An Analysis Framework for the Quantization-Aware Design of Efficient, Low-Power Convolutional Neural Networks

by

Stone Yun

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2022

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

The following two papers are incorporated into this thesis. I was co-author with major contributions to the design, analysis, writing and editing.

S. Yun and A. Wong, "Where Should We Begin? A Low-Level Exploration of Weight Initialization Impact on Quantized Behaviour of Deep Neural Networks", J. Comp. Vis. Imag. Sys., vol. 6, no. 1, pp. 1–5, Jan. 2021.

This paper is incorporated in Chapters 3 and 4.

S. Yun and A. Wong. "FactorizeNet: Progressive Depth Factorization for Efficient Network Architecture Exploration Under Quantization Constraints", 6th Workshop on Energy Efficient Machine Learning and Cognitive Computing (EMC2 2020), Online (San Jose, California), 2020.

This paper is incorporated in Chapters 3 and 5.

Abstract

Deep convolutional neural network (CNN) algorithms have emerged as a powerful tool for many computer vision tasks such as image classification, object detection, and semantic segmentation. However, these algorithms are computationally expensive and difficult to adapt for resource constrained environments. With the proliferation of CNNs for mobile, there is a growing need for methods to reduce their latency and power consumption. Furthermore, we would like a principled approach to the design and understanding of CNN model behaviour. Computationally efficient CNN architecture design and running inference with limited precision arithmetic (commonly referred to as neural network quantization) have become ubiquitous techniques for speeding up CNN inference speed and reducing their power consumption. This work describes a method for analyzing the quantized behaviour of efficient CNN architectures and subsequently leveraging those insights for quantization-aware design of CNN models.

We introduce a framework for fine-grained, layerwise analysis of CNN models during and after training. We present an in-depth, fine-grained ablation approach to understanding the effect of different design choices on the layerwise distributions of weights and activations of CNNs. This layerwise analysis enables us to gain deep insights on how the interaction of training data, hyperparameters, and CNN architecture can ultimately affect quantized behaviour. Additionally, analysis of these distributions can yield additional insights on how information is propagating through the system. Various works have sought to design fixed precision quantization algorithms and optimization techniques that minimize quantization-induced performance degradation. However, to the best of our knowledge, there has not been any prior works focusing on a fine-grained analysis of *why* a given CNN's quantization behaviour is observed.

We demonstrate the use of this framework in two contexts of quantization-aware model design. The first is a novel ablation study investigating the impact of random weight initialization on final trained distributions of different CNN architectures and resulting quantized accuracy. Next, we combine our analysis framework with a novel "progressive depth factorization" strategy for an iterative, systematic exploration of efficient CNN architectures under quantization constraints. We algorithmically increase the granularity of depth factorization in a progressive manner while observing the resulting change in layerwise distributions. Thus, progressive depth factorization enables the gain of in-depth, layer-level insights on efficiency-accuracy tradeoffs. Coupling fine-grained analysis with progressive depth factorization frames our design in the context of quantized behaviour. Thus, it enables efficient identification of the optimal depth-factorized macroarchitecture design based on the desired efficiency-accuracy requirements *under quantization*.

Acknowledgements

I would like to thank all the people who made this thesis possible. Whether it be conversations with my colleagues and friends, chats with various authors and presenters I have met during my studies or valuable feedback from peer reviewers, our conversations inspire me to continue striving to learn beyond what I know.

More specifically, I would like to thank my supervisor, Dr. Alexander Wong for his invaluable guidance, friendship and support. You inspire me every day to be a diligent, insightful researcher who never stops asking the deeper questions. In working with you, I've been exposed to an entirely new world of research and learning.

Next, I would like to thank Dr. Clausi and Dr. Wang for reviewing my thesis. I greatly appreciate you taking the time to review and comment on my work. Your feedback has been invaluable and I really enjoy learning from you.

I would also like to thank my manager Alireza for his continued support as I work full-time and study part-time. Besides our many insightful chats about research and engineering being incredibly helpful, your faith and understanding in my ability to work as a full-time engineer while conducting graduate studies were crucial in allowing me this opportunity.

Finally, I would like to thank my amazing, loving fiancée, Sydney for her support throughout the countless late nights of paper writing, paper reading, and experiment debugging. I can't imagine completing this thesis without you by my side. Also, thank you for sitting through my presentation rehearsals. I know it's not the most interesting content, but it really helps keep me calm and ready.

Dedication

This is dedicated to the one I love, my beautiful fiancée Sydney.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep convolutional neural networks (CNN) are a class of neural network (NN) architectures that have become the dominant method for solving various computer vision and image processing tasks. Their tremendous accuracy (given adequate training data) is due to the fact that they leverage two fundamental assumptions for vision processing: 1) sparse, local connections/receptive field and 2) weight sharing between spatial locations (leading to translational equivariance). Despite first being introduced in 1989 [27], deep convolutional neural networks did not gain widespread academic attention until after AlexNet [26] won the ImageNet competition [40] in 2012. This was in large part due to a lack of adequate computing power at the time of their inception. Since then, there has been a rapid proliferation of research related to advancing the application and understanding of deep CNNs. Thus, enabling dramatic advances in computer vision. Recent research has demonstrated incredible performance on vision tasks such as image classification [16, 26, 47], object detection [10, 31, 37, 53], image segmentation [14, 38] and many more [8, 28, 42, 46, 51, 56].

Evidently, CNNs are a powerful tool for creating accurate, data-driven algorithms to solve various tasks. However, the computation required to train and test these algorithms leads to heavy power consumption and the need for expensive hardware. Current state-of-the-art vision models have over 60 million parameters and require tens or, depending on the use-case, even hundreds of gigabytes of memory to train. While computing power has continued to grow at an astonishing rate, considering methods to improve the efficiency of CNNs would be germane to the mainstream adoption and accessibility of deep learning technologies. Increased wireless connectivity has made cloud-AI computing for mobile devices a reality but there are still plenty of applications that would be better served with on-device processing such as camera image signal processing (ISP), photo/video gallery search, photo-editing, and many more. Efficient CNN architecture design and limited

precision, quantized inference have emerged as two primary techniques for enabling fast, low-power processing of CNNs on edge-devices.

1.1 Convolutional Neural Networks

The first feed-forward, multi-layer neural networks - also called multilayer perceptrons (MLP) - consisted of densely connected neurons where every value in an input feature vector was connected to every neuron in that given layer (see Figure 1.1). This meant that they could only handle a fix-sized image input (i.e., the number of input pixels could not change) and that the number of parameters in the network dramatically increased with image resolution. For example in Figure 1.1, we can see that for a given dense layer with inputs of length N (orange for input layer, blue for output layer) and M number of neurons (blue for input layer, green for output layer), the number of parameters required is $N \times M$. For a standard HD image, the input vector alone is over 6 *million* values. Most image classification tasks resize images to 224×224 which still results in over 150,000 values. We can see that for most real-world applications, the dimensionality of a naive dense layer rapidly explodes.

Furthermore, dense connectivity also meant that any small local translation could significantly change the output of the network. Consequently, training such networks to generalize to vision tasks, where object and patterns can appear anywhere in the image, requires unreasonably high amounts of data. To illustrate, for an MLP to learn to recognize a simple, handwritten uppercase-L, we would need to have samples for every possible location that the L could appear in (e.g., upper-right, center, lower-left), samples for different styles of L (e.g., more angular L's, more perpendicular L's), and many more. Evidently, the number of data samples required to generalize would exponentially increase with the dimensionality and complexity of vision tasks.

Inspired by the Neocognitron [9] and findings at the time on neuronal connection patterns of the V1 visual cortex, LeNet [27] proposed a novel neural network architecture to greatly simplify computation and data requirements based on two fundamental assumptions/inductive biases of vision:

- Sparse, local neuron connectivity — That is, the relevant features that any given neuron needs to learn/detect is constrained to a local neighbourhood. Consequently, neurons no longer needed to be connected to every single value of the input.
- Weight sharing across spatial locations — That is, the actual relevant features in an image/video are sparsely, redundantly encoded in the pixels and can show up at any

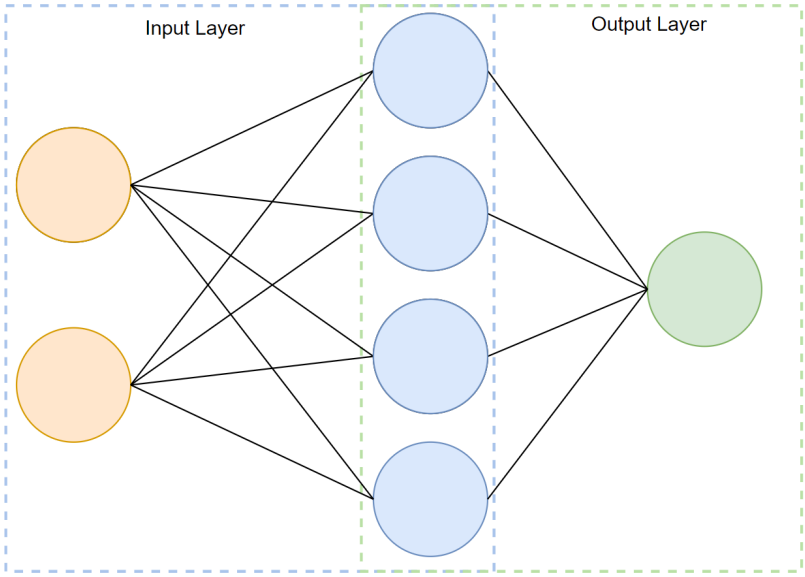

Figure 1.1: **Basic MLP**. An illustrative example of a densely connected neural network. Each edge/connection is a single layer-weight.

given spatial location. Thus, weight sharing could also be thought of as weight reuse where we reuse the same set of weights to process different locations of the input. Consequently, weight sharing also leads to translational equivariance.

In the context of 2D-image/video processing, convolutional kernels naturally emerge from these two inductive biases of spatial/visual pattern recognition. Thus, the convolutional neural network was invented. This re-parametrization of neural networks for vision would prove to be an incredibly powerful technique for significantly improving the generalization of NNs for vision as well as enabling much more computationally efficient processing as CNNs could make use of highly efficient convolutional kernel optimizations. The combination of sparse, local connectivity and weight sharing (see Figure 1.2 for illustration) led to a dramatic reduction in the number of unique parameters in a neural network. Thus, improving computational efficiency and sample efficiency (i.e., fewer samples are required in the training set in order to generalize). Coincidentally, convolutional filters have long been used in digital signal processing and image processing applications for feature detection and image filtering. Thus, an alternative, albeit misleading interpretation of CNNs is that they consist of a stack of learnable image filters that perform a series of hierarchical, differentially optimized image processing operations on the input.While helpful for initial

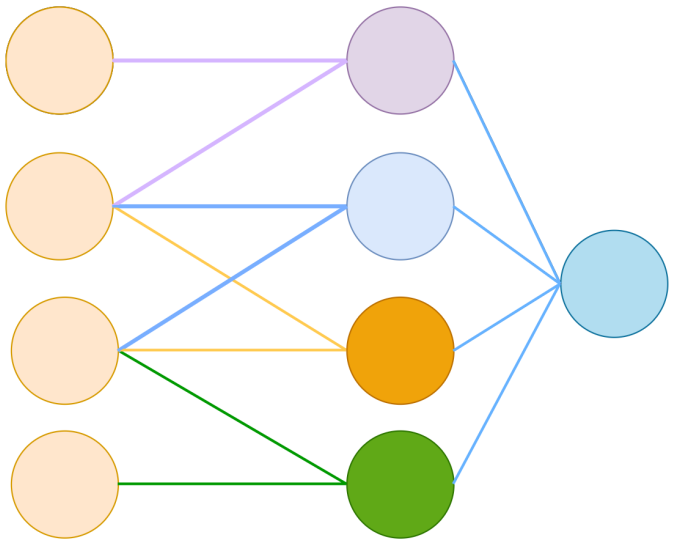

Figure 1.2: **Local connectivity**. An illustrative example of a locally connected neural network. In this architecture, each neuron is only connected to a local neighbourhood of inputs.

intuition, this viewpoint obfuscates the underlying principles of local neuron connectivity upon which CNNs were formulated. In principle, CNNs are learning the local, spatial relations between pixels/image features rather than learning the optimal filtering operations to apply to an image.

Many early CNN architectures consisted of a simple sequential stack of convolutional layers such as the original LeNet [27] and VGG-Net [45]. However since then, there has been an abundance of research in increasingly complex architectures such as multiple parallel branches of convolutions, skip/forwarding connections, and multi-scale architectures as examples. Furthermore, architecture complexities have grown with the complexity of various vision tasks which CNNs are now being applied to such as object detection, pose estimation, crowd counting, and segmentation. Along with these advances, the computational demands of state-of-the-art CNN architectures has continued to grow. Thus, there has also been increasing interest in designing more computationally efficient CNN architectures as well as other methods for accelerating CNN computation to reduce latency and power consumption such as fixed point, limited precision computation.

1.2 Motivation

Previously, CNN algorithms were bound to the domain of high performance computing. Only the most powerful gaming GPUs and cloud computing servers could handle the tremendous computational load required for both training and inference. However, as mobile computing power has increased, so has the interest in using CNNs in everyday, resource constrained settings. In mobile computing platforms, such as cellphones, smart-watches, drones, and Internet-of-Things (IoT), latency and power are the primary concern. CNN designers must find ways to balance accuracy with hardware performance. To illustrate, many desktop gaming CPUs have a power budget on the order of 50-100 watts. In stark contrast, mobile computing platforms typically have a power budget of 1-2 watts or less *for the entire system*. A couple percentage points of increased accuracy often cannot justify reduced battery life and slower response times. Furthermore, tight limits on memory, storage, area, and computational density further constrain the resources available for CNN inference and processing. Thus, mobile CNN algorithms must consider a number of computational cost factors such as the memory required for the neural network parameters, memory required for intermediate layer outputs, number of operations in a single inference (measured in this work as the number of multiply-accumulate operations aka MACs. Alternatively, some works count the number of floating point operations or FLOPs which translates to 2× the number of MACs), and power/energy consumed for a single inference.

To illustrate, most mobile devices only have a few megabytes allocated for the L2-cache (also referred to as on-chip memory). However, minimizing latency and power consumption typically requires keeping as much of the data in L2-cache as possible since accessing external memory incurs energy costs that can be as much as 100× greater than on-chip memory access [3]. A state-of-the-art CNN architecture such as InceptionResNetV2 [47] can require over 50 MB of memory for the parameters alone (assuming parameters are stored as 8-bit integers. Costs significantly increase with 32-bit floating point numbers). "Always-on" algorithms such as wakeword detection (e.g., "OK, Google") and high data-bandwidth algorithms such as video colour enhancement will incur even greater cost if not designed efficiently as they must either be running for extended periods of time or at a high frequency on large amounts of data (e.g., a standard HD video requires processing over six million pixels at a rate of at least 30 frames/second or higher or alternatively, less than 33 ms/inference).

Evidently, there is a growing need for methods of designing efficient, low-power convolutional neural networks that can operate within these constraints. A mobile CNN should deliver high accuracy without occupying the majority of the computational/power resources. With adaptation of co-processors such as digital signal processors (DSP) for

CNN processing, deploying CNN models with fixed point integer weights and activations for quantized inference has become a popular technique for minimizing storage, latency, and power. This is because processors such as the DSP have been well optimized for highly parallel image processing using fixed point arithmetic. However, most CNN design-flows perform design, experimentation, and analysis on the floating point (fp32) model behaviour. Thus, it is often unclear what effect different design choices may have on the actual deployed model since the on-device CNN will often be performing inference with very different precisions. This work explores the effects of efficient CNN design and quantized inference in tandem to better understand the behaviour of deployed, on-device CNN algorithms.

1.3 Contributions

We introduce a novel framework for fine-grained, layerwise analysis of CNN models during and after training. We present an in-depth, fine-grained ablation approach to understanding the effect of different design choices on the distributions of weights and activations of different CNN architectures. This layerwise analysis enables us to gain deep insights on how the interaction of training data, hyperparameters, and CNN architecture can ultimately affect quantized behaviour. Additionally, analysis of these distributions can yield additional insights on how information is propagating through the system.

We demonstrate the use of this analysis framework in two different applications. The first is a novel, systematic ablation study investigating the impact of random weight initialization on final trained distributions of different CNN architectures and in turn, the 8-bit quantized (quint8) inference behaviour. The second is a novel "progressive depth factorization" strategy for efficient CNN architecture exploration under quantization constraints. Coupling the proposed strategy with fine-grained analysis of layer-wise distributions enables the gain of in-depth, layer-level insights on efficiency-accuracy tradeoffs under fixed-precision quantization and increasing depth factorization of convolution. Our overall contribution is framing the design and analysis of CNNs in the context of fixed-point integer computation as it has become a common method for low-power inference. Analyzing CNN behaviour through the lens of quantization drives development of CNN models in a more efficient pipeline that always keeps the deployment environment in mind.

1.4 Outline

The outline of this thesis is as follows:

In Chapter 2, we provide the relevant background for framing the context of this work including neural network quantization and the computational costs of CNN training and inference.

In Chapter 3, we introduce and describe the details of our framework for fine-grained, layerwise analysis of CNN models.

In Chapters 4 and 5 we demonstrate two applications of this analysis framework for quantization-aware design of CNN models.

Finally, Chapter 6 discusses the overall implications and insights of our results as well as future directions for this research.

Chapter 2

Background

2.1 Neural Network Training and Inference

CNN training is comprised of two main stages: forward propagation (also referred to as inference) of the training samples, and backward propagation of the gradient with respect to the loss function (after backward propagation there is also the parameter update step, but for simplicity we will include it with backpropagation). However, after training, a deployed model only needs to compute the forward pass to make predictions on the new samples (sometimes referred to as samples in-the-wild or test samples). The following two subsections will briefly detail the main computation of forwards and backwards propagation as well as the background relevant to why we might be concerned with devising more efficient CNN algorithms.

2.1.1 Forward Propagation

Forward propagation refers to the process of providing an input x to a neural network and applying a sequence of layer operations under composition to produce an output y. This can be more precisely described in Eq. 2.1 where $F(x)$ represents the N-layer neural network function and f_i is the i-th layer of the network.

$$\begin{gathered} y = F(x) \text{ where} \\ F(x) = f_i \circ f_{i-1} \circ \ldots \circ f_1(x) \text{ for } i \in 1, 2, \ldots, N \end{gathered} \tag{2.1}$$

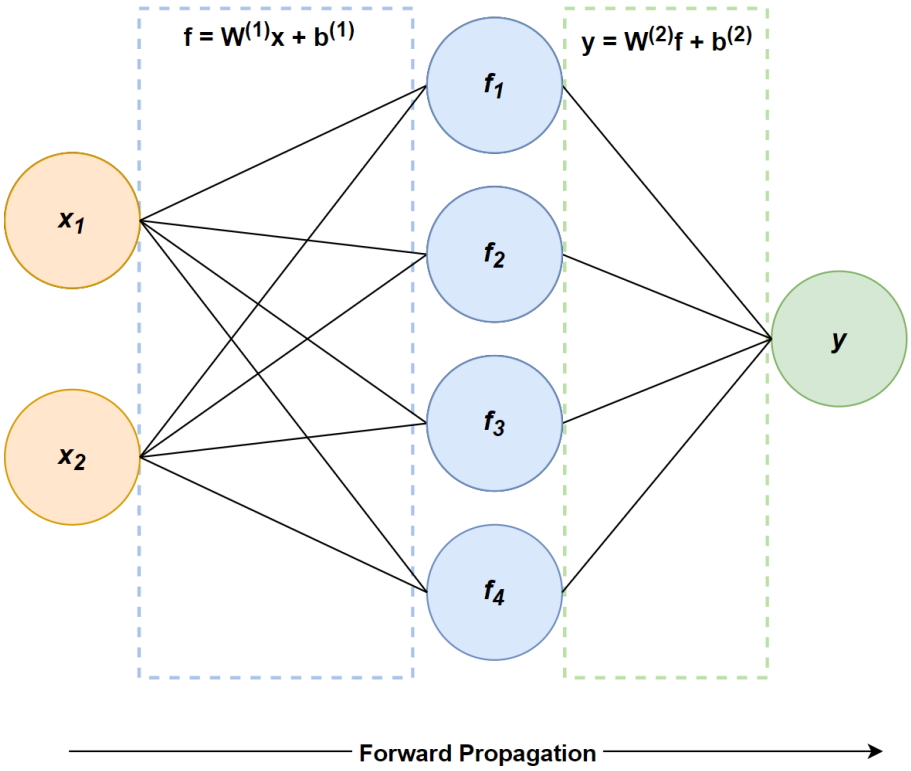

Figure 2.1: **Forward Propagation**. An illustrative example of forward propagation from input vector x to produce output y. The circles represent the values of the feature vector at each stage. Subscripts simply refer to the i-th element of the feature vector. In many cases, y is a multi-dimensional vector or tensor. Each layer has weight matrix $W^{(i)}$.

We can conceptualize computing $F(x)$ as propagating our input x from the first/bottom layer of the network, through successive layers all the way to the output/top layer of the neural network, applying a non-linear transform to the input data/tensor at each layer. The intermediate layers, f_i are referred to as hidden layers and their outputs are often called activations, hidden activations, or hidden feature maps. During training, we often compute forward propagation on a batch of B samples (referred to as a batch-size of B) to utilize a GPU's parallel processing power and reduce the amount of time required to train a model. At inference-time, it is typically assumed that the model will be receiving a single image at a time (i.e., a batch-size of 1).

Figure 2.1 visualizes forward propagation in a simple 2-layer, fully connected neural

network. It should be noted that while for simplicity and illustrative purposes, we describe the neural network as a sequence of single layers $f_i(x)$, $f_i(x)$ is often a more complicated function implemented as multiple layers and operations applied in sequence and/or in parallel. Thus, $f_i(x)$ can also be thought of as the i-th stage or module of the neural network $F(x)$. One can easily see how the amount of computation quickly explodes with the increase in dimensionality of our input data, hidden activations and model parameters.

2.1.2 Backward Propagation and Training

Backward propagation, widely popularized in [39], is the actual mechanism by which a neural network's parameters can be updated/trained to improve its predictions and minimize the target loss function. Leveraging the chain rule, it is the method that has allowed us to efficiently compute the gradient of a cost function with respect to each of the parameters in a multi-layer network using straightforward matrix multiplications for the majority of computation. Thus, backward propagation enables the ability to train powerful, deep networks in a reasonable amount of time. This is illustrated in Algorithm 1.

Algorithm 1: Backpropagation of Error

Result: Compute gradient of loss with respect to each layer's parameters

$L(y) = L \circ F(x)$

// $F(x)$ is N-layer neural network, $L(y)$ is the loss

$\nabla y_N = \frac{\partial L}{\partial y}$ // gradient of the loss w.r.t. output of layer-N

for *i in N, N-1, . . . , 1* **do**

$y_i = f_i(x_{i-1})$; // current layer's output

$\nabla y_i = \frac{\partial L}{\partial f_i}$; // gradient of loss w.r.t. output of layer i

$\frac{\partial L}{\partial W^{(i)}} = \nabla y_i \cdot \frac{\partial f_i}{\partial W^{(i)}}$; // Gradient of loss w.r.t. weights of layer i

$\frac{\partial L}{\partial x_{i-1}} = \nabla y_i \cdot \frac{\partial f}{\partial x_{i-1}}$; // Gradient of loss w.r.t. input to layer i

// $\frac{\partial L}{\partial x_{i-1}}$ is the gradient of loss w.r.t. output of layer i-1 (i.e., ∇y_{i-1}). Thus, we can now backprop to the next layer

end

Similar to forward propagation, computing the gradients in a deep convolutional neural network is a very computationally dense operation. Furthermore, due to the noisy, iterative nature of stochastic gradient descent (SGD), high precision is typically required to accumulate many small gradient steps over training to effectively traverse the loss surface and minimize cost. The traditional weight update computation for stochastic gradient

descent is illustrated in Eq. 2.2 where L is the cost function/loss, η is the learning rate and θ are the model parameters. Note that other variants of SGD exist that seek to speed-up convergence (i.e., reduce the number of steps/iterations required to converge). These typically involve modifying the weight update equation based on some momentum-based term. However, the core ingredient is that they make use of iterative, gradient based updates. The learning rate is a hyperparameter that scales the magnitude of any given update. The intuitive rationale is that when we are optimizing on a non-convex, "poorly behaved" loss surface, we should take smaller incremental steps so as not to overshoot the global minima. Typical values for learning rate can be in the range $\eta \in [10^{-1}, 10^{-4}]$. However, the learning rate can often be even smaller over the course of training due to other hyperparameters such as learning rate schedules and learning rate decay. Learning rate decay describes the process of reducing the learning rate over the course of training so that the optimization process can settle in a global minima. It is analogous to reducing the temperature in simulated annealing. While learning rate schedule and learning rate decay are often used interchangeably, here we make a distinction since there have been recent works proposing learning rate schedules that do not strictly decay η. Thus, one can see how high precision is required for gradient descent to accurately update network parameters during training and converge towards an accurate minimum.

$$\theta_t = \theta_{t-1} - \eta \frac{\partial L}{\partial \theta_{t-1}} \tag{2.2}$$

While there are many promising methods and research in the area of using 16-bit floating point (half-precision), a mix of 16-bit and 32-bit floating point (mixed precision) and even 8-bit floating point numbers for computing gradients and performing backpropagation [2, 6, 33, 50], the standard practices still typically use 32-bit floating point (FP32) since FP32 usually guarantees enough precision for iterative gradient descent methods to converge. The rounding errors introduced by less precision can typically lead to unstable training and a diverging cost function. Thus, when training a neural network, the computation of forwards and backwards propagation both require massive amounts of high-precision computation as well as high memory and bandwidth costs.

2.1.3 Computational Cost of Training and Inference

Neural network training and inference involve high density computation that was previously impossible for most computing hardware to handle in a reasonable amount of time. For example, the state-of-the-art ResNet-152 [16] requires 5.65 billion MACs (or 11.3 billion

FLOPs since a multiply-accumulate operation would contain two floating point operations) for a single inference alone [16]. The less accurate, but more efficient ResNet-34 still requires 1.8 billion MACs for a single inference. These computations are made even more costly when considering the data movement required to load weights and activations into memory, store accumulated partial sums of activations, and also the energy cost of running power-hungry floating point operations. To illustrate, ResNet-152 contains over 60 million parameters. When stored as 32-bit floating point numbers that is more than 228 MB of storage/memory required for just the CNN model alone. Such a large memory footprint guarantees that multiple external memory accesses will be required for the model parameters alone. The memory required for computing activations, gradients, and parameter updates further increases the data movement requirements of a CNN model by several times.

When looking at the hardware cost of a CNN in terms of latency and power consumption, external memory accesses (e.g., DRAM accesses) dominate data movement costs by a couple orders of magnitude [3]. They are much more expensive when compared to accessing a cache or local buffer. Thus, any optimizations that can minimize off-chip data movement (i.e., reading/writing from DRAM) will significantly improve the hardware performance of a CNN algorithm. Besides data movement, reducing the number of MACs in a CNN can also greatly reduce hardware costs as decreasing the number of computations per inference can reduce the time/power consumed from computing all MACs of a forward pass and can also potentially reduce the clock frequency at which the computation needs to be performed. Table 2.1 shows some of the associated memory and computational costs of some state-of-the-art CNNs from the past few years. Eq. 2.3 and Eq. 2.4 describe the number of MACs and number of parameters respectively associated with computing an output activation-map with dimensions $H \times W$ for a single convolutional layer with $K \times K$ filters, C_{in} input channels, and C_{out} output channels. Input channels refers to the number of channels of the incoming image/tensor and output channels refers to the number of channels in the outgoing image/tensor produced by the current layer.

$$MACs = K \times K \times H \times W \times C_{in} \times C_{out} \tag{2.3}$$

$$Params = K \times K \times C_{in} \times C_{out} \tag{2.4}$$

Evidently, computation of a given layer quickly grows with image dimensions, channel depth and kernel size. State-of-the-art models often have several tens of layers, or even over a hundred layers, and the hidden activation maps can have hundreds of channels. Thus, the

computation of accurate CNNs can be difficult to efficiently execute and fit within a tightly constrained computational budget. While it is expected that training is quite expensive considering the massive dataset requirements for training accurate algorithms and the high precision requirements for iterative gradient descent, CNN inference only involves forward propagation. Thus, recent research in enabling "AI-on-the-edge" has focused on designing CNN architectures that are fast and efficient *for inference* such that neural networks can be deployed on mobile devices after they have been intensively trained "offline" (i.e., using a powerful GPU or GPU server). A few prominent areas of efficient CNN research include designing efficient CNN architectures, pruning unnecessary weights/filters, and reducing the precision of the neural network computation (i.e., fixed point quantization or CNN quantization). Oftentimes, deploying a CNN for mobile use-cases will involve some combination of methods from the aforementioned topics, particularly CNN quantization as it can be easily adapted for existing efficient, low-power hardware.

2.2 Fixed Point Quantization of CNN Weights and Activations

In the default setting, CNNs require 32-bit floating point computations for the forward and backward propagation during training. As mentioned in Sec. 2.1.2, this is mainly due to the fact that during iterative gradient descent optimization, gradient updates accumulate in very low magnitudes. Thus, 32-bits of floating point precision are required to adequately represent the data without losing these updates. This high precision data format can have several implications in terms of computational cost. Using high-precision representations usually mean that the weights and activations of the model cannot fit in the on-chip memory (i.e., the on-chip cache). Thus, over the course of a single inference hundreds of megabytes of data may have been moved to/from the external memory. Furthermore, floating point computations often require more complicated, power-hungry arithmetic logic units (ALU) that are less suitable for resource-constrained platforms such as cell-phones, drones, and IoT devices. While modifying the CNN architecture is one way to reduce data movement and computation, it is not always guaranteed to produce an algorithm with comparable accuracy to the more expensive model. Additionally, simply modifying the CNN architecture does not address the issue of performing costly floating point operations.

In light of these challenges, fixed point quantization of convolutional neural networks has quickly emerged as an essential tool for running efficient CNN inference. Models deployed in low-power settings that only require forward propagation/inference computations can often be run with fixed point integer arithmetic. Thus, leading to major savings in

Network Name	Parameter Storage Size (MB)	Num. Parameters	Num. MACs
ResNet-50 [16]	99.2	26M	2.0B
ResNet-152 [16]	288.9	60M	5.65B
ResNeXt-101 [55]	320.4	84M	16B
Inception-v4 [47]	183.1	48M	6.5B
Inception-ResNet-v2 [47]	213.6	56M	6.5B
Xception [5]	87.7	23M	4.2B
SENet [18]	556.9	146M	21B
NASNet-A [58]	339.5	89M	12B
DenseNet-264 [20]	129.7	34M	3.0B
EfficientNet-B0 [49]	20.2	5.3M	0.20B
EfficientNet-B1 [49]	29.8	7.8M	0.35B
EfficientNet-B2 [49]	35.1	9.2M	0.5B

Table 2.1: List of the various state-of-the-art CNNs from the past few years and their associated hardware costs. We limit the quantifiers of cost to those that would be agnostic to underlying hardware/software implementations. For example, bandwidth numbers can be influenced by chip architecture, memory technology and the bus bitwidth. Reported numbers are from [49].

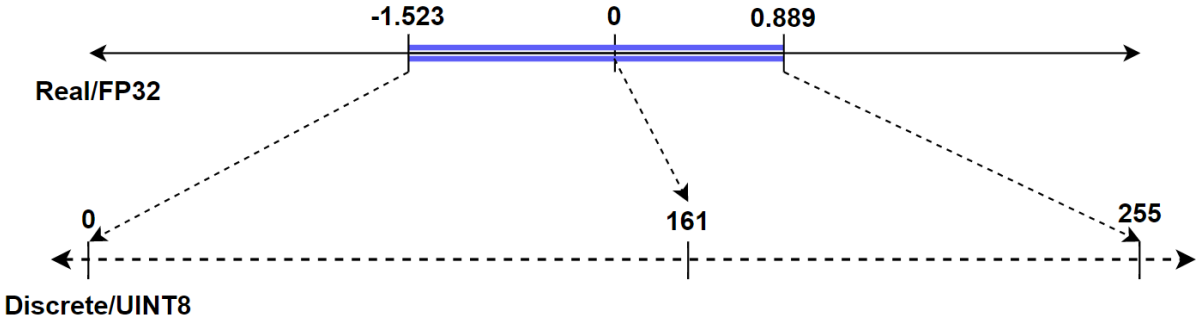

Figure 2.2: **Real/Continuous FP32 mapped to Discrete/Quantized UINT8 representation**. An illustrative example of mapping a set of real/continuous numbers to a quantized/discrete number line. In the simplest case shown above, we perform a linear/affine mapping.

latency, power, and storage. In the commonly used deployment setting, CNN weights and activations are quantized to discrete 8-bit integer representations. Thus, reducing memory/storage requirements of weights and activations by $4\times$ and significantly reducing the computational cost of inference as complex floating point ALUs can be replaced with fast, simple integer ALUs. As seen in Fig. 2.2, this translates to linearly/affinely projecting the real, continuous-valued weights/activations from a known, finite range onto a discrete, linear space. This method of linear quantization is also referred to as uniform quantization due to the fact that the discrete states in the integer space are uniformly distributed along the number line. It has been used in traditional signal processing for decades. Fig. 2.3 illustrates how quantization creates discretized "steps" to approximate a continuous function; each value in the quantized space (i.e., each quantized integer number) represents one of those discrete steps or "levels" in the real number space.

Converting between real and quantized values primarily involves three steps:

1. Determine the range of our data (e.g., range of values in a layer's weights). This will be parameterized by the maximum and minimum values, max and min. For CNN weights, we often take the absolute max/min of the tensor. However, for activations we first need to compute forward propagation of the CNN on a set of "representative" samples (often called the "calibration dataset") to profile the distribution of activations for each layer. A representative calibration dataset should hopefully capture the actual distribution of activation values we expect in the real-world data so that we can select a quantization range that does not lose too much information.

2. Determine the step-size, S as calculated in Eq. 2.5, where max and min are determined from the previous step and N is the number of bits used to represent a quantized number, usually eight. S is the distance/delta when moving between two adjacent points/levels in the discrete integer space. In Fig. 2.3, it is the vertical distance when moving from one consecutive quantization level to another. Step-size can be thought of as inversely proportional to the "resolution" of the quantized mapping as smaller steps mean that we can represent the real numbers with finer precision (i.e., higher resolution).

3. Finally, Eqs. 2.6 and 2.7 show the relations for mapping between real values, r and the quantized values, q. The mapping is parameterized by the step-size from Step 2 and a zero-point, Z. Z is simply the quantized integer value that represents the real-zero. To make the real-zero perfectly map to Z can sometimes require slightly adjusting the values of max and min. This is referred to as making real-zero perfectly quantized. As mentioned in [24] there can be many numerical and, depending on if signed integers are used for the quantized numbers, hardware benefits to making zero perfectly quantized. For quantizing from real to integer, there may also be a clamping operation as seen in Eq. 2.7. This is necessary for cases where max and min are not the absolute maximum and minimum of the tensor such as when quantizing activations.

$$S = \frac{max - min}{2^N - 1} \tag{2.5}$$

$$r = S(q - Z) \tag{2.6}$$

$$q = \frac{clamp(r, min, max)}{S} + Z \tag{2.7}$$

$$clamp(x, a, b) = \begin{cases} x & a \leq x \leq b \\ a & x < a \\ b & x > b \end{cases} \tag{2.8}$$

As can be seen from the formulation of quantization, the main sources of error when mapping from the continuous, real space to the discrete, integer space are rounding error, and clamping errors. Values within the defined max/min range may introduce rounding

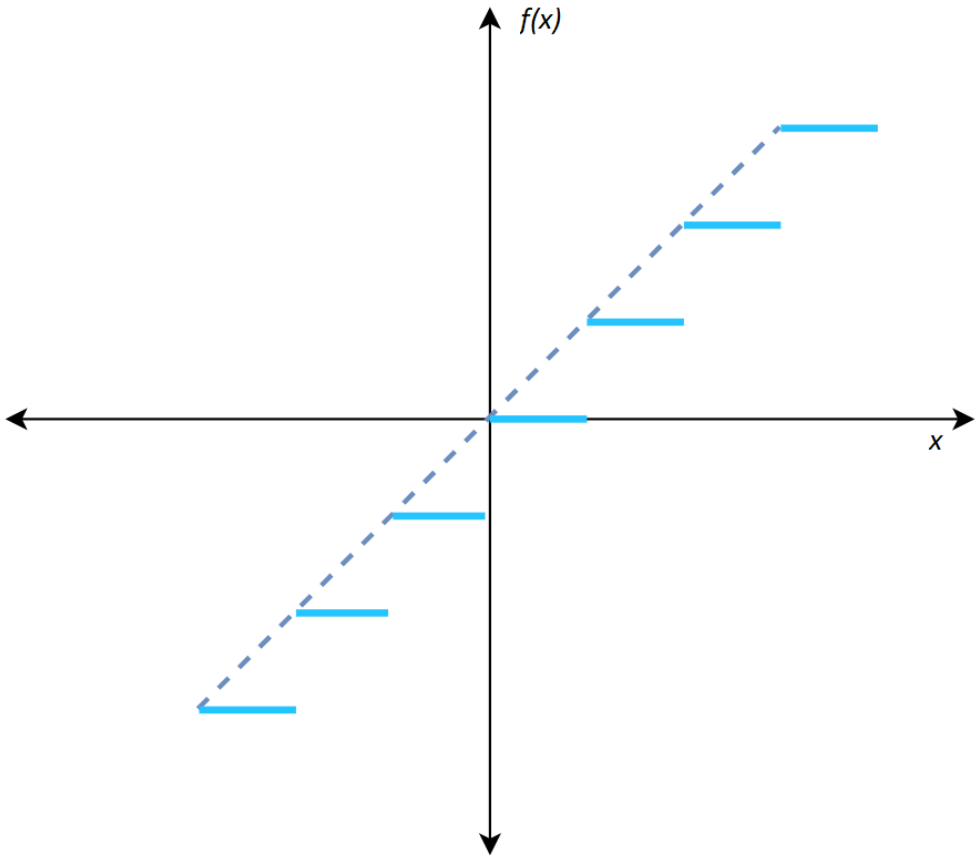

Figure 2.3: **Discrete Steps From Quantization**. An illustrative example of how quantization uses a set of discrete steps to approximate a continuous function/signal.

error if they do not fall exactly on one of the quantization levels whereas values that fall outside the defined range are clamped to the range's endpoints, thus introducing clamping errors. Various works have explored different quantization algorithms [12, 24, 34] to minimize the loss of information when mapping CNN weights and activations into a discretized space. The ability to convert CNN computation entirely to 8-bit fixed point arithmetic means that engineers can easily make use of existing hardware such as DSP blocks which have been optimized over decades for massively parallel, fixed point integer arithmetic. The use of DSPs as co-processors/accelerators for CNN inference has provided significant speed and power benefits as DSPs have been optimized for efficiently processing millions of pixels in images and videos in real-time. However, directly converting the floating point CNN model to quantized integer arithmetic for inference as-is can sometimes lead to unacceptable degradation in accuracy.

The trained distribution of values in CNN weights and activations are typically clustered around zero in non-uniform distributions. Thus, a uniform, affine quantization encoding is actually sub-optimal from an information theoretic point-of-view. It would make sense to place more quantization-levels where there are many real-values and allocate less levels in the outlier domains. There have been works such as those in [12, 34] that seek to create more complicated, non-linear quantization schemes (or with Vector Quantization [12], even non-scalar) such as Log-2 based quantization that provide significant theoretical benefits over uniform quantization for retaining floating-point accuracy. However, mobile hardware accelerators are usually limited in the types of operations that can be parallelized for fast execution. Thus, the aforementioned quantization methods are often not supported by existing mobile hardware. As such, other works have focused on quantization-method-specific optimization (e.g., targeting 8-bit uniform quantization). These include quantization-aware fine-tuning [24] and differential optimization of quantization parameters [4, 25], e.g., finding the optimal max/min values of each layer for minimal quantized degradation. These methods train a model that is robust to quantized perturbations by simulating the error/noise of fixed point arithmetic to make them ready for on-device deployment. In our work, we will also be focusing on uniform 8-bit quantization of CNN weights and activations.

In this chapter, we have seen the dramatic costs of CNN training and inference. Besides needing to make CNNs "fit" on hardware in terms of memory, storage and MACs, the power and time consumption of these algorithms is massive. We have also seen how fixed point quantization is a very promising and straightforward method for significantly reducing the latency and power of CNN inference. However, quantization is a noisy process that can introduce non-trivial, and sometimes even catastrophic error. The next chapter will describe the framework we propose for better understanding the sources of this quantization noise and how that can be leveraged for quantization-aware design of CNNs.

Chapter 3

Fine-grained, Layerwise Analysis of Weights/Activations Distributions to Better Understand Quantization Noise in the System

We seek to better understand how various design decisions for improving efficiency of our convolutional neural networks can affect the expected behaviour of deployed models. Namely, how do decisions such as training hyperparameters or architecture design affect the inference behaviour of a trained model under 8-bit, uniform quantization of weights and activations? In a common design flow, design choices are made based on the observed fp32 behaviour of a model. Trade-offs between accuracy, latency, power and other hardware performance metrics are primarily made based on offline testing with fp32 computation. In recent years, the increased importance of quantization for edge deployment of CNNs has led to increased research on quantization-simulation [4, 24, 25] and improved quantization methods [12, 34, 35].

Quantization simulation can be very powerful as it allows for a direct simulation of how the CNN is expected to perform under quantization. Furthermore, methods such as [4, 21, 24, 25] have successfully devised ways to train "quantization-aware" CNNs and enable models to adapt to error induced by quantization noise. However, quantization simulation is bit-width specific and may not necessarily transfer to a given target device if the simulated quantization does not match the on-device computation. Thus, insights gained from quantization simulation are not guaranteed to transfer to different settings.

Another viable desigh flow is to have "quantization-in-the-loop" wherein effects of quantization are simulated on the GPU or candidate models are quantized for on-device testing and re-iteration of design. However, such a design flow is also bit-width and device-specific. While the aforementioned methods are incredibly useful and well established, it would also be useful to glean some more general, lower-level insights beyond a CNN's output behaviour under quantization.

In most cases, quantization behaviour of the model is only observed at a coarse-grained, high-level/model-level viewpoint. That is, ablation studies and comparisons are done by observing the inference outputs of the fp32 and quint8 models and comparisons are largely made based on the difference in floating point output and quantized 8-bit output of a given CNN. For example, the quantized mean squared error (QMSE) of a given CNN would be the mean squared error between the fp32 model output and the quint8 model output. Another commonly used measure is the quantized KL-divergence (QKL-Div) which is usually the average KL-divergence between the fp32 model softmax outputs and the quint8 model softmax outputs. Whereas QMSE directly quantifies quantization error, QKL-Div could be interpreted as a measure of distributional shift between the fp32 model outputs and the quint8 model outputs. Note that as most works are concerned with classification networks, the model outputs are usually softmax outputs which can be interpreted as a distribution. In such cases, KL-divergence can be directly calculated with the model output tensors and the mean KL-divergence computed. However, quantized KL-divergence becomes less easily computed if the model output is not a softmax or some other distribution. In such cases, one would typically need to generate a histogram of the fp32 model and quint8 output distributions. From this point, KL-divergence can be computed as usual.

In some cases, CNN quantization [25] will also show plots of some layerwise distributions but they are typically included for illustrative purposes and do not analyze the CNN layers in a fine-grained manner. However, as CNN algorithms are complex, multi-layer systems with millions of parameters, high-level analysis of quantized model behaviour does not adequately capture the rich interactions of the CNN with quantization noise and the resulting dynamics that emerge as a result. For example, how might quantization error in an early layer propagate through the CNN and ultimately affect the output of the model? Can it be amplified or attenuated and what might lead to such effects? These questions are not easily answered unless we go to a more fine-grained level of analysis.

Thus, we would like to go to a lower-level of abstraction and better understand quantization dynamics at a layerwise and even channelwise level. In this work we propose a novel fine-grained analysis framework for understanding the layerwise distributions of CNN weights and activations. From this analysis, we wish to gain detailed insight on the layerwise distributions of final trained weights and activations. This information can give us an

in-depth look at how the learning dynamics of various models play out. For example, the dynamic ranges of each weight/activation tensor determine the resolution of the quantized step-size and, by extension, the quantization noise in a CNN. Thus, this analysis can help explain the observed quantized inference behaviour of different trained models.

We propose systematically ablating through a variety of hyperparameter/design choices while tracking the dynamic ranges of each layer's weights and activations during and after training. In this way, we can isolate the effect of these different choices and analyze the changing distributions at each layer. We also track the "average channel precision". Average channel precision is defined as Eq. 3.1 where $range_i$ refers to the dynamic range of channel-i in a convolutional weight tensor with C output channels and $range_{tensor}$ refers to the dynamic range of the entire convolutional weight tensor. Channel precision in this context is the ratio between an individual channel's range and the range of the entire layer. Nagel et al. [35] use this precision quantity to algorithmically maximize the channel precisions of each layer in a network prior to quantization. It can be seen as a measure of how well the overall layer-wise quantization encodings represent the information in each channel.

$$average\_precision = \frac{1}{C}\sum_{i=1}^{C}\frac{range_i}{range_{tensor}} \tag{3.1}$$

For dynamic ranges of activations, we randomly sample N training inputs from our training set and observe the corresponding activation responses. To reduce outlier noise, we perform symmetric percentile clipping (e.g., top and bottom 1%) and track the dynamic range and average precision of the clipped activations. As percentile clipping has become a ubiquitous default quantization setting we feel that this method establishes a realistic baseline of what can be expected during inference-time. Finally, there is one more set of dynamic ranges that must be observed. Applying batch normalization (BatchNorm) [23] after each convolutional layer has become the best-practice in a large range of CNN algorithms. However, the typical method of applying the BatchNorm layer after convolution is not well-suited for mobile hardware processing. Best practice for fast CNN inference usually involves folding/fusing the scale and variance parameters of a BatchNorm layer into the preceding layer's convolution parameters prior to quantization. The method for obtaining BatchNorm-folded weights is shown in Eq. 3.2 where γ is the BatchNorm scaling parameter [23], w is the weight tensor, σ_B^2 is the variance of the layer's activations for a given batch B, and $EMA()$ refers to the exponential moving average computed across batches over training. The term ϵ is a small constant for numerical stability. Essentially, the scaling that is normally applied to each channel of activations can be applied to the out-

put channel of convolutional weights instead. Thus, BatchNorm-folding effectively removes BatchNorm layers from the network. Linearity guarantees that this folding operation is mathematically equivalent to normal Conv-BatchNorm operation. Note that this is only valid if the BatchNorm layer is placed directly after the convolution, prior to any non-linearity/activation function being applied. It is the BatchNorm-folded weights that are being quantized and run on-device. Therefore, we must also track the dynamic range and precision of our CNN's batchnorm-folded (BN-Fold) weights.

$$w_{fold} = \frac{\gamma w}{\sqrt{EMA(\sigma_B^2) + \epsilon}} \tag{3.2}$$

In this manner, we can iterate through various designs and configurations, gaining insights at each step on the trained models and their learning dynamics as well as the final weights and activations distributions. Our method can be extended as a framework to analyze a plethora of different design choices. These can include architecture choices such as layer-type, skip/residual connections as well as training hyperparameters such as random weight initialization method, learning rate schedules, batch-size, and optimizers. Despite their simplicity, such analyses can provide deep insight on the interplay of these various design choices and perhaps yield new understanding on their interaction. In the following chapters we will demonstrate different applications of our analysis method.

$$SQNR(x) = 10 \log \left(\frac{E[x^2]}{E[\delta_x^2]}\right) \tag{3.3}$$

Several groups perform a layerwise analysis of the signal-to-quantization-noise-ratio (SQNR) in a CNN [29, 32, 43]. They use SQNR, defined in Eq. 3.3 where δ_x is the quantization noise/error of x, to estimate the amount of useful information passing from layer to layer in a CNN after quantization. While Lin et al. [29] and Meller et al. [32] mainly use SQNR to perform mathematical analysis and make numerical decisions related to quantization, Sheng et al. [43] use a layerwise SQNR analysis to identify architectural choices that were hurting the quantized performance of MobileNets-v1 before retraining a modified MobileNets architecture. Our method can be seen as expanding on this approach and going to an even lower level, directly analyzing the distributions at each layer. This insight can help guide further exploration for quantization-based optimizations such as the above-mentioned works or provide a baseline expectation of quantized accuracy trade-offs in scenarios where tight timelines/limited resources may force engineers to deploy their quantized model as-is.

Chapter 4

Analyzing the Effect of Random Weight Initialization on Trained Layerwise Distributions and Behaviour of the Quantizated Models

4.1 Random Weight Initialization Methods

As we have described in Chapter 3 the motivations for analyzing the layerwise distributions of a CNN for understanding quantization, it would seem natural to first demonstrate the application of our analysis to a hyperparameter that directly influences the layerwise weight distributions of a neural network. Random weight initialization is an often overlooked, but crucial hyperparameter of neural network training. Weight initialization methods decide where we should begin on the loss surface and in the model-space. They directly influence the starting point of each layer's distributions. Thus, it seems reasonable to expect that even for the same CNN architecture, the choice of random weight initialization method can significantly affect the quantization behaviour of the fully-trained network.

Weights initialization strategies are often designed with the goal of solving issues such as vanishing/exploding gradients [11, 13, 15]. However, another aspect of weights initialization is its impact on the final trained distributions of each layer. As they determine our starting point on the loss surface and initial conditions, initial distributions of each weight tensor will have a profound impact on the final trained model. Gradient descent is an incremental process with many small, noisy steps. Thus, an intelligent weight initialization strategy

will have significant impact on the local minima reached on our path through the loss space and consequently the model space as well. With regards to quantization, this means that weight initialization choices could have significant impact on the dynamic ranges and data distributions of the weights and activations in a trained CNN. Thus, affecting the noise in our system and the expected quantized inference behaviour.

We use our proposed framework for in-depth, fine-grained quantitative analysis of the impact of various weights initialization strategies on final accuracy and quantized behaviour. By analyzing the trained distributions of each layer's weights and activations, we can gain deep insight on how different weights initialization strategies will affect the dynamic ranges of each layer. This in turn provides insight on the quantized behaviour of a CNN. Furthermore, we analyze the effect of these different weights initializations for a small set of different CNN architectures. Thus, we are able to isolate and observe the interplay between the CNN architecture choices (the parameterization) and the weights initialization strategy (the starting point on the parameterized loss surface). To our best knowledge, we are the first to perform such a systematic, low-level, quantitative analysis of various weights initialization strategies and their effect on quantized behaviour.

4.2 Background

In early research, neural network parameters were often randomly initialized based on sampling from a unit normal or uniform distribution. The respective variance and range of these distributions would be hyperparameters for the practitioner to decide. While easily taken for granted, researchers have provided mathematical proofs showing how intelligent weights initialization strategies can solve issues of vanishing and exploding gradients [11, 13, 15]. At initialization, if we assume linearity of each layer, we can model the neural network's forward pass as Eq. 4.1 — a series of matrix multiplications of input variable X with random variables W_i (i.e., the randomly initialized weights). Typically, the initial weights W_i and the input X are assumed to be zero-mean distribution with variances $Var(W_i)$, $Var(X)$ respectively and we also assume that each column of W_i is independently sampled from the same distribution. Thus, it follows that the variance of the product distribution Y is the product of variances illustrated in Eq. 4.2 where n_i is the number of columns in matrix W_i. Similarly, the variance of the gradients in the backwards pass at initialization are also determined by the product of variances of the randomly initialized parameters.

$$Y = (\prod_{i=1}^{n} W_i)X \tag{4.1}$$

$$Y = Var(X) \cdot \prod_{i=1}^{n} n_i Var(W_i) \tag{4.2}$$

Thus, to prevent these series of matrix multiplications from pushing variances too high (exploding activations/gradients) or too low (vanishing activations/gradients), we would want each layer's weights to be initialized to a distribution with unit variance. This idea forms the core principle of different weight initialization research works. These research works define *fan_in* and *fan_out* of a fully connected layer as the number of input/output units respectively. For convolution, it is defined as Eq. 4.3 where K is the kernel width of a given layer's square convolutional kernel and $C_{in/out}$ is the number of input/output channels respectively. In the aforementioned works, they provide proofs on how their proposed fan_in/fan_out-aware initialization strategies scale the variance of gradients at each layer to roughly unit variance. Thus, avoiding failure modes created by vanishing and exploding gradients. Note that these proofs rely on some assumptions such as the modeling of NN layers as linear matrix multiplication done in [11]. Furthermore, in [11] the authors' experiments only use Tanh and Sigmoid activation and they assume that at initialization, each layer is operating in the linear region of the activations.

$$fan_{in/out} = K \times K \times C_{in/out} \tag{4.3}$$

In [15] the authors note that these linearity assumptions are inaccurate for Relu activations [36]. More specifically in the forward pass, the output of a single layer is the result of a matrix multiplication followed by applying a rectifier (see Eq. 4.4). The rectifier breaks linearity assumptions. Furthermore, the output of this layer and input to the subsequent layer is no longer zero-mean and we cannot assume a simple product of variances to be the distribution of the Relu-activated outputs. However, the authors note that if they assume W_i to be a symmetric distribution around zero and the layer's bias parameters are also zero, we can model the variance of the product distribution Y to be a product of variances scaled by $\frac{1}{2}$ as illustrated in Eq. 4.5. Evidently, there are some simplifying assumptions that must be made for these formulations. However, empirical evidence has shown tangible benefits to using such fan_in/fan_out-aware initialization strategies and they have become widely adopted in the field. The weight initialization methods described in [11, 15] have become some of the most popular random initialization methods. Both papers describe methods for random weight initialization by sampling from either Gaussian or uniform distributions. The distributions from which each of these methods sample their random weights are described in detail in Sec. 4.4

$$y = max(0, x) \tag{4.4}$$

$$Y = Var(X) \cdot \prod_{i=1}^{n} \frac{1}{2} n_i Var(W_i) \tag{4.5}$$

While the introduction of batch normalization [23] layers has greatly mitigated training issues involving gradient scales, the choice of "where to begin" in the parameterized loss space is still extremely relevant. An often-overlooked effect of these initialization strategies is their impact on the trained dynamic ranges of each layer. As gradient descent is a noisy, iterative process with small, incremental steps, the final distributions of each layer are profoundly impacted by their starting point.

4.3 Fine-grained Layerwise Analysis

Besides a high-level study of how different weight initializations affect 32-bit floating point (fp32) and eight-bit quantized (quint8) accuracy, we also wish to better understand the journey from initial distributions to the final distributions of each layer's trained weights and activations. We will systematically ablate through a variety of different weight initialization strategies and apply our fine-grained layerwise analysis framework to better understand how our choice of weight initialization method can ultimately impact post-training quantized inference behaviour. Note that while our main focus is analyzing the final trained distributions, we could also be tracking the distributions of each layer's weights and activations during training. This information can give us an in-depth look at how the learning dynamics of various weight initializations play out.

4.4 Experiment

For our experiment we use a simple, VGG-like macroarchitecture with four variations that differ in the micro-architecture of each layer (i.e., the type of convolution block used and the use of BatchNorm in-between convolution and Relu. See Figure 5.1 for the general macro-architecture). These four variations are determined by the type of conv-block used at each layer: Regular_Conv_With_BN, Regular_Conv_No_BN, DWS_Conv_With_BN, and DWS_Conv_No_BN. These respectively correspond to using regular convolution followed

by BatchNorm and Relu, regular convolution followed by only Relu and no BatchNorm, depthwise separable convolution blocks with BatchNorm and Relu after each convolution layer (same as the MobileNets block in [17]), and finally depthwise seperable convolution with only Relu and **no** BatchNorm after each convolution layer. The very first convolution layer stays fixed for all architectures, but follows the With/Without BatchNorm behaviour of the rest of the layers.

Our four CNNs are trained and tested on CIFAR-10 with a wide variety of different weight initialization strategies. These strategies can be separated into two categories of naive, straightforward strategies and more intelligent, layer-aware methods. Furthermore, the most common random weight initializations can also be categorized by the type of sampling distribution: random sampling from uniform distributions (hereafter referred to as RandUni) and random sampling from normal distributions (hereafter referred to as RandNorm). With considerations of dynamic range in mind, we seek to select distributions for the naive methods that would roughly correspond to small, medium, and large initial weights ranges. For the layer-aware initialization strategies, we use four commonly used methods introduced in [11, 15]. Named after the authors, we call them Glorot Uniform (GlorotUni) and Glorot Normal (GlorotNorm) from [11], He Uniform (HeUni) and He Normal (HeNorm) from [15]. See Equations. 4.6, 4.7, 4.8, 4.9 for the respective sampling distributions for obtaining initial weights. In these works, the distribution range (for uniform sampling) and standard deviation (for normal sampling) for each layer are calculated based on *fan_in*, *fan_out*, or some combination of the two. We choose to focus on only the convolution layers and so the fully connected layers are always initialized using Glorot Uniform initialization. Furthermore, we also keep the weight initialization of the first convolution layer constant; only Glorot Uniform initialization is used. This was to keep the very first convolution layer as constant as possible.

$$W_{GlorotUni} \sim \mathcal{U}[-\sqrt{\frac{6}{fan\_in + fan\_out}}, +\sqrt{\frac{6}{fan\_in + fan\_out}}] \tag{4.6}$$

$$W_{GlorotNorm} \sim \mathcal{N}(0, \frac{2}{fan\_in + fan\_out}) \tag{4.7}$$

$$W_{HeUni} \sim \mathcal{U}[-\sqrt{\frac{6}{fan\_in}}, +\sqrt{\frac{6}{fan\_in}}] \tag{4.8}$$

$$W_{HeNorm} \sim \mathcal{N}(0, \frac{2}{fan\_in}) \tag{4.9}$$

Initialization Method	Standard Deviation	Max/Min Value	C
RandNorm Large	1	N/A	N/A
RandNorm Med	0.5	N/A	N/A
RandNorm Small	0.1	N/A	N/A
RandUni Large	N/A	+/- 1	N/A
RandUni Med	N/A	+/- 0.5	N/A
RandUni Small	N/A	+/- 0.25	N/A
ModGlorotUni Large	N/A	N/A	1296
ModGlorotUni Med	N/A	N/A	36
GlorotUni	N/A	N/A	N/A
GlorotNorm	N/A	N/A	N/A
HeUni	N/A	N/A	N/A
HeNorm	N/A	N/A	N/A

Table 4.1: List of the various weight initialization strategies used. For methods that require some hyperparameter selection we include the values selected.

Based on initial results showing Glorot Uniform having the most success in fp32 accuracy, we further experiment with Modified Glorot Uniform (ModGlorotUni) weights initialization strategies. The method of computing the max/min range of the uniform sampling distribution in Glorot Uniform initialization can be generalized as Eq. 4.10 where C is some constant. In the original paper, $C = 6$. Following our established method of selecting distributions corresponding to small, medium, and large initial weights ranges, we select two values of C that would roughly correspond to medium and large ranges. The original Glorot Uniform leads to fairly small ranges. See Table 4.1 for a detailed breakdown of the sampling methods used in each of the 48 experiments.

$$max/min = \pm\sqrt{\frac{C}{fan\_in + fan\_out}} \tag{4.10}$$

Each network is trained for 200 epochs of SGD with Momentum = 0.9 and batch-

size = 128. Initial learning rate is 0.01 and we scale it by 0.1 at the 75th, 120th, and 170th epochs. For the activation range tracking we perform top/bottom 1% clipping computed on a random sample of 1024 training samples. Basic data augmentation includes vertical/horizontal shift, zoom, vertical/horizontal flip and rotation. We use Tensorflow-1.15 for training and quantizing the weights and activations to quint8 format.

For each network we evaluate testing performance with respect to 4 metrics: fp32 accuracy, quint8 accuracy, quantized mean-squared error (QMSE), and quantized crossentropy (QCE). Results are presented in Table 4.2. QMSE refers to the MSE between the fp32 network outputs and the quint8 network outputs after dequantization. Similarly, QCE measures the cross entropy between the fp32 network outputs and the dequantized quint8 network outputs. While QMSE directly measures how much the quint8 network outputs deviate from the fp32 network, QCE quantifies the difference in the distribution of the network outputs. Together, QMSE and QCE can help us understand by how much the quantized network and floating point network outputs differ as well as how the shapes of each network's outputs diverge from each other. For classification tasks, the quantized network can predict the same class as the fp32 network, despite deviations in logit values, if the overall shape of the output distribution is similar. Therefore QCE can sometimes be more reflective of differences in quantized behaviour. Additionally, we also observe the relative percent accuracy degradation (see 4.11) of each network after quantization. Though these quantities often track together, there can be scenarios where a network with more QMSE or QCE actually has less relative quantization degradation from a pure accuracy standpoint. This could be explained by favourable rounding within the network leading to the correct top-1 class prediction despite relatively large errors in the quint8 network outputs and possibly shifting distributions of values among other class probabilities.

$$percent\_acc\_decrease = \frac{|Accuracy_{fp32} - Accuracy_{quint8}|}{Accuracy_{fp32}} \times 100 \tag{4.11}$$

4.5 Discussion

We can see in Table 4.2 that besides affecting the final FP32 accuracy of a given CNN architecture, the weights initialization strategy also has significant impact on the QUINT8 accuracy. Particularly worth noting is the markedly improved quantized behaviour in the DWS_Conv_With_BN networks trained using RandUni_Large initialization. Equally noteworthy is the stark drop in QUINT8 accuracy observed with the DWS_Conv_With_BN

Network Architecture	FP32 Accuracy	QUINT8 Accuracy	QMSE	QCE	Percent Accuracy Decrease
DWS_Conv_No_BN_GlorotUni	10.00	10.00	0.000	2.303	0.00
DWS_Conv_No_BN_ModGlorotUni_Large	74.42	70.70	0.009	1.109	5.00
DWS_Conv_No_BN_RandNorm_Large	69.94	63.07	0.006	0.893	9.82
DWS_Conv_No_BN_RandNorm_Med	74.68	73.26	0.006	0.951	1.90
DWS_Conv_No_BN_RandUni_Large	75.22	72.77	0.004	0.872	3.26
DWS_Conv_With_BN_GlorotNorm	80.10	69.76	0.014	1.127	12.91
DWS_Conv_With_BN_GlorotUni	81.04	71.02	0.012	1.054	12.36
DWS_Conv_With_BN_ModGlorotUni_Large	76.33	68.64	0.012	1.789	10.07
DWS_Conv_With_BN_ModGlorotUni_Med	80.16	70.86	0.014	1.011	11.60
DWS_Conv_With_BN_HeNorm	79.48	55.56	0.033	2.400	30.10
DWS_Conv_With_BN_HeUni	80.51	62.49	0.024	1.786	22.38
DWS_Conv_With_BN_RandNorm_Large	74.93	66.32	0.010	1.358	11.49
DWS_Conv_With_BN_RandNorm_Med	77.99	66.32	0.016	1.694	14.96
DWS_Conv_With_BN_RandNorm_Small	80.61	70.12	0.013	1.464	13.01
DWS_Conv_With_BN_RandUni_Large	76.60	74.18	0.003	0.811	3.16
DWS_Conv_With_BN_RandUni_Med	78.40	67.82	0.016	1.993	13.49
DWS_Conv_With_BN_RandUni_Small	79.02	64.25	0.017	1.452	18.69
Regular_Conv_No_BN_GlorotNorm	87.03	84.46	0.005	0.585	2.95
Regular_Conv_No_BN_GlorotUni	86.89	85.51	0.003	0.403	1.59
Regular_Conv_No_BN_HeNorm	86.20	85.56	0.001	0.228	0.74
Regular_Conv_No_BN_HeUni	86.20	85.89	0.006	0.485	0.36
Regular_Conv_With_BN_GlorotNorm	89.34	86.33	0.005	0.340	3.37
Regular_Conv_With_BN_GlorotUni	88.53	88.33	0.002	0.207	0.23
Regular_Conv_With_BN_ModGlorotUni_Large	60.35	57.03	0.005	1.920	5.50
Regular_Conv_With_BN_ModGlorotUni_Med	84.60	60.08	0.029	3.217	28.98
Regular_Conv_With_BN_HeNorm	86.87	86.30	0.003	0.311	0.66
Regular_Conv_With_BN_HeUni	87.88	86.47	0.004	0.693	1.60
Regular_Conv_With_BN_RandNorm_Large	55.41	45.43	0.009	2.070	18.01
Regular_Conv_With_BN_RandNorm_Med	59.57	56.55	0.001	1.465	5.07
Regular_Conv_With_BN_RandNorm_Small	80.19	68.96	0.017	2.016	14.00
Regular_Conv_With_BN_RandUni_Large	58.69	58.15	0.002	1.577	0.92
Regular_Conv_With_BN_RandUni_Med	67.03	66.15	0.002	1.260	1.31
Regular_Conv_With_BN_RandUni_Small	76.28	75.80	0.002	0.888	0.63

Table 4.2: Detailed results for each combination of weight initialization strategy and CNN architecture. The initialization strategies that suffered from vanishing/exploding gradients are ommitted. DWS_Conv_No_BN_GlorotUni is kept for illustrative purposes.

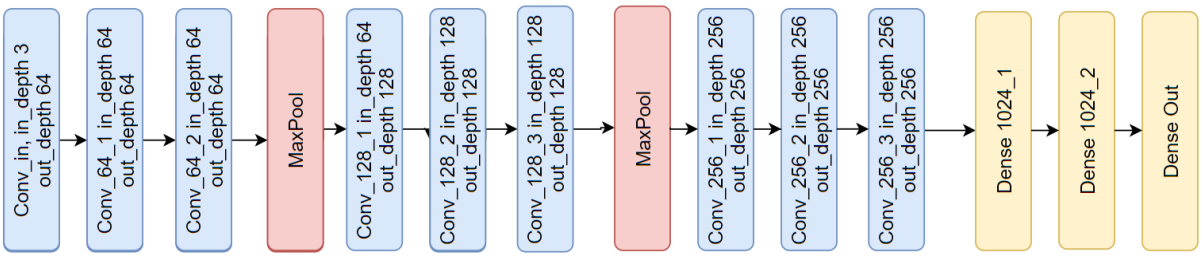

Figure 4.1: **General Macroarchitecture of the CNN**. For our analysis we use a fixed macro-architecture so that we can isolate the interactions between various weight initialization strategies and a few different convolutional layer choices. A flatten layer is used between the final Conv and first Dense layer.

networks trained using the HeNorm and HeUni weight initialization methods and the Regular_Conv_With_BN network trained with ModGlorotUni_Med initialization. As expected, quantized accuracy usually worsened when BatchNorm layers were introduced. This is often attributed to the increased dynamic ranges/distributional shift introduced by BatchNorm Folding. BatchNorm Folding can sometimes lead to large increases in the dynamic ranges of a given layer's convolutional weights and significant distributional shift of the quantized filter values. Thus, leading to increased error and information loss.

While each CNN architecture is trained on twelve different initialization methods, Regular_Conv_No_BN only has four results. This is because the other initialization methods had issues of exploding gradients. Their results were omitted. It is possible that with a sufficiently small learning rate, these models would have been able to train. However, for the sake of our systematic ablation study, we focused on isolating just the effects of weight initialization method. Most of the DWS_Conv_No_BN experiments also did not learn but suffered from vanishing gradient issues instead. However, in our analyses we found that these vanishing gradients were not necessarily caused by a deep architecture leading to the gradient progressively vanishing during backpropagation. Instead, we observed a "vanishing activations" type phenomenon in the *forward* pass wherein the activations of the final Depthwise Separable Convolution block are exceedingly small. Thus, no gradients are able to propagate past the fully connected layers. Figure 4.2 shows a plot of the network activations in DWS_Conv_No_BN_GlorotUni. For illustrative purposes, we keep the DWS_Conv_No_BN_GlorotUni result and omit the rest. The normalization introduced by BatchNorm alleviates this issue as expected. One could consider an additional aspect of BatchNorm as adding capacity to the network in the form of a learned (if BatchNorm scale γ is used), or layer-dependent **explicit scaling**. Scaling that would otherwise be

too difficult for the convolution parameters to learn in addition to extracting features. We seek to follow-up on this hypothesis in future works. While we focus on the variations in quantized behaviour in this work, the varying FP32 accuracies are also worthy of close study. Our method sets out a framework through which we can systematically study these phenomena.

To better understand why we are observing the given quantized behaviour, we can use the proposed fine-grained analysis and inspect the distributions of each model layer-by-layer. These layerwise plots enable a more low-level, focused look. As we look to analyze any anomalies or unexpected behaviour, our fine-grained approach allows us to gain much more detailed insight as to what dynamics are at play when we introduce quantization noise. With regards to the significantly improved quantized accuracy for the depthwise seperable network, DWS_Conv_With_BN_RandUni_Large, we observe in Figures 4.3, 4.4 and 4.5 that weights ranges don't necessarily tell the whole story. Despite having generally larger weights ranges, we start to see several other key areas in which the RandUni_Large layers stand out. For example, while the two He-initialized models tend to have a spike in the BN-Fold weights range at layer 2, RandUni_Large actually decreases in range. Furthermore, when we compare the BN-Fold weights precisions we also see a drop in precision for the other networks at layer 2 while the precision for RandUni_Large increases. With the activations, we see that all of the activation ranges increase at layer 2 while activation precisions decrease. However, RandUni_Large experiences a significantly smaller drop in activation precision. Thus, suggesting that RandUni_Large has a higher retention of information in those crucial early stages of low-level feature extraction. Also of note is the significant drop in quantized accuracy for the He-initialized DWS_Conv models. This large quantization error serves to illustrate how sometimes strong gains in floating point performance can easily be lost at deployment time when the on-device computation differs from the offline testing. Thus, necessitating the need for robust quantized testing and more quantization-aware design methodologies.

Analyzing inter-model changes in the layerwise distributions might explain *why* we observe such a wide range of behaviour caused by varying weight initialization. From Figures 4.3, 4.5 we can see how our models end up with widely varying layerwise distributions. It would also be worthwhile to observe the relative change in range/precision after BatchNorm folding. This would be a proxy for observing the distributional shift of the weights. While it is intractable to pinpoint any single reason, our layer-level analysis reveals a rich set of interactions that build a detailed picture of each network's system dynamics as well as inter-network trends. We could further expand our analysis to more rigorous, yet scalable statistics. For example, we know that a uniformly distributed tensor would best utilize the quantized steps of our given discretization method. Thus, the KL-divergence between a

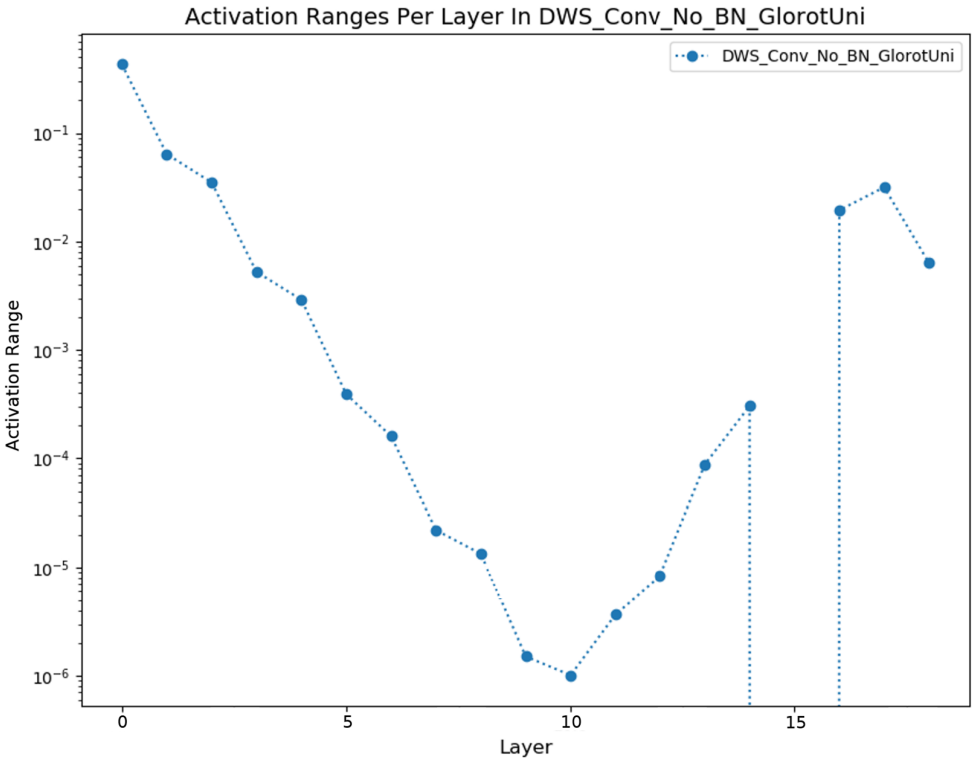

Figure 4.2: **Vanishing Act!** In this figure we can see how the activation ranges of DWS_Conv_No_BN become increasingly small until they practically disappear. Consequently, gradients are not able to propagate past the fully-connected layers (final three points on the graph).

given weight/activation tensor and its corresponding uniform distribution (i.e., a uniform distribution with the same bounds as the tensor) is a potential metric to explore. Overall, from these initial analyses, we see that a fine-grained, systematic approach to analyzing various design choices can yield detailed insights on the learning dynamics of a CNN.

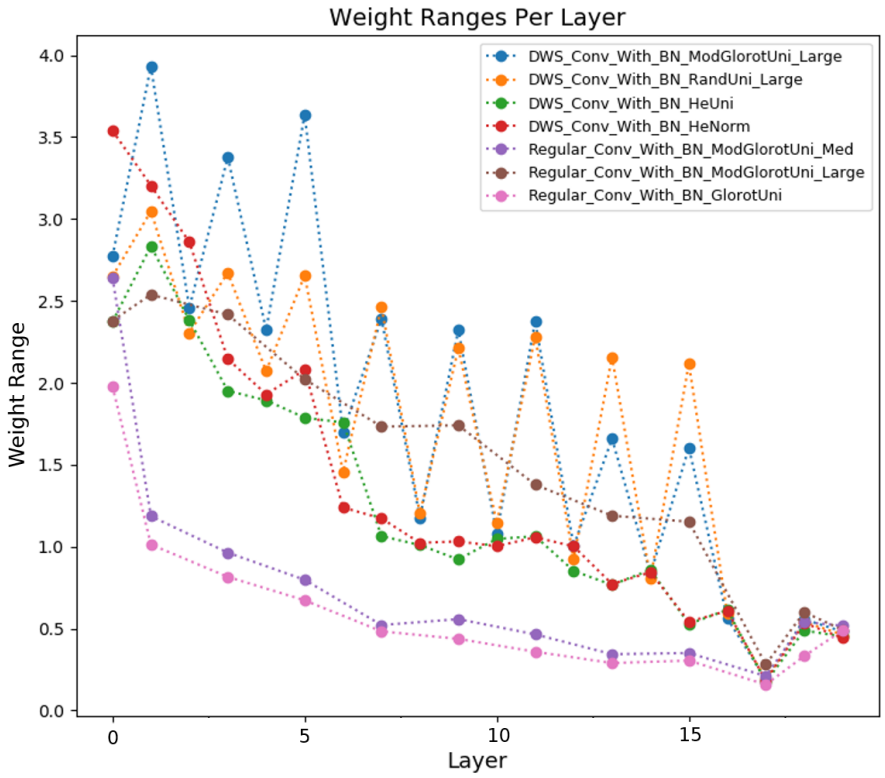

Figure 4.3: Layerwise plot of the dynamic range of convolutional filter weights for a selected subset of the trained models.

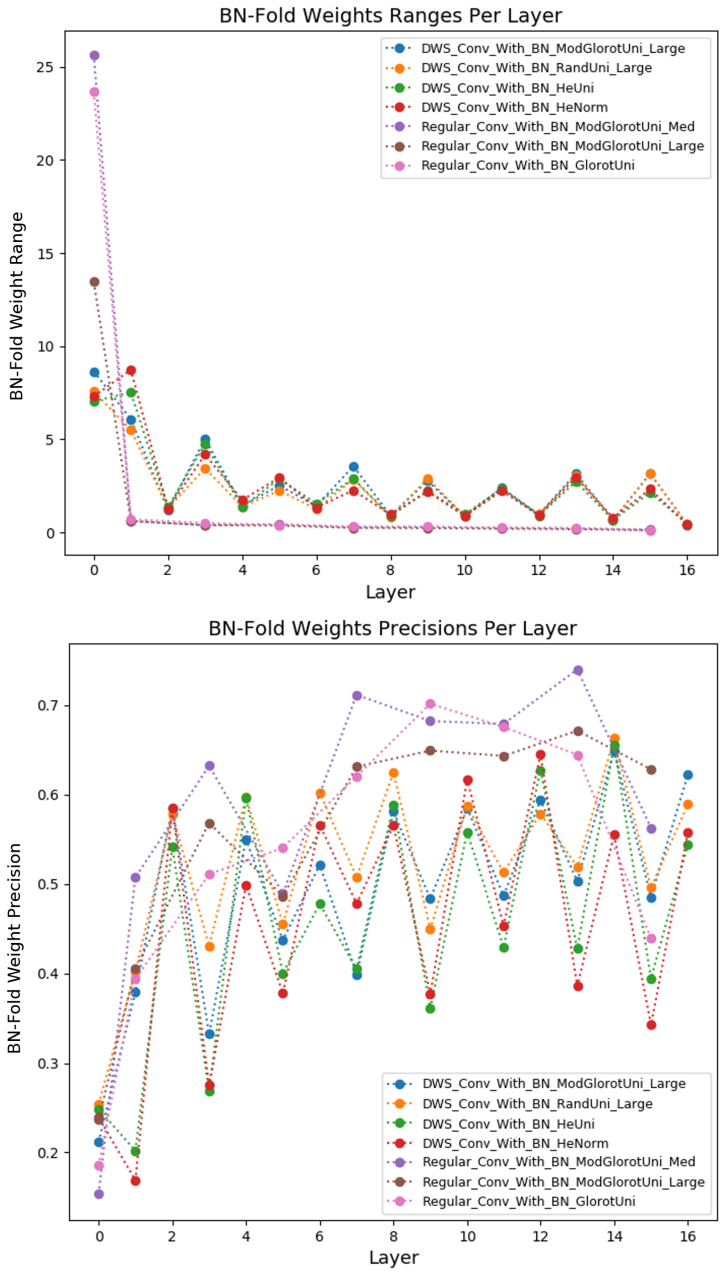

Figure 4.4: Layerwise plot of the dynamic range and average precision of BN-Folded convolutional filter weights for a selected subset of the trained models. By comparing to Fig. 4.3 we can see how BatchNorm Folding changes the distribution of convolutional weights that are actually being quantized.

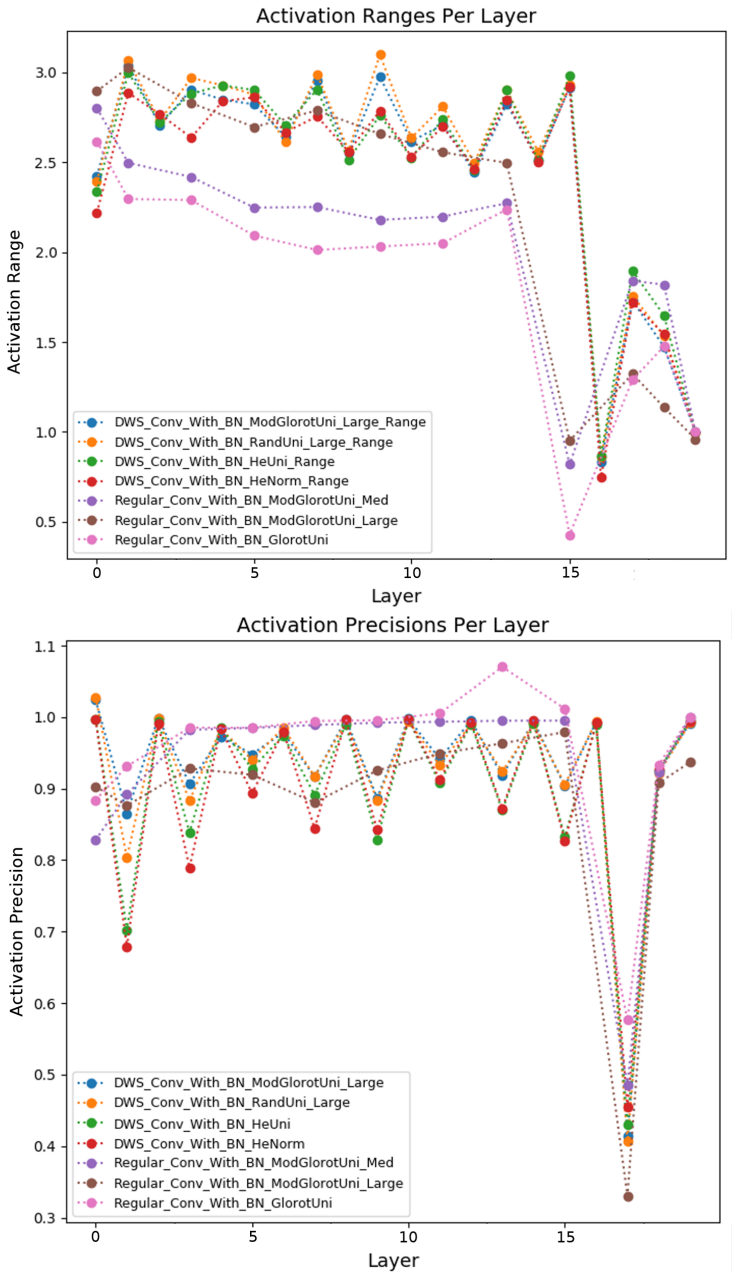

Figure 4.5: Layerwise plot of the dynamic range and average precision of convolutional layer activations for a selected subset of the trained models.

4.6 Quantized Behaviour Over Training Epochs

In addition to analyzing the fully trained behaviour of our CNNs, it would be useful to see how quantization error evolves as training progresses. If the quantization error of networks early-on in training can be predictive of the expected quantization error of fully-trained models, then there could be opportunities for predicting quantization error. This could be incredibly useful in applications such as neural architecture search (NAS) where there is significant interest in predicting CNN behaviour without needing to spend the costly GPU-hours involved with training to convergence.

For this study, we selected eight of the trained networks and quantized their corresponding checkpoints saved from Epochs {5, 50, 75, 150}. For each of these quantized checkpoints we computed their fp32 accuracy, quint8 accuracy, percent accuracy decrease (those three values in Table 4.3), and QMSE (see Table 4.4). These epochs correspond roughly to early, middle, and late stages of training. We would like to see if there are any trends in quantization degradation during different stages of training. The models were selected to evenly represent the four different architectural choices in our ablation study - with/without batchnorm and depthwise-separable convolution vs regular convolution. We also wanted to select the models to represent a variety of quantization degradation (i.e., large quantization error vs. small quantization error). We also show some plots of how each model's layerwise statistics evolve during training. However, due to space constraints we only include a randomly selected subset of plots for discussion. From plots of some of the layerwise distributions we can see that dynamic ranges and average precisions of our models do converge. However, the point at which these statistics stabilize can widely vary between layers and between models. We wonder if quantization error will converge before end-of-training. Figures 4.7, 4.6, and 4.8 show some examples of randomly selected layers from randomly selected models. We notice that while the values usually tend to stabilize, there can sometimes be large variations early-on in training. This is most pronounced in the activation ranges (Fig. 4.8) as well as plots for the weights of one of the depthwise-convolution layers (Fig. 4.6). Unfortunately, there is not enough space to show all layers and all models.

As expected, the quantization accuracy decrease and QMSE of our models in early training (Epoch 5) are not representative of our final trained models. For example, DWS_Conv_No_BN_RandNorm_Large only has a relative accuracy decrease of 0.41% at Epoch 5 compared to 9.82% decrease in the final trained model. In general, it seems to be hard to predict the quantized accuracy ranking of models based on intermediate quantization results before they are fully trained. This may be explained by the large variations in the layerwise distributions as observed in the aforementioned plots. However, by

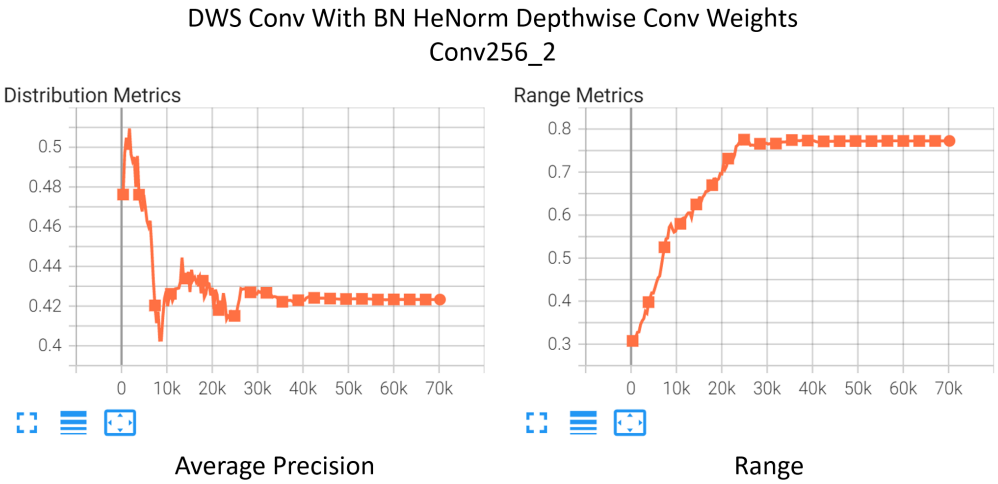

Figure 4.6: Plot of average precision vs training epoch (left) and range vs. training epoch (right) for randomly selected layer weights from a DWS Conv model.

Epoch 75, the quantization degradation of the worst model, DWS_Conv_With_BN_HeNorm is already apparent. Furthermore, when we observe the QMSE of each of these models over training in Table 4.4 we can identify the two worst models by Epoch 75 as well (DWS_Conv_With_BN_HeNorm and DWS_Conv_With_BN_GlorotUni). Perhaps intermediate quantization error can be used for early pruning of the models that have the worst quantized performance. Any method that can reduce the number of models that need to be fully trained would save a lot of compute/GPU-hours. It would be interesting to see if there were some other more refined metric that can better predict the expected quantization behaviour of our final models. It might also be helpful to get intermediate quantization results from more epochs. However, there would be a tradeoff as this will also increase quantization overhead.

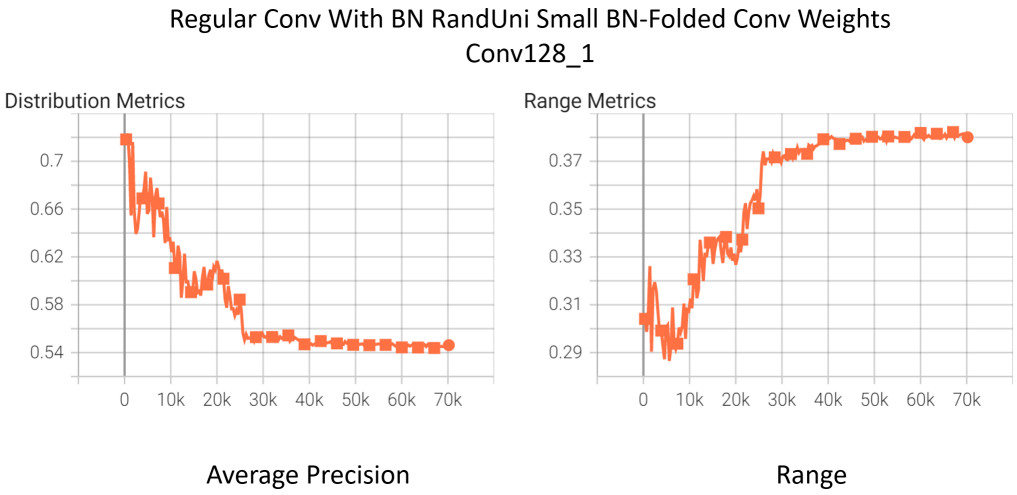

Figure 4.7: Plot of average precision vs training epoch (left) and range vs. training epoch (right) for randomly selected layer weights from a Regular Conv model.

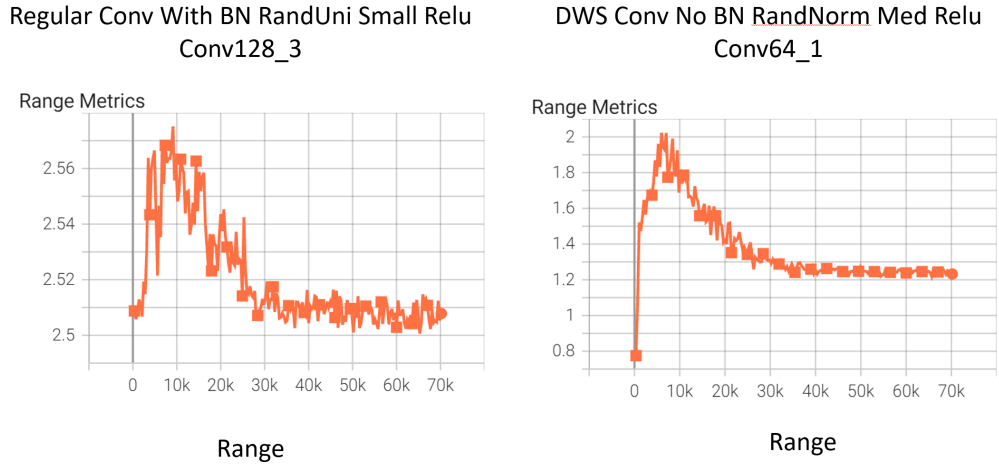

Figure 4.8: Plot of activation range vs. training epoch for randomly selected layers from a Regular Conv (left) and a DWS Conv (right) model. We can see how the activation distributions change quite a bit over the course of training.

Network	Epoch 5	Epoch 50	Epoch 75	Epoch 150	Epoch 200 (Final)
DWS_Conv_No_BN_RandNorm_Med	39.83/36.49/8.39	64.85/61.54/5.10	68.73/64.69/5.88	74.58/72.33/3.02	74.68/73.26/1.90
DWS_Conv_No_BN_RandNorm_Large	38.76/38.6/0.41	59.15/57.84/2.21	64.53/61.89/4.09	69.79/68.76/1.48	69.94/63.07/9.82
DWS_Conv_With_BN_HeNorm	38.60/37.90/1.81	73.75/65.12/11.70	75.15/58.44/22.24	79.76/68.71/13.85	79.48/55.56/30.1
DWS_Conv_With_BN_GlorotUni	41.25/38.2/7.39	74.7/68.7/8.03	77.46/70.0/9.63	81.07/70.18/13.43	81.04/71.02/12.36
Regular_Conv_No_BN_HeUni	47.25/46.56/1.46	80.21/79.92/0.36	82.12/80.87/1.52	86.38/86.61/0.27	86.20/85.89/0.36
Regular_Conv_No_BN_GlorotNorm	39.81/39.2/0.0153	79.66/76.80/3.59	83.11/78.12/6.00	87.11/85.73/1.58	87.03/84.46/2.95
Regular_Conv_With_BN_ModGlorotUni_Large	31.37/29.13/7.14	55.22/46.12/16.48	56.73/51.62/9.00	60.18/57.57/4.34	60.35/57.03/5.50
Regular_Conv_With_BN_RandUni_Small	39.88/39.92/0.10	68.85/64.55/6.25	71.75/63.66/11.28	76.42/66.97/12.37	76.28/75.8/0.63

Table 4.3: Results stated as (FP32 Acc/QUINT8 Acc/% Decrease) evolution of floating point vs quantized accuracy over the course of training. We'd like to see if there are any patterns that can be discerned early-on to predict quantization degradation of our final model.

Network	Epoch 5	Epoch 50	Epoch 75	Epoch 150	Epoch 200 (Final)
DWS_Conv_No_BN_RandNorm_Med	0.006	0.004	0.01	0.005	0.006
DWS_Conv_No_BN_RandNorm_Large	0.004	0.007	0.007	0.007	0.006
DWS_Conv_With_BN_HeNorm	0.003	0.011	0.023	0.020	0.033
DWS_Conv_With_BN_GlorotUni	0.004	0.010	0.014	0.016	0.012
Regular_Conv_No_BN_HeUni	0.001	0.004	0.003	0.001	0.006
Regular_Conv_No_BN_GlorotNorm	0.004	0.004	0.006	0.003	0.005
Regular_Conv_With_BN_ModGlorotUni_Large	0.002	0.005	0.003	0.003	0.005
Regular_Conv_With_BN_RandUni_Small	0.001	0.005	0.010	0.011	0.002

Table 4.4: QMSE at various epochs in training for a subset of the trained models.

4.7 Conclusion

We conduct the first in-depth, quantitative study of the impact of weight initialization strategies on final quantized inference behaviour of various basic CNN architectures. This study also serves as an example of how we can apply the finegrained, layerwise analysis framework introduced in Chapter 3. We show that in addition to affecting final floating point accuracy, a well-chosen weight initialization can also significantly affect a CNN's quantized accuracy. We also demonstrate potential for analyzing changing model dynamics/behaviours over the course of training as an additional avenue of insight. Future work includes further exploration of the interaction of BatchNorm with initial weight distributions, analysis of other intelligent initialization strategies, modifying other training hyperparameters, and analysis of weight initialization's impact on more complex architectures.

Chapter 5

Factorizing Convolution to Reduce the Parameters and Operations in a CNN While Monitoring the Impact on Quantized Behaviour

Convolutional neural networks require significant computational load. For example, the state-of-the-art ResNet-152 [16] requires over 5 *billion* MACs for a single inference. Consequently, researchers have sought to design architectures that will increase the computational efficiency of CNNs. Depth factorization of convolutional layers is an effective tool for reducing computational complexity that we will explore and analyze. In particular, we will investigate the trade-offs in MACs versus accuracy for both floating point and quantized inference and we will apply fine-grained layerwise analysis to better understand the interaction of depth factorization with quantization.

5.1 Depth Factorization of Convolutional Neural Networks

Following the recent explosion in deep learning research, there has been increased attention on complexity reduction strategies for deep convolutional neural networks (CNN) to enable inference on mobile processors. Quantization [24, 25, 35], and depth factorization [5, 17, 55, 57] have quickly emerged as two highly effective strategies for reducing

the power and computational budget needed for on-device inference. These two methods work orthogonally. As previously mentioned, fixed point quantization enables simple, low bit-width integer operations which are several times faster/less power than floating point (fp32) operations. Thus, quantization directly reduces the costs associated with performing the computation itself. For example, the cost of performing a single 3×3 convolution will be inherently lower when using fixed-point integer computation instead of floating point. Conversely, depth factorization reduces the actual number of CNN parameters and multiply-accumulate (MAC) operations that need to performed. Thus, depth factorization reduces the theoretical computation complexity of our CNN models. For depth factorization, we split the input channels into f groups and apply f groups of filters independently to their respective channel groups. For a given factorization rate, f, the number of MACs in a convolution layer with a $K \times K$ kernel goes from 5.1 to 5.2 (where $H \times W$ is the size of the output activation, C_{in}, C_{out} are the input-channel depth and output-channel depth respectively), thus reducing computation by a factor of f. For simplicity, our equations have excluded the MAC contribution from the pointwise convolution that typically follows the group convolution. Pointwise convolution is often used for the dual purpose of mixing channel information and changing (usually increasing) channel depth.

$$MAC_{conv} = K \times K \times H \times W \times C_{in} \times C_{out} \tag{5.1}$$

$$MAC_{factorized\_conv} = K \times K \times H \times W \times \frac{C_{in}}{f} \times \frac{C_{out}}{f} \times f \tag{5.2}$$

Depthwise separable convolution as described in MobileNets [17] has become a staple in efficient network design. It represents the extreme end of the depth factorization spectrum with one convolution filter per input channel. However, perhaps we do not always need to go to the extreme. A key tradeoff when designing CNNs for limited compute is efficiency vs. accuracy. As we scale down our architectures, we will necessarily lose accuracy. While depthwise separable convolutions are extremely efficient, they suffer from low data parallelism making them less suited to hardware acceleration. Also as mentioned in [5], they should not be assumed as the optimal point on the depth-factorization-spectrum. Furthermore, with quantization emerging as essential for on-device inference, we must consider the additional component of quantization error. In general, efficient architectures have so few parameters that they often suffer more quantized accuracy loss compared to higher complexity networks. However, there is still limited understanding of how different architectural choices impact quantized accuracy. Given the significant investment involved with

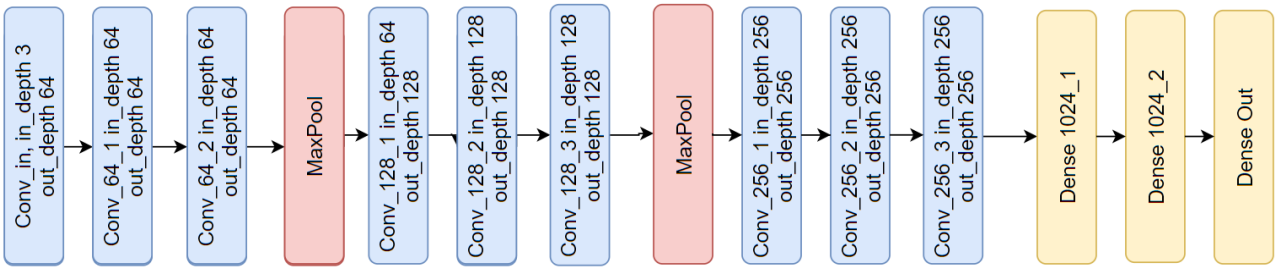

Figure 5.1: **FactorizeNet Macroarchitecture**. For our progressive, fine-grained analysis we start with a simple regular CNN and fix the macroarchitecture. We then progressively increase the level of factorization of each block using Groupwise Separable Convolution with varying *f*. The very first convolution layer stays fixed. A Flatten layer is used between final Conv layer and first Dense layer.

architecture search/design, it would be beneficial to gain detailed insights on the potential quantizability of an architecture during the design phase. Thus, helping speed-up the quantization optimization process.

We introduce a systematic, progressive depth factorization strategy for exploring the efficiency/accuracy trade-offs of scaling down CNN architectures under quantization and computation constraints. Starting with a simple, fixed macroarchitecture (see Figure 5.1) we algorithmically increase the granularity of depth factorization in a progressive manner while analyzing the final trained layerwise distributions of weights and activations at each step. Our proposed strategy enables a fine-grained, low-level analysis of layer-wise distributions to gain in-depth, layer-level insights on efficiency-accuracy tradeoffs under fixed-precision quantization. Furthermore, we can identify optimal depth-factorized macroarchitectures which we will refer to as **FactorizeNet**. While previous studies [19, 55] have performed ablation studies on the effect of different factorization choices on testing accuracy, they used a high-level approach and were mainly concerned with fp32 accuracy. [43] performs layerwise analysis of the signal-to-quantization-noise-ratio (SQNR) to identify layers that were hurting the quantized accuracy of MobileNets-v1 before retraining a modified MobileNets architecture. Our method can be seen as expanding on this approach and going to an even lower level, directly analyzing the distributions at each layer. Insights gained from such a fine-grained approach can help guide further exploration for quantization-based optimizations or provide a baseline expectation of quantized accuracy trade-offs when engineers deploy their quantized model as-is.

5.2 Related Works

Quantization and efficient CNN architecture design together have vastly enabled the use of CNNs in everyday life. Works such as MobileNet, ShuffleNet, and SqueezeNet, explore various ways to factorize the convolution operation to reduce overall MAC footprint and create efficient parametrizations while maintaining high accuracy. In fact, the authors in [22] outline a systematic, ablative approach to exploring the space of SqueezeNet architectures. They explore trade-offs such as 1×1 vs. 3×3 kernels, varying the dimensionality of Fire modules, and varying the Squeeze ratio. However, their method does not explore the depth factorization space and they limit their study to the high-level impact of various architecture choices on the final floating point accuracy. Besides manual design, research into automated search methods such as CondenseNet, DARTS, MnasNet, GenSynth, FBNets ([19, 30, 48, 52, 54]) have taken a neural architecture search (NAS) approach to search the design space for optimal solutions. Making use of algorithmic optimization approaches, these methods have further pushed the Pareto frontier of CNN design in the complexity/accuracy space.

In conjunction with efficient neural network architecture design methods, low bit-width (16 bits and below), fixed point quantization has enabled highly parallelized processors such as DSPs to run fast, low-power inference entirely with integer arithmetic. Methods such as log-quantization [34], uniform quantization (Tensorflow), vector quantization (Vector Quantization by Gong [12]) look to design various projection methods that maximize the representational power of our n-bit integers in attempts to recover floating-point accuracy. However, the noise induced by quantization error is still poorly understood and it can often be hard to predict which CNN architectures will quantize well aka are "quantization friendly." Given the problem of quantization robustness, various research works explore methods to increase the robustness of models to quantization noise. Some methods such as quantization-aware training (QAT) in [24] make use of simulated quantization to adapt the network to quantization noise at training time. This method essentially adds quantization noise as a regularizer that the CNN must adapt to. Trained quantization thresholds (TQT) [25] go a step further and also make use of simulated quantization and differential optimization to find the optimal min/max thresholds of each layer for uniform quantization. Data-free quantization (DFQ) [35] eschews quantization training and seeks to perform linear transformations on the weight distributions of each layer in order to reduce quantization error without changing mathematical behaviour of the network.

Our method falls at the intersection of these areas. It is a systematic, algorithmic approach to efficient CNN architecture exploration that utilizes fine-grained analysis to gain insight on how a given architectural choice might impact quantized inference behaviour.

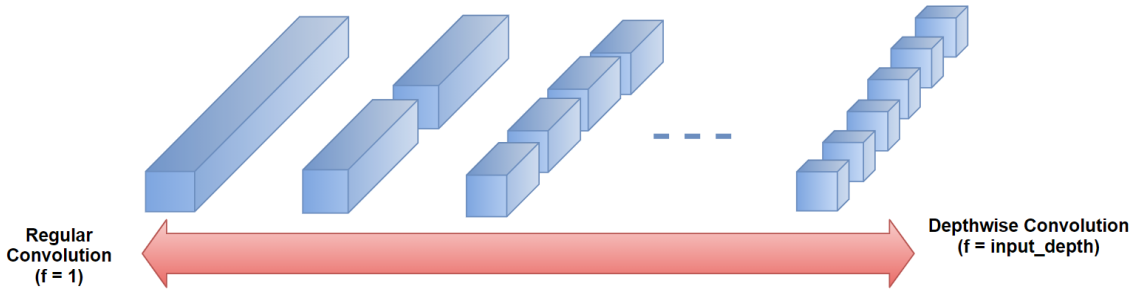

Figure 5.2: The depthwise factorization spectrum. On one end, we have regular convolution, with factorization rate of $f = 1$. On the other end we have depthwise convolution with factorization rate of $f = input\text{-}depth$. For a given layer in a CNN architecture, the optimal level of factorization could lie anywhere on this spectrum.

This insight can help guide further exploration for quantization-based optimizations such as the above-mentioned works or provide a baseline expectation of quantized accuracy trade-offs in scenarios where tight timelines/limited resources may force engineers to deploy their quantized model as-is.

5.3 Progressive Depth Factorization and Fine-Grained Layer Analysis

Consider a spectrum of depth factorization (see Figure 5.2) with regular convolution on one end (factorization rate $f = 1$) and depthwise convolution on the other (factorization rate $f = input\ depth$). As we turn the knob from $f = 1$ to $f = input\ depth$ for each layer or set of layers in a given macroarchitecture, we will observe a range of efficiency/accuracy trade-offs. Thus, a given CNN macroarchitecture is a search space in itself where a large range of factorization levels and combinations of factorizations can be realized to meet given efficiency-accuracy constraints. Besides searching for the optimal factorization configuration, we also wish to gain detailed insight on the impact of various factorization choices on the layer-wise distributions of final trained weights and activations. This information can help us understand which factorization settings are the most amenable to quantization as well as provide detailed insight on the response of various stages of a CNN to depth factorization. We propose algorithmically increasing the factorization of a given CNN macroarchitecture in a progressive manner while conducting a low-level analysis of the layerwise distributions for each level of factorization. At each factorization step, we

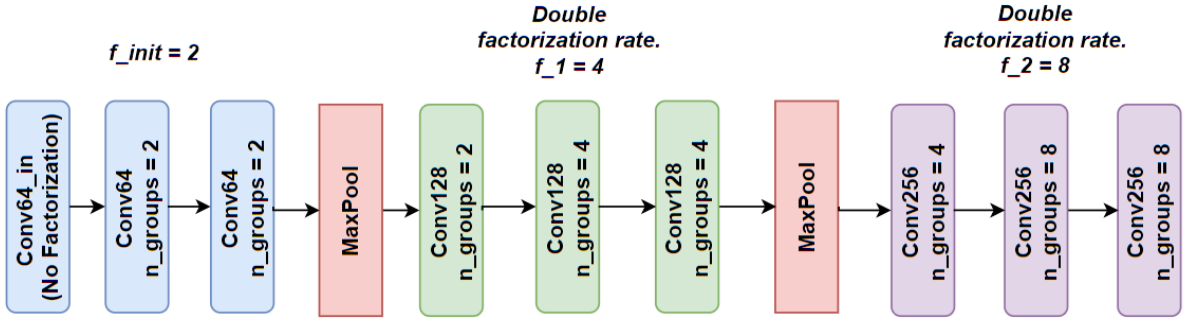

Figure 5.3: **Reverse Pyramid Factorization Scheme**. For this factorization scheme, we start with an initial factorization rate, f_{init}, and double the factorization rate each time the input depth doubles, thus preserving the number of channels per group throughout the network. For $f_{init} = 64$, we recover the depthwise separable CNN.

train the factorized CNN and track the distributions of weights and activations of each layer as described in Chapter 3. Making systematic, iterative changes to our CNN depth-factorization and observing the changing distributions should reveal insights on how various factorizations affect the learned distributions as well as the interactions of the factorized convolutional layer with quantization noise. For example, how will dynamic ranges change as factorization increases? Will BatchNorm-folded weight distributions exhibit similar behaviour at various levels of factorization? The iterative steps will allow us to isolate effects of different design choices and then use our fine-grained analysis to take a low-level, detailed look at the resulting changes in our trained model.

In this manner, we can iterate through progressively increasing factorization configurations, gaining insights on the efficiency/accuracy trade-offs at each step as well as the final layerwise distributions. Progressive Depth Factorization provides a general framework not only for systematically understanding the efficiency/accuracy trade-offs of factorization, but also for finding the optimal factorization configuration. As there are many directions that can be taken through the "Progressive Depth Factorization space", our method can be merged with automated search methods such as GenSynth [52] to trace out various paths through the space, especially for increasingly complex architectures.

5.4 Experiment

We start with a VGG-like macroarchitecture (see Figure 5.1) trained and tested on CIFAR-10. As we begin to factorize, the regular convolution layers (except for the first layer, which

stays constant) are replaced with "Groupwise Separable" Convolution where factorization rate f is a programmable parameter. We refer to the resulting set of architectures as FactorizeNet. The groupwise separable convolution follows the structure of depthwise separable convolutions [17]. i.e., GroupConv-BatchNorm-Relu-PointwiseConv. When f = *input depth*, we recover depthwise separable convolutions. Following best practices, we always use a Conv-BatchNorm-Relu op-pattern. We demonstrate two progressively increasing factorization methods. The first is a uniform factorization configuration. i.e., A single factorization rate is applied to every Groupwise Separable Conv layer in the network. We progressively double this factorization rate on each step through the search space. We train networks with uniform factorizations of f = *2, 4, 8, 16*. These networks are denoted FactorizeNet-f_j where j is uniform factorization rate (e.g., FactorizeNet-f_2 is the network with a uniform factorization rate of 2). The second approach is to progressively double the factorization rate as we go deeper into the CNN in a Reverse Pyramid configuration (see Figure 5.3 for details). For Reverse Pyramid factorization, we train networks with f_{init} = *2, 4*. These networks are denoted FactorizeNet-f_{initk} where k is initial factorization rate (e.g., FactorizeNet-f_{init2} is the network with reverse pyramid factorization and initial factorization rate of 2). We also train FactorizeNet with regular convolution and depthwise separable convolution in place of Groupwise Separable Conv (denoted Regular_Conv and DWS_Conv). Each network is trained from scratch for 200 epochs of SGD with Momentum = 0.9, batch-size = 128, and Glorot Uniform initializer [11] for all layers. Initial learning rate is 0.01 and we scale it by 0.1 at the 75th, 120th, and 170th epochs. For the activation range tracking we perform top/bottom 1% clipping computed on a random sample of 1024 training samples. Basic data augmentation includes vertical/horizontal shift, zoom, vertical/horizontal flip and rotation. We use Tensorflow for training and quantizing the weights and activations to quint8 format. Basic top/bottom 1% percentile clipping is used for activation quantization as it is a common, low-overhead method.

For each network we observe the efficiency-accuracy trade-offs with respect to 4 quantities: fp32 accuracy, quint8 accuracy, percent relative accuracy decrease, quantized mean-squared error (QMSE), and quantized crossentropy (QCE), similar to those in Chapter 4. Figures 5.4 and 5.5 shows these quantities vs MAC-count.

5.5 Discussion

From Figures 5.4 and 5.5, we have a high-level picture of the efficiency/accuracy trade-offs. Interestingly, FactorizeNet-f_{init2} (104.3 MMACs, 86.01% fp32 acc, 80.31% quint8 acc) has less MACs than FactorizeNet-f_2 (153.8 MMACs, 86.54% fp32 acc, 80.05% quint8

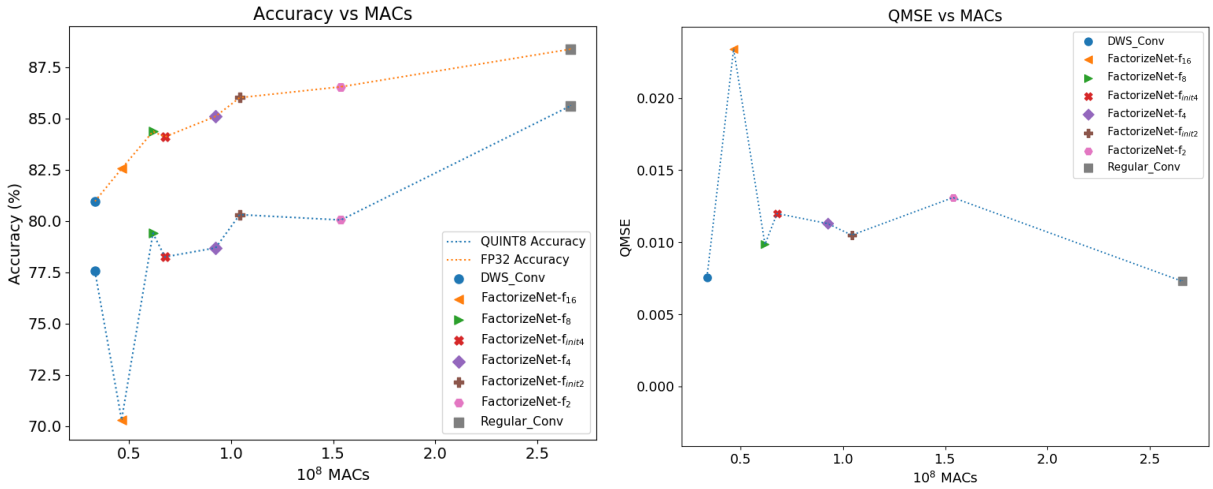

Figure 5.4: Best viewed in colour. **Left**: Accuracy vs MACs (fp32 and quint8 accuracy) under depth factorization. Since the Dense layers are fixed, we only compare the MAC totals of the convolution layers. **Right**: QMSE vs MACs.

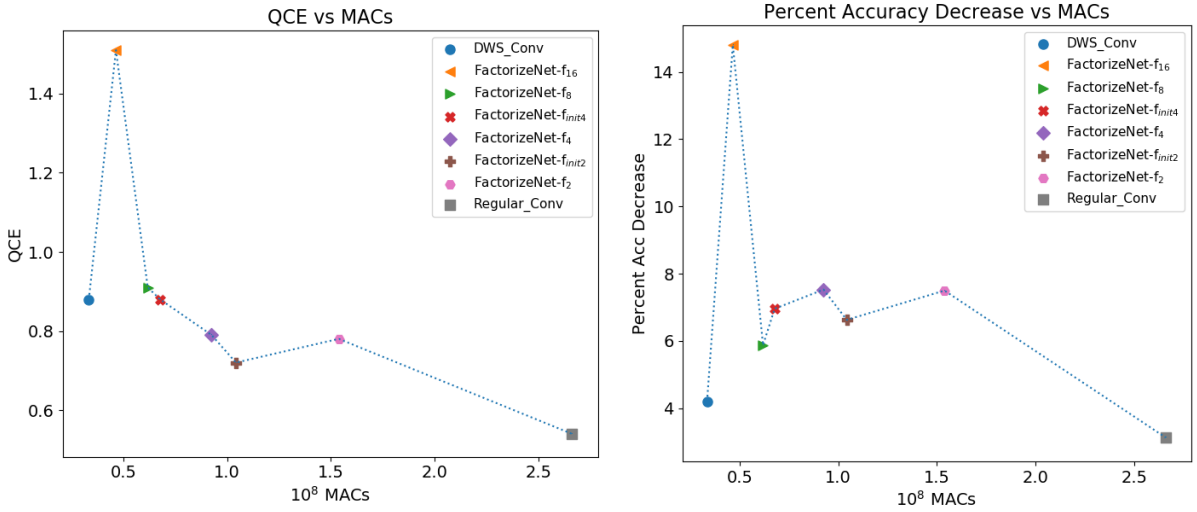

Figure 5.5: Best viewed in colour. **Left**: QCE vs MACs. **Right**: Percent accuracy decrease vs MACs.

acc) but similar accuracy. Furthermore, if targeting fp32 environments, FactorizeNet-f_{init2} would offer over 2.5x MAC reduction from Regular_Conv (266.0 MMACs, 88.37% fp32 acc, 85.60% quint8 acc) with a very small accuracy reduction. When analyzing quantized accuracy, some interesting anomalies emerge. Specifically the sharp drop in accuracy for FactorizeNet-f_{16} (14.8% relative accuracy drop). Also worth noting is that while most of the other models have higher quantized accuracy, DWS_Conv experiences a noticeably smaller *relative decrease* in quantized accuracy (4.21% vs. 5.88% - 7.53%). This may be due to the much smaller increase in range of the BN-Fold weights in its first layer.

To better understand the factors contributing to the degradation in FactorizeNet-f_{16}, we move to our low-level analysis. Figures 5.6, 5.7, 5.8 show the dynamic ranges of each layer. This low-level information gives us a direct look at the underlying distributions and how they interact with quantization noise. For example, besides generally smaller weights ranges (both convolution weights and batchnorm-folded weights), Regular_Conv activations ranges are also noticeably lower. This begins to explain why Regular_Conv is so robust to quantization (3.13% relative accuracy loss). Going back to FactorizeNet-f_{16}, the increased BN-Fold weights ranges early in the network may begin to explain why this CNN experienced a sharp drop in quantized accuracy. Furthermore, if we analyze the average precision of the BN-Fold weights in FactorizeNet-f_{16} we see a combination of large range and low precision in the early, low-level feature extraction layers. Interestingly, the BN-Fold weights in FactorizeNet-f_2 show an even worse average precision in the first layer. However, the precision of BN-Fold weights in FactorizeNet-f_2 is higher on average and hints at a more representative projection of the network's layers from their continuous distribution into a discretized space. Furthermore, we observe a generally lower range of activations for Factorizenet-f_2. See Figures 5.9, 5.11, 5.10, and 5.12 for detailed comparisons. The rest of the plots are also included below.

There would appear to be some rich dynamics involving information loss from both quantized activations and quantization of parameters. Error from quantized activations could be viewed as noise in the "message" being passed onto the next layer whereas error from quantizing the BN-Folded weights is noise introduced in the "encoding" process or inaccuracy of the encoded relationships between inputs (e.g., input pixels). Given that zero can be perfectly encoded in uniform quantization methods as described in [24], it would also be interesting to see if there is a correlation between layerwise sparsity of weights and activations and quantization error. Zooming back out to the inter-network trends, we can see from the BN-Fold weights ranges that there may be a significant loss of information in the early low-level feature extraction stages. It would be interesting to see how these distributions change if we do not use BatchNorm for the first layer since the pre-BN-Fold weights have a much smaller range. From these initial analyses, we see another aspect of

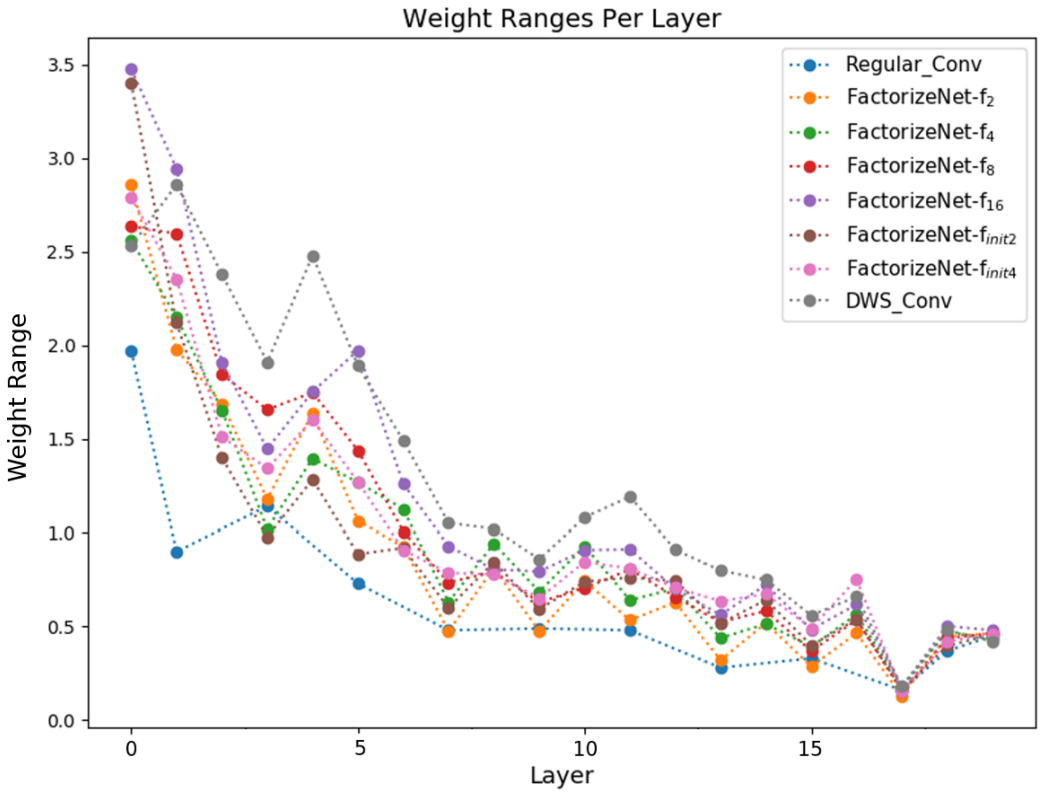

Figure 5.6: Weights ranges per layer. In general, we see how ranges get smaller as we go deeper into the network.

CNN design in which a fine-grained, systematic analysis can yield detailed insights to help further guide our design process.

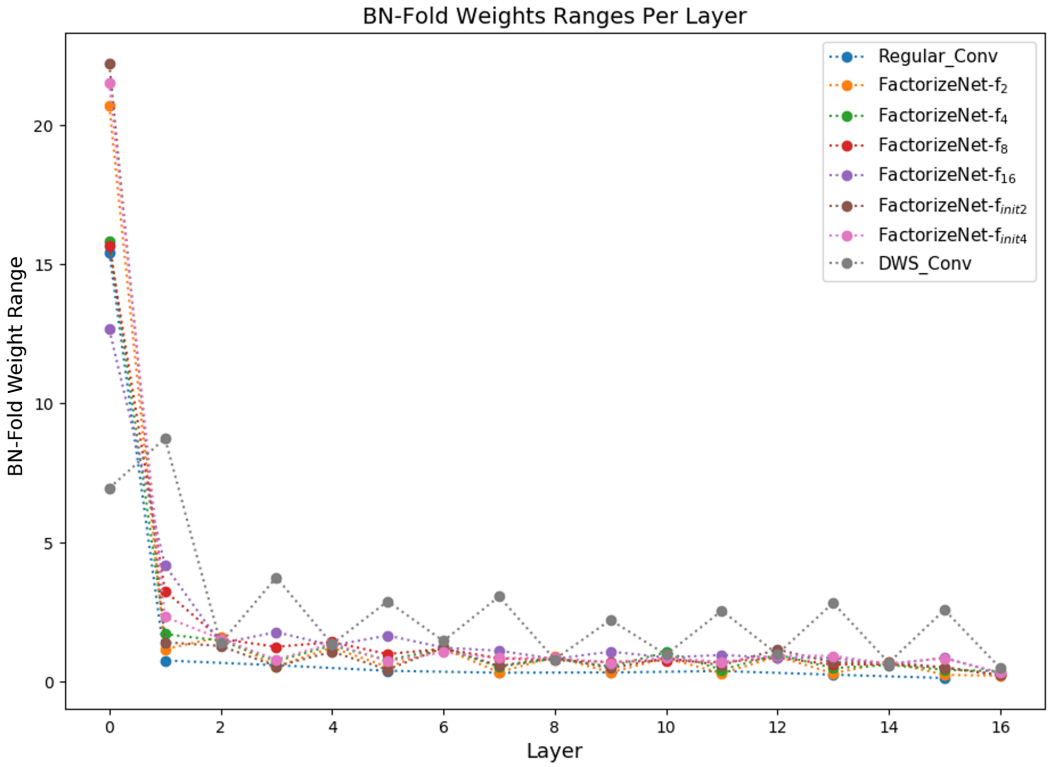

Figure 5.7: BatchNorm-folded weights ranges per layer. Here we can see how the BatchNorm parameters have led to a significant change in the dynamic ranges of each layer. Most notable is the large spike in ranges of the first convolution layer.

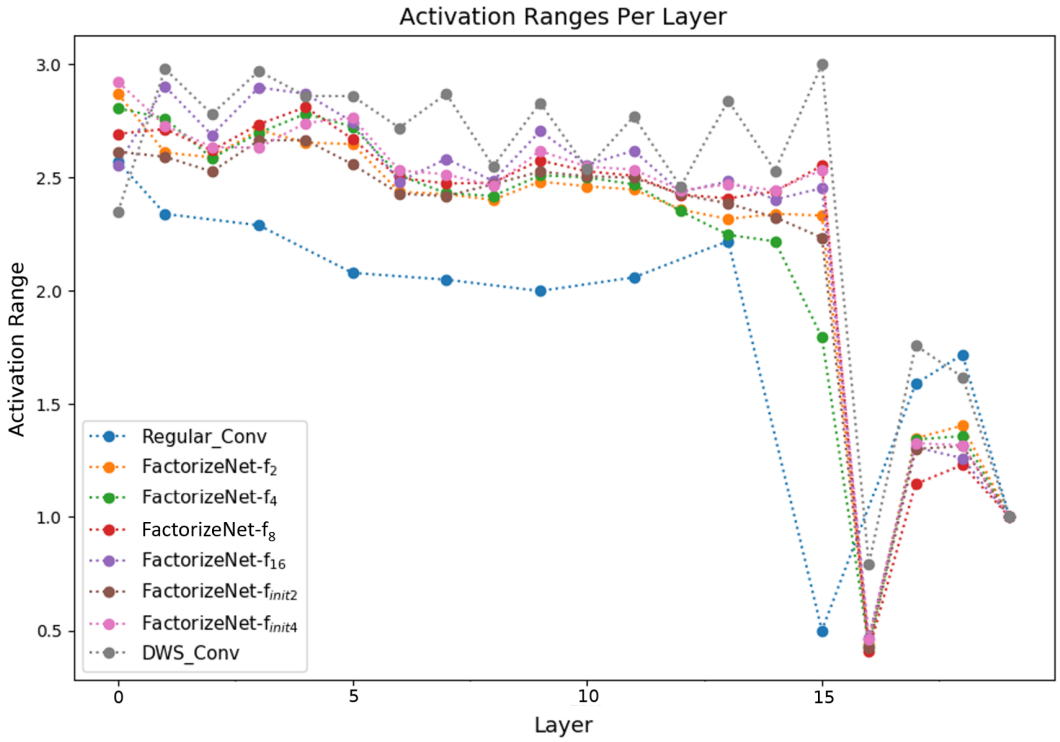

Figure 5.8: Activation ranges per layer. We performed percentile clipping to obtain the activation ranges.

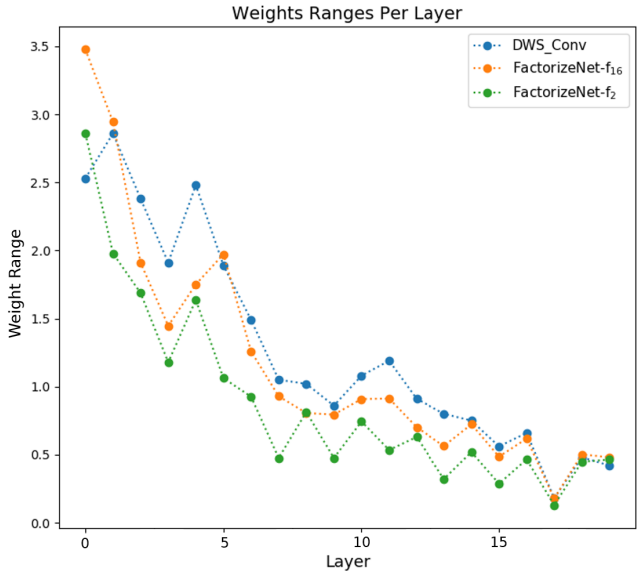

Figure 5.9: Comparing layerwise weights ranges for DWS_Conv, FactorizeNet-f_{16}, and FactorizeNet-f_{f2}.

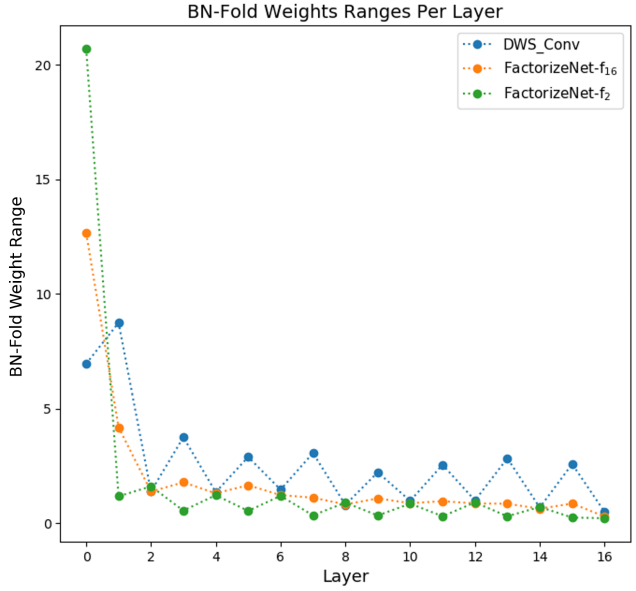

Figure 5.10: Comparing layerwise BatchNorm-folded weights ranges for DWS_Conv, FactorizeNet-f_{16}, and FactorizeNet-f_{f2}.

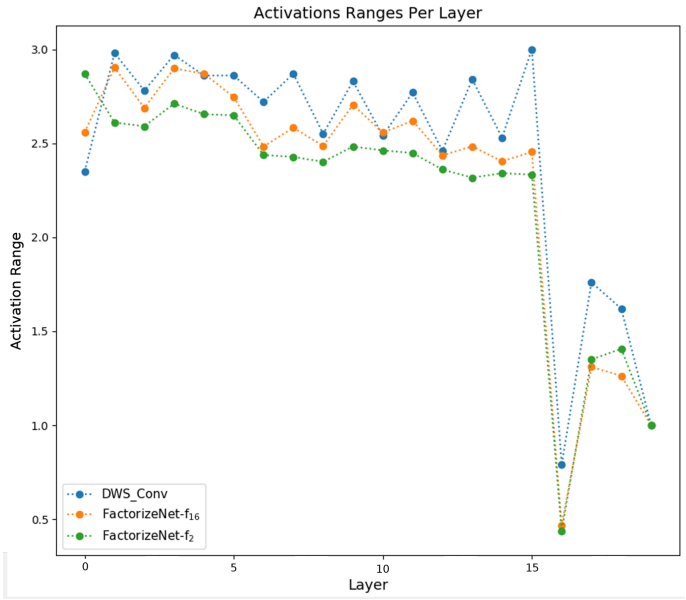

Figure 5.11: Comparing layerwise activation ranges for DWS_Conv, FactorizeNet-f_{16}, and FactorizeNet-f_{f2}.

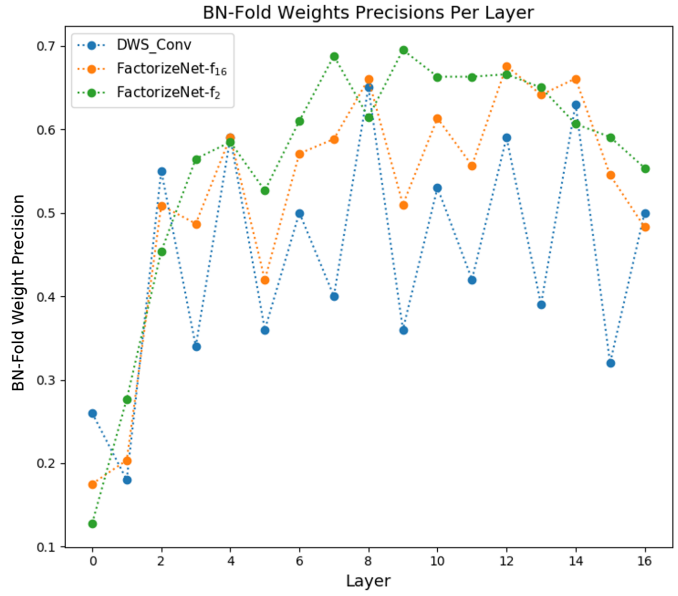

Figure 5.12: Comparing average precision of each layer's BatchNorm-Folded weights for DWS_Conv, FactorizeNet-f_{16}, and FactorizeNet-f_{f2}.

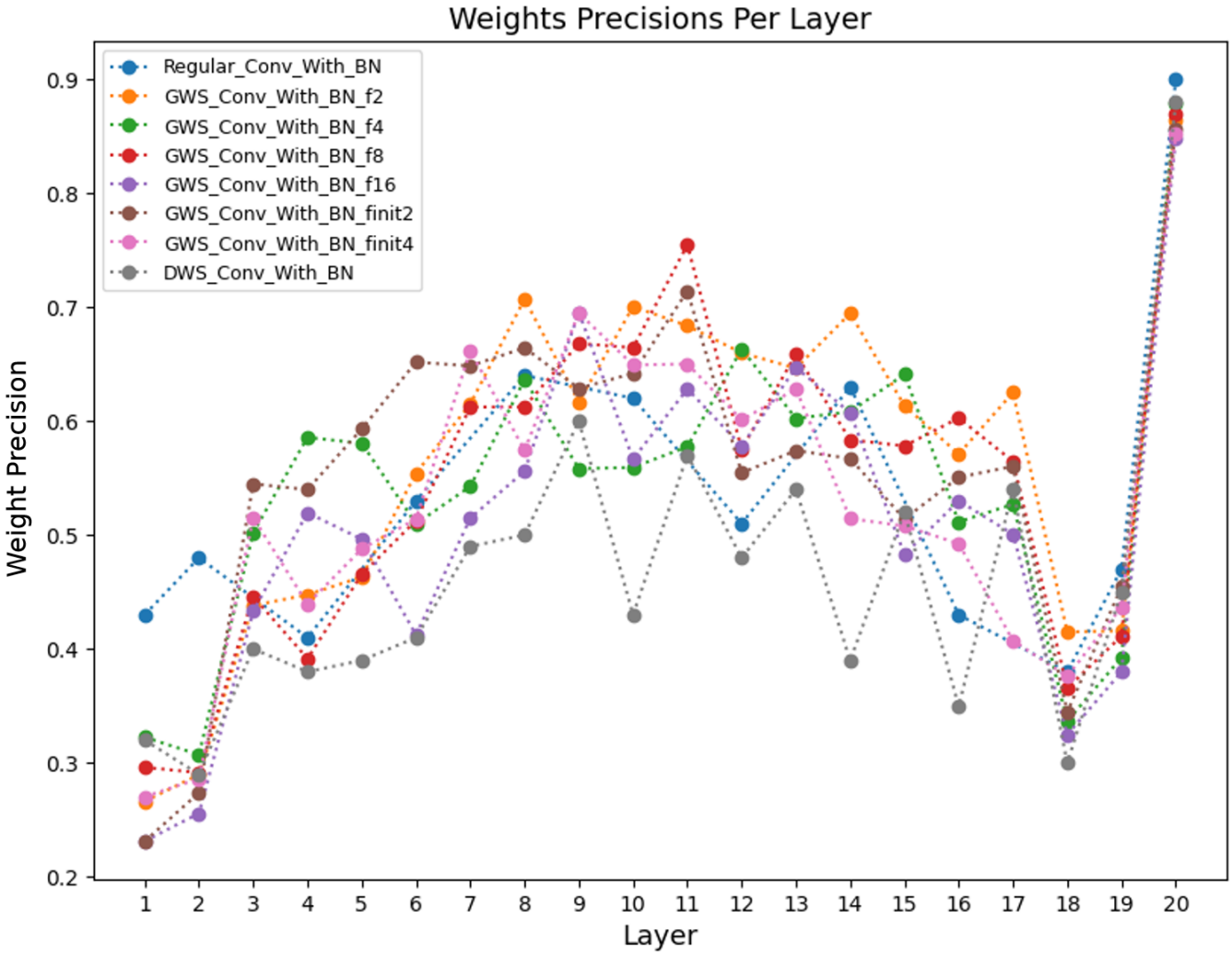

Figure 5.13: Weights precisions per layer.

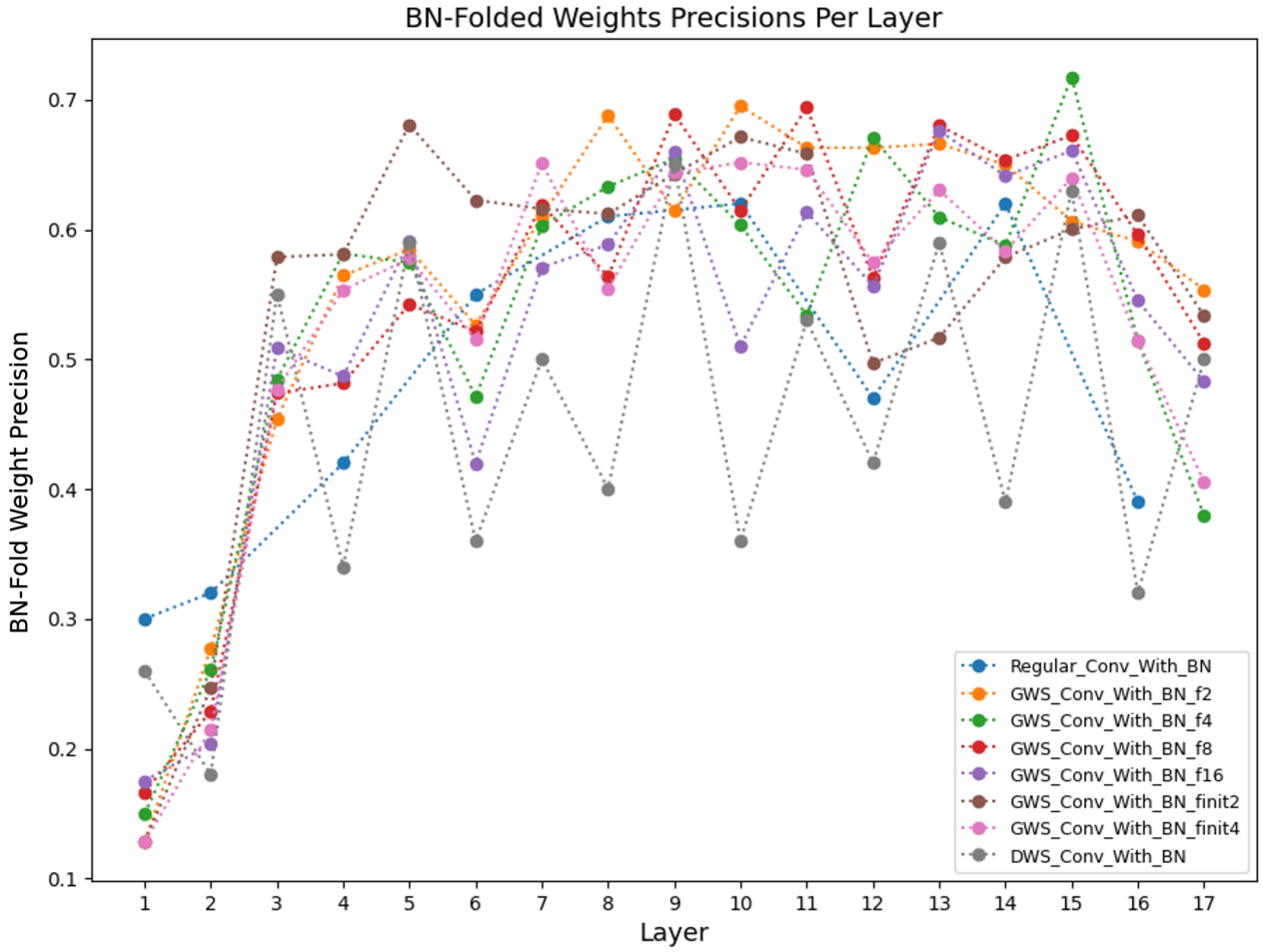

Figure 5.14: BatchNorm-folded weights precisions per layer.

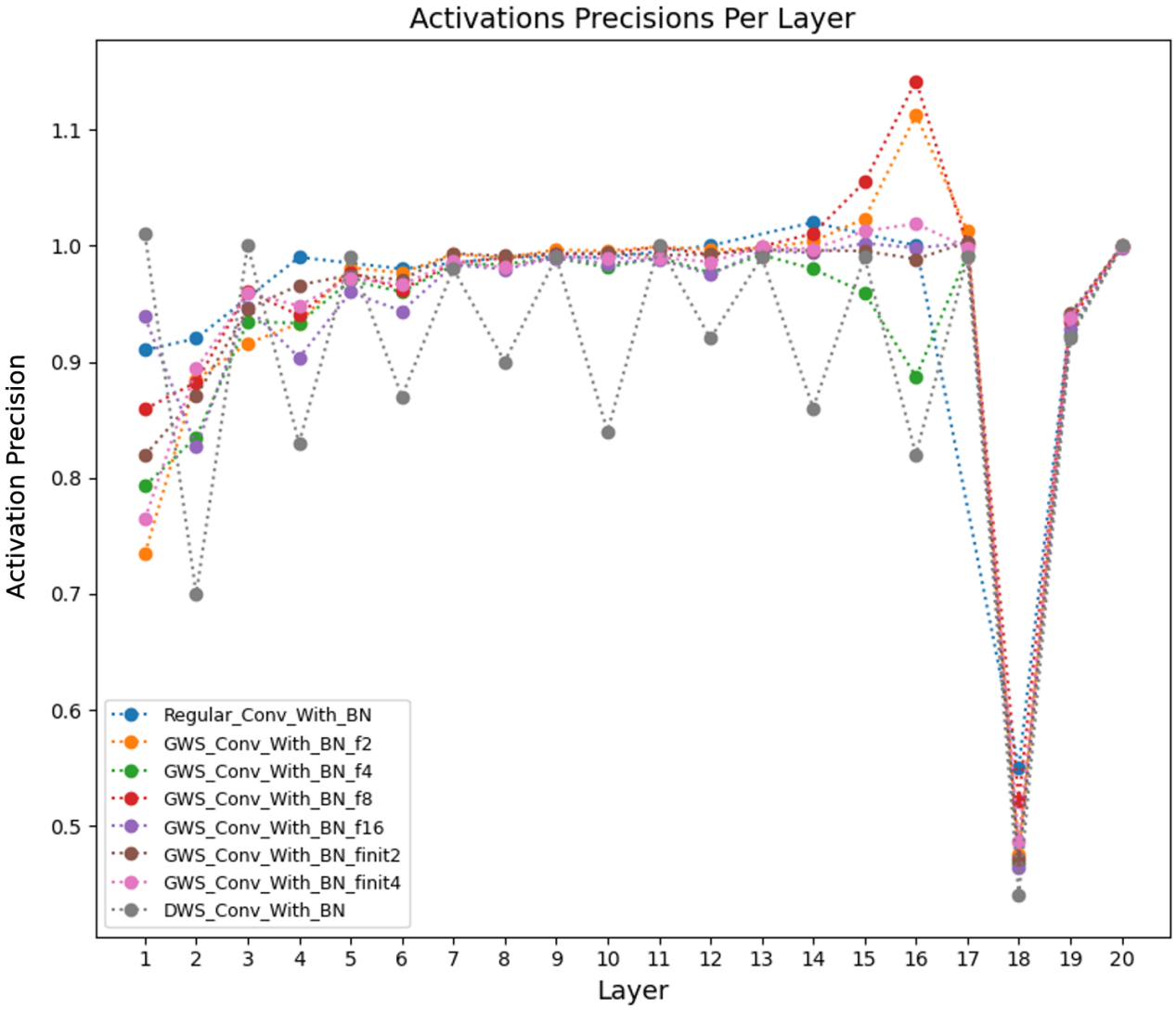

Figure 5.15: Activation precisions per layer. We performed percentile clipping to obtain the activation ranges of the tensor and each channel.

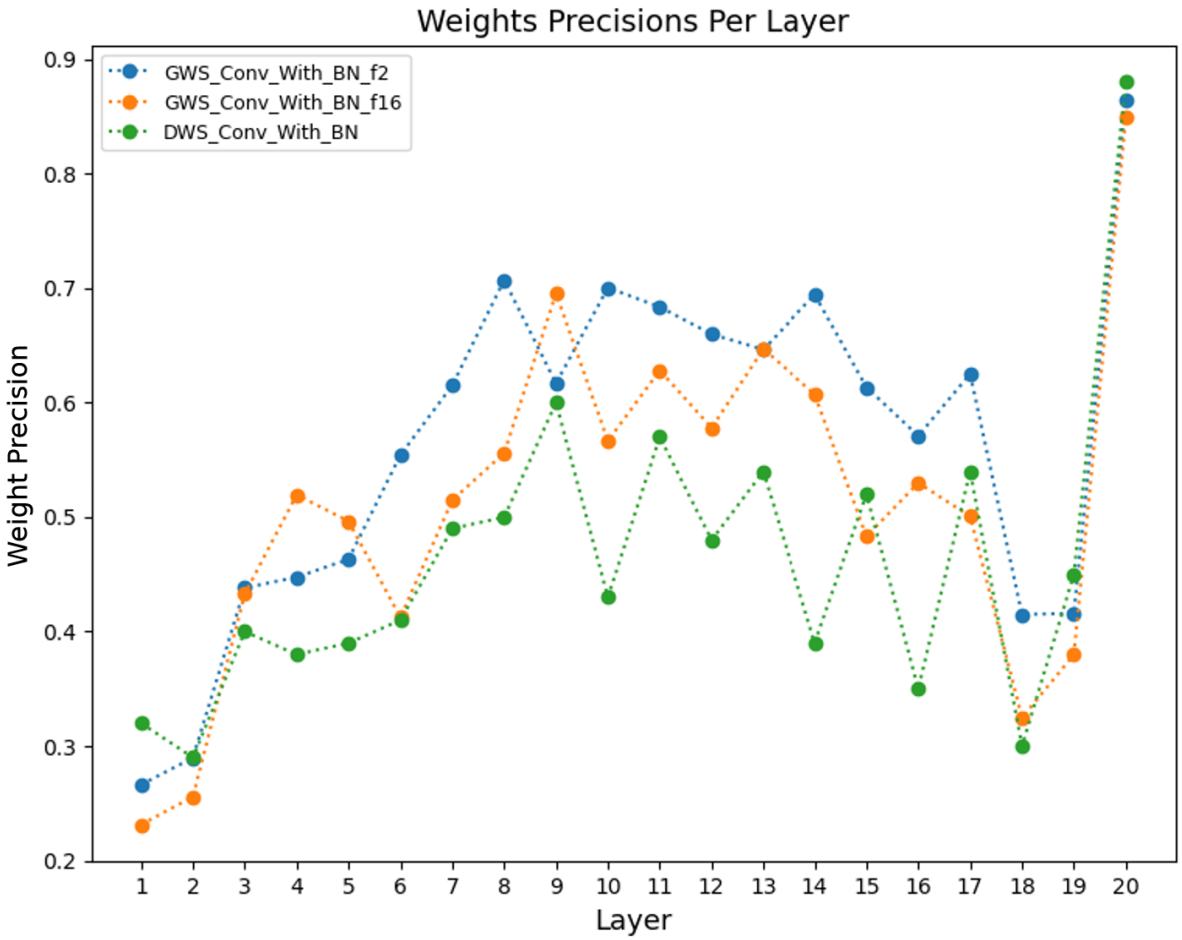

Figure 5.16: Comparing layerwise weights precisions for DWS_Conv, FactorizeNet-f_{16}, and FactorizeNet-f_{f2}.

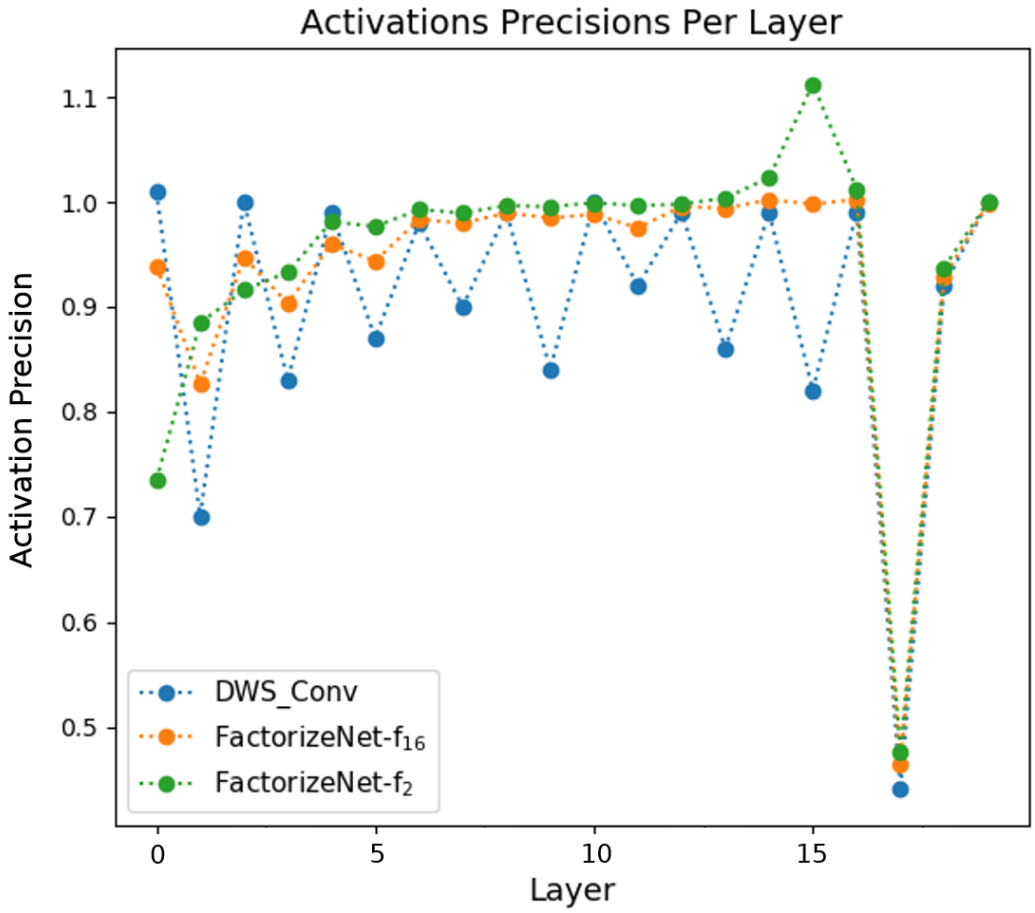

Figure 5.17: Comparing layerwise activation precisions for DWS_Conv, FactorizeNet-f_{16}, and FactorizeNet-f_{f2}.

5.6 Conclusion

We introduce a systematic, progressive depth factorization strategy coupled with a fine-grained layerwise analysis for exploring the efficiency/accuracy trade-offs of factorizing CNN architectures. In this chapter, we demonstrate another way in which our fine-grained layerwise analysis framework can be applied to better understand CNN design. Namely, in understanding the effect of architectural choices on the learned layerwise distributions. In doing so, we can gain detailed insights on the impact of depth factorization on final floating point and quantized accuracy and also identify the optimal factorization configuration (i.e., FactorizeNet). Future work includes using more sophisticated algorithms for increasing factorization, investigating activation sparsity under factorization, and factorizing more complex blocks/architectures.

Chapter 6

Conclusions

6.1 Discussion

This work serves as an initial study on using more systematic, fine-grained approaches to understanding behaviour of our trained CNN models, especially as it pertains to quantization and leveraging that understanding to design better quantized models. Furthermore, we have demonstrated how framing our analysis and design in the context of quantization can create a more streamlined pipeline for designing CNNs for deployment on-the-edge. As we have seen, there is significant room for exploration when it comes to analyzing the behaviour of neural network models at different scales, from the layer-scale up to the model-scale. Such iterative, systematic approaches can help us learn more about the design space and guide our exploration of various hyperparameter and design choices of our NN algorithm.

We have seen how there are rich dynamics at play in each CNN we train and how these layerwise distributions can give rise to complex interactions with quantization noise that require a fine-grained analysis to better understand. For example, something as simple as the choice of random initialization method can lead to unexpected quantized behaviour that does not necessarily align with observed floating point behaviour. From our experiments, we have seen how a fine-grained, systems-based analysis of CNNs can deepen our understanding of observed high-level model behaviour/issues and inspire future works for improving on such observed error. For example, progressively increasing the level of depth factorization *within* a CNN (such as in Reverse Pyramid factorization desribed in Chapter 5) can improve quantization robustness while still yielding a large reduction in

MACs. Other factorization settings should be explored for quantization-robust, efficient CNN architectures.

While our primary focus was on quantization behaviour, we believe that our proposed fine-grain analysis framework can help us understand other aspects of CNN behaviour such as the layerwise dynamics during training, learning dynamics in general, layerwise representations, and layerwise sparsity. Below, we suggest some future directions in which we believe we could be expanding our framework with new analyses and improving on our existing ones. Furthermore, we suggest how insights gained may be leveraged for creating new methods of designing quantization-aware CNNs.

6.2 Future Research

6.2.1 Exploring Additional Layerwise Statistics

We have started with two simple layerwise statistics that are easy to calculate and demonstrate the utility of our approach. However, range and precision are not the only layerwise statistics that we could collect. For example, the metrics we used to quantify the output error of our quantized model such as QCE and QMSE could be used for measuring layerwise quantization noise as well. Such an analysis could reveal how error propagates through the system and also how quantization noise induces distributional shifts in each layer. Minimizing layerwise QMSE might also be a useful method for further improving the quantized accuracy of hard-to-quantize models as it provides a stronger supervisory signal for minimizing quantization error.

Layerwise activation and weight sparsity is also an interesting statistic that could be explored. For example, observing how layerwise sparsity changes under increasing depth factorization might reveal some insights on how depth factorization affects the types of deep representations learned. Furthermore, as zero is often perfectly represented in uniform quantization methods, varying levels of sparsity might also point to another factor affecting a given CNN's sensitivity/robustness to quantization noise. For example, would CNNs with a high degree of sparsity be more robust to quantization noise? Current research focuses on learning highly sparse weights for the sake of reducing number of stored parameters or for structured pruning to reduce computations. However, not much exploration has been done on the relationship between sparsity and quantization robustness. Do sparse activations or sparse weights increase robustness? We believe both are worthy of investigation.

6.2.2 Improving Robustness/Generalizability of our Analyses

One aspect of quantization error that we have not accounted for in these initial studies is the effect of the data used for "calibrating" the network quantization. I.e., the dataset used to determine the quantization parameters of each layer's activations. From papers in the literature (e.g., Choi et al. [4] and Jain et al. [25]) we have seen that the choice of activation ranges used for quantization is highly non-trivial and there are a plethora of ways to determine them. In our simple post-training quantization method we use a randomly sampled set of images from the training set. It would be beneficial in future works to compute the average/expected quantized behaviour of a model across various "quantization trials" where different randomly sampled sets of data are used. Future works should focus on multiple trials of both training and quantization so that we can get a more accurate picture of *expected* behaviour for various models, especially when comparing quantized behaviour across various CNN architectures. Quantifying both the average quantized behaviour and the variation in behaviour might reveal how different models are more/less stable to quantization and other choices.

6.2.3 Analyzing More Complex CNN Architectures

As mentioned before, we should also extend fine-grained layerwise analysis to more complex CNN architectures such as residual blocks. The experiments in this work use a simple VGG-like architecture with straightforward connectivity. However, we have seen in recent years that most state-of-the-art CNNs have various complex connectivity patterns such as ResNet [16], DenseNet [20], and MobileNet-v2 [41]. In this context, we see that a natural generalization of fine-grained layerwise analysis is to also consider "block" or "module"-wise analysis wherein multiple layers/operations can be grouped together into a module. Considering groups of layers adds another level of granularity between the layer-scale and the model-scale and may also be useful for understanding how different parts or "stages" of a network transform/extract information and affect quantization. Such multi-scale analysis further reinforces the approach of analyzing CNN algorithms from a system perspective.

6.2.4 Fine-grained Layerwise Analysis for Understanding Learning Dynamics

As briefly illustrated in Chapter 4, Section 4.6, our fine-grained layerwise analysis framework can also be applied to analyze how a model changes during training. Expansion of our

layerwise analysis could include observing various aspects of how layerwise distributions change over time as well as layerwise gradient information during backpropagation. This could help to understand what is being learned/encoded in our models. In Section 4.6 we saw that we could only exclude the worst models with significant quantization error when predicting quantized accuracy based on early behaviour. This lack of predictability was reflected in the fluctuation of layerwise range and precision for multiple layers that in some cases persisted through most of training. Similarly, perhaps observing some other metric/statistic will yield insights as to how different CNNs progress through training. Such insights could potentially yield better training methods for learning accurate, quantization-robust CNNs.

For example, from our layerwise analyses we have seen that distributions with smaller range and higher average precision tend to be more robust to quantization. Thus, it would make sense to devise methods of training CNNs that promote learning of compact, non-heavy-tailed distributions of weights and activations. It is unclear whether this can be achieved by devising new learning algorithms or if it should be achieved by modifying the cost function that is being optimized. Shkolnik et al. [44] devise a regularizer based on the kurtosis of weights to promote learning distributions of weights that are more uniform. However, their regularizer penalizes kurtosis of the entire weight tensor and we have seen that a good match between channelwise distributions and the tensorwise distribution can be important. A stronger constraint that also regularizes the kurtosis of each channel is a potential direction to explore. Alternatively, perhaps an iterative solver that can better find flat minima/low curvature areas of the loss surface or computes weight updates that lead to smoother distributions would be well suited for learning quantization-robust CNNs. Besides learning algorithms based on iterative optimizers, meta-learning algorithms may also be adapted for quantization robustness. Learning to predict quantization-aware weight updates is an interesting direction of exploration.

6.2.5 Better Quantization-Aware Design of CNNs

Finally, it would be most pertinent to leverage the insights gained from our fine-grained analysis framework for better quantization-aware design of CNNs. For example, we saw in Chapter 4 how the use of BatchNorm would consistently lead to increased quantization error, especially for depthwise-separable CNNs. Thus, better "BatchNorm-free" training of CNNs may be an effective tool for training more quantization-robust CNNs. The authors of [1] have presented such a training method but did not investigate the implications of their method for CNN quantization. In another example, we saw in Chapter 5 that increasing depth factorization of convolutional layers often leads to greater spikes in weights ranges

and worse average precision. The CNNs from those experiments appear to be particularly sensitive to greater depth factorization in earlier layers. Thus, designing CNN architectures with less depth factorization in the early convolutional layers may provide a better trade-off between computational efficiency and quantization robustness. Another related avenue of investigation could be using mixed-precision quantization where layers with greater depth factorization can be allocated a greater number of bits as their computational cost is lower and more bits can be allocated to represent the larger dynamic ranges. This is a simple heuristic that would likely be outperformed by one that also takes into account the statistics of the layerwise distributions.

Referring back to our findings in Chapter 4, revisiting analysis of random weight initialization with quantization in mind could yield more quantization-friendly initialization methods. For example, we saw that the "naive" initialization methods worked surprisingly well for training depthwise-separable CNNs without BatchNorm that had low quantization error. It would be interesting to see if further exploration of this phenomenon could yield a more concrete method for parameterizing the random sampling distributions for depthwise separable CNNs similar to what is done by Glorot et al. [11] and He et al. [15]. Furthermore, current best practices for quantization-aware training with simulated quantization require first training the CNN with FP32 precision until convergence before fine-tuning the model with simulated quantization. For example, the Tensorflow guide to quantization-aware training suggests it is better to finetune with quantization rather than train from scratch [7]. If a better initialization could reduce the amount of FP32 training iterations required prior to switching to quantization simulation it would greatly reduce the training costs of quantized CNNs when using quantization-aware training.

An additional area of design that we did not yet discuss is quantization-friendly activations. In [24], Jacob et al. use ReLU6 (a ReLU activation with the output range clamped to the interval $[0, 6]$) as a straightforward means of limiting the dynamic range of activations. Other choices of the maximum clamping value could be useful and prove better for a given application. However, besides limiting dynamic range, there may be nonlinearities that map the activations into a more evenly distributed space. For example, maybe ReLU followed by a $Log_2(x)$ function would produce more linearly distributed activations. Miyashita et al. [34] propose using a Log_2 based quantization scheme as it better fits the heavy-tailed distributions of weights and activations often found in neural networks. However, perhaps this mapping could be directly performed in FP32 during training. Alternatively, if future research reveals a strong correlation between activation sparsity and quantization robustness, then perhaps activation functions such as ReLU that promote sparsity would be desirable for quantization.

In general, we believe that analyzing trends in layerwise distributions using our pro-

posed fine-grained analysis framework will yield valuable insights for improving the design of CNNs such that the distributions of feature maps and weight tensors are more amenable to quantized inference.

References

[1] Andy Brock, Soham De, Samuel L Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 1059–1071. PMLR, 18–24 Jul 2021. ICML'21 Virtual, 18–24 July, 2021.

[2] Neil Burgess, Jelena Milanovic, Nigel Stephens, Konstantinos Monachopoulos, and David Mansell. Bfloat16 processing for neural networks. In *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, pages 88–91, 2019. In Kyoto, Japan, 10–12 June 2019.

[3] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 367–379, 2016. In Seoul, Korea, 18-22 June, 2016.

[4] Jungwook Choi, Swagath Venkataramani, Vijayalakshmi (Viji) Srinivasan, Kailash Gopalakrishnan, Zhuo Wang, and Pierce Chuang. Accurate and efficient 2-bit quantized neural networks. In A. Talwalkar, V. Smith, and M. Zaharia, editors, *Proceedings of Machine Learning and Systems*, volume 1, pages 348–359, 2019. In Stanford, California, USA March 29–April 2, 2019.

[5] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 1251–1258, 2017. In Honolulu, USA, 21–16 July, 2017.

[6] Intel Corporation. bfloat16 - hardware numerics definition intel whitepaper. Available at https://www.intel.com/content/dam/develop/external/us/en/documents/bf16-hardware-numerics-definition-white-paper.pdf.

[7] Tensorflow Documentation. Quantization aware training comprehensive guide: tensorflow model optimization, Jan 2022. Accessed on 30 March, 2022 at https://www.tensorflow.org/model_optimization/guide/quantization/training_comprehensive_guide.

[8] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2758–2766. In Santiago, Chile, 7–13, December 2015.

[9] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[10] Ross Girshick. Fast R-CNN. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15 in Santiago, Chile, 7–13 December 2015, page 1440–1448, USA, 2015. IEEE Computer Society.

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[12] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir D. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv*, abs/1412.6115, 2014.

[13] Boris Hanin and David Rolnick. How to start training: The effect of initialization and architecture. *Advances in Neural Information Processing Systems*, 2018-December:571–581, 2018. Publisher Copyright: © 2018 Curran Associates Inc..All rights reserved. Copyright: Copyright 2019 Elsevier B.V., All rights reserved.; 32nd Conference on Neural Information Processing Systems, NeurIPS 2018, Montreal, Canada ; Conference date: 02-12-2018 Through 08-12-2018.

[14] K. He, G. Gkioxari, P. Dollar, and R. Girshick. Mask R-CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, Los Alamitos, CA, USA, oct 2017. IEEE Computer Society. In Venice, Italy, 22–29, October 2017.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference*

on Computer Vision (ICCV), ICCV '15 in Santiago, Chile, 7–13 December 2015, pages 1026–1034.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. In Las Vegas, USA, June 26–July 1, 2016.

[17] 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*, abs/1704.04861, 2017.

[18] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. In Salt Lake City, USA, 18–22 June, 2018.

[19] Gao Huang, Shichen Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. In Salt Lake City, USA, 18–22 June, 2018.

[20] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017. In Honolulu, USA, 21–26 July 2017.

[21] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16 in Barcelona, Spain, 5–10 December 2016, page 4114–4122, Red Hook, NY, USA, 2016. Curran Associates Inc.

[22] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *arXiv*, abs/1602.07360, 2016.

[23] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15 in Lille, France, 6–11 July, 2015, page 448–456. JMLR.org, 2015.

[24] 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 (CVPR)*, June 2018. In Salt Lake City, USA, 18–22 June, 2018.

[25] Sambhav Jain, Albert Gural, Michael Wu, and Chris Dick. Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks. In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, volume 2, pages 112–128, 2020. In Austin, USA, 2–4 March 2020.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. NIPS'12 in Lake Tahoe, USA, 3–8 December 2012.

[27] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[28] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 105–114, 2017. In Honolulu, USA, 21–16 July, 2017.

[29] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2849–2858, New York, New York, USA, 20–22 Jun 2016. PMLR.

[30] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *ICLR 2019*, April 2019. In New Orleans, USA, 6–9 May 2019.

[31] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 21–37, Cham, 2016. Springer International Publishing. In Amsterdam, The Netherlands, 8–16 October 2016.

[32] Eldad Meller, Alexander Finkelstein, Uri Almog, and Mark Grobman. Same, same but different: Recovering neural network quantization error through weight factorization. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4486–4495. PMLR, 2019. In Long Beach, USA, 09–15 June 2019.

[33] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *International Conference on Learning Representations*, 2018. In Vancouver, Canada, April 30–May 3 2018.

[34] Daisuke Miyashita, Edward H. Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *arXiv*, abs/1603.01025, 2016.

[35] M. Nagel, M. Baalen, T. Blankevoort, and M. Welling. Data-free quantization through weight equalization and bias correction. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, ICCV'19 in Seoul, Korea, Oct. 27–Nov. 2, 2019, pages 1325–1334, Los Alamitos, CA, USA, nov 2019. IEEE Computer Society.

[36] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In Johannes Fürnkranz and Thorsten Joachims, editors, *ICML*, ICML'10 in Haifa, Israel, 21–24 June 2010, pages 807–814. Omnipress, 2010.

[37] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. in Las Vegas, USA, June 26–July 1 2016.

[38] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing. In Munich, Germany, 5–9 October 2015.

[39] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning Representations by Back-Propagating Errors*, page 696–699. MIT Press, Cambridge, MA, USA, 1988.

[40] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C.

Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[41] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. In Salt Lake City, USA, 18–22 June, 2018.

[42] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823, 2015. In Boston, USA, 7–12, June 2015.

[43] T. Sheng, C. Feng, S. Zhuo, X. Zhang, L. Shen, and M. 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, 2018. In Williamsburg USA, March 25, 2018 in conjunction with ASPLOS 2018.

[44] Moran Shkolnik, Brian Chmiel, Ron Banner, Gil Shomron, Yury Nahshan, Alex Bronstein, and Uri Weiser. Robust quantization: One model to rule them all. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33 of *NeurIPS'20 Virtual, 6–12 December 2020*, pages 5308–5317. Curran Associates, Inc., 2020.

[45] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[46] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. PWC-Net: Cnns for optical flow using pyramid, warping, and cost volume. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8934–8943, 2018. In Salt Lake City, USA, 18–22 June, 2018.

[47] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17 in San Francisco, USA, 4–9 February 2017, page 4278–4284. AAAI Press, 2017.

[48] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. Long Beach, USA, 16–20, June 2019.

[49] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019. In Long Beach, USA.

[50] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18 in Montréal, Canada, 3–8, December 2018, page 7686–7695, Red Hook, NY, USA, 2018. Curran Associates Inc.

[51] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. ESRGAN: Enhanced super-resolution generative adversarial networks. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. In Munich, Germany, 8–14 September 2018.

[52] A. Wong, M. Javad Shafiee, B. Chwyl, and F. Li. Gensynth: a generative synthesis approach to learning generative machines for generate efficient neural networks. *Electronics Letters*, 55(18):986–989, 2019.

[53] B. Wu, A. Wan, F. Iandola, P. H. Jin, and K. Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 446–454, 2017. In Honolulu, USA, 21–26 July 2017.

[54] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10726–10734, 2019. In Long Beach, USA, 16–20 June 2019.

[55] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *2017 IEEE Conference on Com-*

puter Vision and Pattern Recognition (CVPR), pages 5987–5995, 2017. In Honolulu, USA, 21–26 July 2017.

[56] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 586–595, 2018. In Salt Lake City, USA, 18–22 June, 2018.

[57] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6848–6856, 2018. In Salt Lake City, USA, 18–22 June, 2018.

[58] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8697–8710, 2018. In Salt Lake City, USA, 18–22 June, 2018.