process. Syst.*, 2018.

[19] Babak Hassibi and David G Stork. *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann, 1993.

[20] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[21] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. In *Proc. Int. Conf. Learn. Repren.*, 2018.

[22] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Proc. Adv. Neural Inf. Process. Syst.*, pages 3123–3131, 2015.

[23] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Proc. Adv. Neural Inf. Process. Syst.*, pages 4107–4115, 2016.

[24] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Proc. Eur. Conf. Comp. Vis.*, pages 525–542, 2016.

[25] Luis Guerra, Bohan Zhuang, Ian Reid, and Tom Drummond. Switchable precision neural networks. *arXiv preprint arXiv:2002.02815*, 2020.

[26] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *Proc. Int. Conf. Learn. Repren.*, 2017.

[27] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.

[28] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018.

[29] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *Proc. Int. Conf. Learn. Repren.*, 2019.

[30] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[31] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[32] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 4510–4520, 2018.

[33] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.

[34] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *Proc. Int. Conf. Learn. Repren.*, 2016.

[35] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[36] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[37] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. Int. Conf. Learn. Repren.*, 2015.

[38] Nick Kanopoulos, Nagesh Vasanthavada, and Robert L Baker. Design of an image edge detection filter using the sobel operator. *IEEE Journal of solid-state circuits*, 23(2):358–367, 1988.

[39] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[40] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[42] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pages 213–229. Springer, 2020.

[43] Irwan Bello, Barret Zoph, Ashish Vaswani, Jonathon Shlens, and Quoc V Le. Attention augmented convolutional networks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3286–3295, 2019.

[44] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jonathon Shlens. Stand-alone self-attention in vision models. *arXiv preprint arXiv:1906.05909*, 2019.

[45] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[46] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021.

[47] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986, 2022.

[48] Paul Smolensky et al. Information processing in dynamical systems: Foundations of harmony theory. 1986.

[49] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[50] Kyung Hyun Cho, Tapani Raiko, and Alexander Ilin. Gaussian-bernoulli deep boltzmann machine. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2013.

[51] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[52] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.

[53] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices. *arXiv preprint arXiv:2007.10319*, 2020.

[54] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul Whatmough. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. *Proceedings of Machine Learning and Systems*, 3, 2021.

[55] Colby R Banbury, Vijay Janapa Reddi, Max Lam, William Fu, Amin Fazel, Jeremy Holleman, Xinyuan Huang, Robert Hurtado, David Kanter, Anton Lokhmotov, et al. Benchmarking tinyml systems: Challenges and direction. *arXiv preprint arXiv:2003.04821*, 2020.

[56] Alexander Ratner, Dan Alistarh, Gustavo Alonso, David G Andersen, Peter Bailis, Sarah Bird, Nicholas Carlini, Bryan Catanzaro, Jennifer Chayes, Eric Chung, et al. Mlsys: The new frontier of machine learning systems. *arXiv preprint arXiv:1904.03257*, 2019.

[57] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.

[58] Dario Amodei and Daniel Hernandez. Ai and compute, 2018. URL https://openai.com/blog/ai-and-compute/.

[59] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

[60] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

[61] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[62] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[63] Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pages 5547–5569. PMLR, 2022.

[64] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[65] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. Int. Conf. Learn. Repren.*, 2015.

[66] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 293–302, 2019.

[67] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[68] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

[69] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[70] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014.

[71] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28:2575–2583, 2015.

[72] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066. PMLR, 2013.

[73] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *Proc. Int. Conf. Learn. Repren.*, 2016.

[74] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

[75] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 8817–8826, 2018.

[76] Sida Wang and Christopher Manning. Fast dropout training. In *international conference on machine learning*, pages 118–126. PMLR, 2013.

[77] Xavier Gastaldi. Shake-shake regularization. *arXiv preprint arXiv:1705.07485*, 2017.

[78] Hongwei Yong, Jianqiang Huang, Deyu Meng, Xiansheng Hua, and Lei Zhang. Momentum batch normalization for deep learning with small batch size. In *European Conference on Computer Vision*, pages 224–240. Springer, 2020.

[79] Chris M Bishop. Training with noise is equivalent to tikhonov regularization. *Neural computation*, 7(1):108–116, 1995.

[80] Caglar Gulcehre, Marcin Moczulski, Misha Denil, and Yoshua Bengio. Noisy activation functions. In *International conference on machine learning*, pages 3059–3068. PMLR, 2016.

[81] Itay Hubara, Yury Nahshan, Yair Hanani, Ron Banner, and Daniel Soudry. Improving post training neural quantization: Layer-wise calibration and integer programming. *arXiv preprint arXiv:2006.10518*, 2020.

[82] N. Boumal, B. Mishra, P.-A. Absil, and R. Sepulchre. Manopt, a Matlab toolbox for optimization on manifolds. *Journal of Machine Learning Research*, 15(42): 1455–1459, 2014. URL https://www.manopt.org.

[83] Yu Bai, Yu-Xiang Wang, and Edo Liberty. Proxquant: Quantized neural networks via proximal operators. In *Proc. Int. Conf. Learn. Repren.*, 2019.

[84] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in optimization*, 1(3):127–239, 2014.

[85] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 3431–3440, 2015.

[86] Guosheng Lin, Anton Milan, Chunhua Shen, and Ian Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1925–1934, 2017.

[87] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[88] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 38(10):1943–1955, 2016.

[89] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric Xing. Neural architecture search with bayesian optimisation and optimal transport. *arXiv preprint arXiv:1802.07191*, 2018.

[90] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[91] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[92] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

[93] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proc. IEEE Int. Conf. Comp. Vis.*, volume 2, page 6, 2017.

[94] Subutai Ahmad and Luiz Scheinkman. How can we be so dense? the benefits of using highly sparse representations. *arXiv preprint arXiv:1903.11257*, 2019.

[95] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Proc. Adv. Neural Inf. Process. Syst.*, pages 1135–1143, 2015.

[96] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing fine-tuning and rewinding in neural network pruning. In *International Conference on Learning Representations*, 2020.

[97] Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. *arXiv preprint arXiv:1911.07412*, 2019.

[98] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. In *Proc. Int. Conf. Learn. Repren.*, 2019.

[99] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*, 2020.

[100] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 4820–4828, 2016.

[101] Diwen Wan, Fumin Shen, Li Liu, Fan Zhu, Jie Qin, Ling Shao, and Heng Tao Shen. Tbn: Convolutional neural network with ternary inputs and binary weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 315–332, 2018.

[102] Peng Chen, Bohan Zhuang, and Chunhua Shen. Fatnn: Fast and accurate ternary neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5219–5228, 2021.

[103] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[104] Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. Reactnet: Towards precise binary neural network with generalized activation functions. In *European Conference on Computer Vision*, pages 143–159. Springer, 2020.

[105] Lin Xiao. Dual averaging method for regularized stochastic learning and online optimization. *Advances in Neural Information Processing Systems*, 22, 2009.

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

[107] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? *Advances in neural information processing systems*, 27, 2014.

[108] Linfeng Zhang, Jiebo Song, Anni Gao, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3713–3722, 2019.

[109] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 1–9, 2015.

[110] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[111] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proc. AAAI Conf. on Arti. Intel.*, volume 4, page 12, 2017.

[112] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 1251–1258, 2017.

[113] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019.

[114] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 5987–5995, 2017.

[115] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[116] Daquan Zhou, Qibin Hou, Yunpeng Chen, Jiashi Feng, and Shuicheng Yan. Rethinking bottleneck structure for efficient mobile network design. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 680–697. Springer, 2020.

[117] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[118] Gilad Yehudai and Shamir Ohad. Learning a single neuron with gradient methods. In *Conference on Learning Theory*, pages 3756–3786. PMLR, 2020.

[119] Weihang Xu and Simon Du. Over-parameterization exponentially slows down gradient descent for learning a single neuron. In *The Thirty Sixth Annual Conference on Learning Theory*, pages 1155–1198. PMLR, 2023.

[120] Yossi Arjevani and Michael Field. Annihilation of spurious minima in two-layer relu networks. *Advances in Neural Information Processing Systems*, 35:37510–37523, 2022.

[121] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[122] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

[123] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.

[124] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[125] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[126] Boan Liu, Liang Ding, Li Shen, Keqin Peng, Yu Cao, Dazhao Cheng, and Dacheng Tao. Diversifying the mixture-of-experts representation for language models with orthogonal optimizer. *arXiv preprint arXiv:2310.09762*, 2023.

[127] Frederick Tung and Greg Mori. Clip-q: Deep network compression learning by in-parallel pruning-quantization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7873–7882, 2018.

[128] Yinghao Xu, Xin Dong, Yudian Li, and Hao Su. A main/subsidiary network framework for simplifying binary neural network. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.

[129] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proc. IEEE Int. Conf. Comp. Vis.*, pages 2755–2763. IEEE, 2017.

[130] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 9194–9203, 2018.

[131] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proc. Eur. Conf. Comp. Vis.*, pages 784–800, 2018.

[132] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Proc. Adv. Neural Inf. Process. Syst.*, pages 2951–2959, 2012.

[133] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X.

[134] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55, 2014.

[135] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[136] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. Int. Conf. Mach. Learn.*, pages 448–456, 2015.

[137] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv preprint arXiv:1609.07061*, 2016.

[138] Sam Leroux, Bert Vankeirsbilck, Tim Verbelen, Pieter Simoens, and Bart Dhoedt. Training binary neural networks with knowledge transfer. *Neurocomputing*, 2019.

[139] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proc. Eur. Conf. Comp. Vis.*, pages 409–424, 2018.

[140] Mason McGill and Pietro Perona. Deciding how to decide: Dynamic routing in artificial neural networks. In *International Conference on Machine Learning*, pages 2363–2372. PMLR, 2017.

[141] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018.

[142] Qing Jin, Linjie Yang, and Zhenyu Liao. Adabits: Neural network quantization with adaptive bit-widths. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2146–2156, 2020.

[143] Jiseok Youn, Jaehun Song, Hyung-Sin Kim, and Saewoong Bahk. Bitwidth-adaptive quantization-aware neural network training: A meta-learning approach. In *European Conference on Computer Vision*, pages 208–224. Springer, 2022.

[144] Kunyuan Du, Ya Zhang, and Haibing Guan. From quantized dnns to quantizable dnns. *arXiv preprint arXiv:2004.05284*, 2020.

[145] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *Proc. Eur. Conf. Comp. Vis.*, 2018.

[146] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5918–5926, 2017.

[147] Daisuke Miyashita, Edward H Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*, 2016.

[148] Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1803–1811, 2019.

[149] Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian Reid. Towards effective low-bitwidth convolutional neural networks. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018.

[150] Tao Sheng, Chen Feng, Shaojie Zhuo, Xiaopeng Zhang, Liang Shen, and Mickey Aleksic. A quantization-friendly separable convolution for mobilenets. In *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, pages 14–18. IEEE, 2018.

[151] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *Int. J. Comp. Vis.*, 88(2):303–338, 2010.

[152] Bharath Hariharan, Pablo Arbeláez, Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. Semantic contours from inverse detectors. In *Proc. Eur. Conf. Comp. Vis.*, 2011.

[153] Shilin Zhu, Xin Dong, and Hao Su. Binary ensemble neural network: More bits per network or more networks per bit? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4923–4932, 2019.

[154] Alexander G Anderson and Cory P Berg. The high-dimensional geometry of binary neural networks. *arXiv preprint arXiv:1705.07199*, 2017.

[155] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. Scalable methods for 8-bit training of neural networks. *arXiv preprint arXiv:1805.11046*, 2018.

[156] Adrian Bulat and Georgios Tzimiropoulos. Xnor-net++: Improved binary neural networks. *arXiv preprint arXiv:1909.13863*, 2019.

[157] Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin. Understanding straight-through estimator in training activation quantized neural nets. *arXiv preprint arXiv:1903.05662*, 2019.

[158] Sajad Darabi, Mouloud Belbahri, Matthieu Courbariaux, and Vahid Partovi Nia. Bnn+: Improved binary network training. 2018.

[159] Thalaiyasingam Ajanthan, Kartik Gupta, Philip Torr, Richad Hartley, and Puneet Dokania. Mirror descent view for neural network quantization. In *International Conference on Artificial Intelligence and Statistics*, pages 2809–2817. PMLR, 2021.

[160] Lu Hou and James T Kwok. Loss-aware weight quantization of deep networks. In *Proc. Int. Conf. Learn. Repren.*, 2018.

[161] Lu Hou, Quanming Yao, and James T Kwok. Loss-aware binarization of deep networks. In *Proc. Int. Conf. Learn. Repren.*, 2017.

[162] Thalaiyasingam Ajanthan, Puneet K Dokania, Richard Hartley, and Philip HS Torr. Proximal mean-field for neural network quantization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4871–4880, 2019.

[163] Charbel Sakr, Jungwook Choi, Zhuo Wang, Kailash Gopalakrishnan, and Naresh Shanbhag. True gradient-based training of deep binary activated neural networks via continuous binarization. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2346–2350. IEEE, 2018.

[164] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4852–4861, 2019.

[165] Christos Louizos, Matthias Reisser, Tijmen Blankevoort, Efstratios Gavves, and Max Welling. Relaxed quantization for discretized neural networks. In *Proc. Int. Conf. Learn. Repren.*, 2019.

[166] Aojun Zhou, Anbang Yao, Kuan Wang, and Yurong Chen. Explicit loss-error-aware quantization for low-bit deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9426–9435, 2018.

[167] Yurii Nesterov. Primal-dual subgradient methods for convex problems. *Mathematical programming*, 120(1):221–259, 2009.

[168] Penghang Yin, Shuai Zhang, Jiancheng Lyu, Stanley Osher, Yingyong Qi, and Jack Xin. Binaryrelax: A relaxation approach for training deep neural networks with quantized weights. *SIAM Journal on Imaging Sciences*, 11(4):2205–2223, 2018.

[169] Wei Tang, Gang Hua, and Liang Wang. How to train a compact binary neural network with high accuracy? In *Proc. AAAI Conf. on Arti. Intel.*, pages 2625–2631, 2017.

[170] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[171] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[172] Jeffrey Pennington and Pratik Worah. The spectrum of the fisher information matrix of a single-hidden-layer neural network. *Advances in neural information processing systems*, 31, 2018.

[173] Yuichi Yoshida and Takeru Miyato. Spectral norm regularization for improving the generalizability of deep learning. *arXiv preprint arXiv:1705.10941*, 2017.

[174] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

[175] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. 2018.

[176] Weiyang Liu, Rongmei Lin, Zhen Liu, Lixin Liu, Zhiding Yu, Bo Dai, and Le Song. Learning towards minimum hyperspherical energy. *Advances in neural information processing systems*, 31, 2018.

[177] Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xian-sheng Hua. Quantization networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7308–7316, 2019.

[178] Qinghao Hu, Peisong Wang, and Jian Cheng. From hashing to cnns: Training binary weight networks via hashing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[179] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[180] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

[181] Mark D McDonnell. Training wide residual networks for deployment using a single bit for each weight. *arXiv preprint arXiv:1802.08530*, 2018.

[182] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. 2016.

[183] Hassan Dbouk and Naresh Shanbhag. Generalized depthwise-separable convolutions for adversarially robust and efficient neural networks. *Advances in Neural Information Processing Systems*, 34:12027–12039, 2021.

[184] Sam Gross, Marc'Aurelio Ranzato, and Arthur Szlam. Hard mixtures of experts for large scale weakly supervised vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6865–6873, 2017.

[185] Ravi Teja Mullapudi, William R Mark, Noam Shazeer, and Kayvon Fatahalian. Hydranets: Specialized dynamic architectures for efficient inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8080–8089, 2018.

[186] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[187] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic convolution: Attention over convolution kernels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11030–11039, 2020.

[188] Brandon Yang, Gabriel Bender, Quoc V Le, and Jiquan Ngiam. Condconv: Conditionally parameterized convolutions for efficient inference. *arXiv preprint arXiv:1904.04971*, 2019.

[189] Mark Horowitz. Computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14. IEEE, 2014.

[190] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[191] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[192] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Andrea Vedaldi. Gather-excite: Exploiting feature context in convolutional neural networks. *arXiv preprint arXiv:1810.12348*, 2018.

[193] Yunpeng Chen, Yannis Kalantidis, Jianshu Li, Shuicheng Yan, and Jiashi Feng. *a*2-nets: Double attention networks. *arXiv preprint arXiv:1810.11579*, 2018.

[194] Jongchan Park, Sanghyun Woo, Joon-Young Lee, and In So Kweon. Bam: Bottleneck attention module. *arXiv preprint arXiv:1807.06514*, 2018.

[195] Bichen Wu, Xuanyu Zhou, Sicheng Zhao, Xiangyu Yue, and Kurt Keutzer. Squeezesegv2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a lidar point cloud. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 4376–4382. IEEE, 2019.

[196] Hang Su, Varun Jampani, Deqing Sun, Orazio Gallo, Erik Learned-Miller, and Jan Kautz. Pixel-adaptive convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11166–11175, 2019.

[197] Qilong Wang, Banggu Wu, Pengfei Zhu, Peihua Li, Wangmeng Zuo, and Qinghua Hu. Eca-net: efficient channel attention for deep convolutional neural networks, 2020 ieee. In *CVF Conference on Computer Vision and Pattern Recognition (CVPR). IEEE*, 2020.

[198] Zilin Gao, Jiangtao Xie, Qilong Wang, and Peihua Li. Global second-order pooling convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3024–3033, 2019.

[199] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

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

[201] Yunsheng Li, Yinpeng Chen, Xiyang Dai, Dongdong Chen, Mengchen Liu, Lu Yuan, Zicheng Liu, Lei Zhang, and Nuno Vasconcelos. Micronet: Improving image recognition with extremely low flops. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 468–477, 2021.

[202] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[203] Kunran Xu, Yishi Li, Huawei Zhang, Rui Lai, and Lin Gu. Etinynet: Extremely tiny network for tinyml. 2022.

[204] Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning*, pages 4651–4664. PMLR, 2021.

[205] Jianyuan Guo, Kai Han, Han Wu, Chang Xu, Yehui Tang, Chunjing Xu, and Yunhe Wang. Cmt: Convolutional neural networks meet vision transformers. *arXiv preprint arXiv:2107.06263*, 2021.

[206] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in Neural Information Processing Systems*, 34, 2021.

[207] Asher Trockman and J Zico Kolter. Patches are all you need? *arXiv preprint arXiv:2201.09792*, 2022.

[208] Ji Lin, Wei-Ming Chen, Han Cai, Chuang Gan, and Song Han. Mcunetv2: Memory-efficient patch-based inference for tiny deep learning. *arXiv preprint arXiv:2110.15352*, 2021.

[209] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *International conference on machine learning*, pages 1319–1327. PMLR, 2013.

[210] Ningning Ma, Xiangyu Zhang, and Jian Sun. Funnel activation for visual recognition. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI 16*, pages 351–368. Springer, 2020.

[211] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet: More features from cheap operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1580–1589, 2020.

[212] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

[213] Keivan Alizadeh Vahid, Anish Prabhu, Ali Farhadi, and Mohammad Rastegari. Butterfly transform: An efficient fft based neural architecture design. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12021–12030. IEEE, 2020.

[214] Kai Han, Yunhe Wang, Qiulin Zhang, Wei Zhang, Chunjing Xu, and Tong Zhang. Model rubik's cube: Twisting resolution, depth and width for tinynets. *Advances in Neural Information Processing Systems*, 33:19353–19364, 2020.

[215] Yinpeng Chen, Xiyang Dai, Dongdong Chen, Mengchen Liu, Xiaoyi Dong, Lu Yuan, and Zicheng Liu. Mobile-former: Bridging mobilenet and transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5270–5279, 2022.

[216] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[217] Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

[218] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[219] Jianhao Zhang, Yingwei Pan, Ting Yao, He Zhao, and Tao Mei. dabnn: A super fast inference framework for binary neural networks on arm devices. In *Proceedings of the 27th ACM international conference on multimedia*, pages 2272–2275, 2019.

[220] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[221] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[222] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

[223] Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. Textbooks are all you need ii: phi-1.5 technical report. *arXiv preprint arXiv:2309.05463*, 2023.

[224] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. *arXiv preprint arXiv:2303.01469*, 2023.

[225] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[226] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[227] Norman Mu, Alexander Kirillov, David Wagner, and Saining Xie. Slip: Self-supervision meets language-image pre-training. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXVI*, pages 529–544. Springer, 2022.

[228] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. *arXiv preprint arXiv:2201.12086*, 2022.

[229] Haoxuan You, Luowei Zhou, Bin Xiao, Noel C Codella, Yu Cheng, Ruochen Xu, Shih-Fu Chang, and Lu Yuan. Ma-clip: towards modality-agnostic contrastive language-image pre-training. 2021.

[230] Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *arXiv preprint arXiv:2205.14135*, 2022.

[231] Tri Dao, Daniel Y Fu, Khaled K Saab, Armin W Thomas, Atri Rudra, and Christopher Ré. Hungry hungry hippos: Towards language modeling with state space models. *arXiv preprint arXiv:2212.14052*, 2022.

[232] Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models. *arXiv preprint arXiv:2302.10866*, 2023.

[233] Patrick J Coles. Thermodynamic ai and the fluctuation frontier. *arXiv preprint arXiv:2302.06584*, 2023.

[234] Alexander Borzunov, Dmitry Baranchuk, Tim Dettmers, Max Ryabinin, Younes Belkada, Artem Chumachenko, Pavel Samygin, and Colin Raffel. Petals: Collaborative inference and fine-tuning of large models. *arXiv preprint arXiv:2209.01188*, 2022.

[235] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.