Designing Deep Neural Networks for Efficient and Robust Inference

Chaim Baskin
Designing Deep Neural Networks for Efficient and Robust Inference

Research Thesis

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Chaim Baskin

Submitted to the Senate of the Technion — Israel Institute of Technology
Nisan 5781 Haifa March 2021
This research was carried out under the supervision of Prof. Alex Bronstein and Prof. Avi Mendelson, in the department of Computer Science.

Some results in this thesis have been published as articles by the author and research collaborators in conferences and journals during the course of the author’s doctoral research period, the most up-to-date versions of which being:

<table>
<thead>
<tr>
<th>Title</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature map transform coding for energy-efficient cnn inference</td>
<td>In 2020 International Joint Conference on Neural Networks (IJCNN), pages 1–9. IEEE, 2020.</td>
</tr>
<tr>
<td>Early-stage neural network hardware performance analysis</td>
<td>Sustainability, 13(2), 2021.</td>
</tr>
</tbody>
</table>

Acknowledgements

The author’s personal acknowledgements.

The generous financial help of the Technion is gratefully acknowledged.
# Contents

List of Figures

Abstract

Abbreviations and Notations

1 Introduction

1.1 Contribution

2 UNIQ: Uniform Noise Injection for Non-Uniform Quantization of Neural Networks

2.1 Motivation

2.2 Introduction

2.2.1 Neural Networks on custom hardware

2.3 Non-uniform quantization by uniform noise injection

2.3.1 Quantization

2.3.2 Training quantized neural networks by uniform noise injection

2.3.3 Gradual quantization

2.3.4 Activations quantization

2.4 Experimental evaluation

2.4.1 Performance of quantized networks on ImageNet

2.4.2 Accuracy vs. complexity trade-off

2.4.3 Ablation study

2.5 Hardware Implementation

2.5.1 MAC operation for non-uniformly quantized data

2.5.2 Logic utilization and power consumption on FPGA

2.6 Discussion

3 Loss Aware Post-training Quantization

3.1 Motivation

3.2 Introduction

3.3 Loss landscape of quantized DNNs

3.3.1 Separable optimization
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.2</td>
<td>Curvature</td>
<td>32</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Hessian of the loss function</td>
<td>34</td>
</tr>
<tr>
<td>3.4</td>
<td>Loss Aware Post-training Quantization</td>
<td>35</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Layer-wise optimization</td>
<td>35</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Quadratic approximation</td>
<td>36</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Joint optimization</td>
<td>37</td>
</tr>
<tr>
<td>3.5</td>
<td>Experimental Results</td>
<td>39</td>
</tr>
<tr>
<td>3.5.1</td>
<td>ImageNet</td>
<td>39</td>
</tr>
<tr>
<td>3.5.2</td>
<td>NCF-1B</td>
<td>39</td>
</tr>
<tr>
<td>3.5.3</td>
<td>Ablation study</td>
<td>40</td>
</tr>
<tr>
<td>3.6</td>
<td>Discussion</td>
<td>45</td>
</tr>
<tr>
<td>4</td>
<td>Towards learning of filter-level heterogeneous compression of CNNs</td>
<td>47</td>
</tr>
<tr>
<td>4.1</td>
<td>Motivation</td>
<td>47</td>
</tr>
<tr>
<td>4.2</td>
<td>Introduction</td>
<td>48</td>
</tr>
<tr>
<td>4.3</td>
<td>Method</td>
<td>49</td>
</tr>
<tr>
<td>4.4</td>
<td>Quantized NAS</td>
<td>51</td>
</tr>
<tr>
<td>4.5</td>
<td>Slimmable NAS</td>
<td>52</td>
</tr>
<tr>
<td>4.6</td>
<td>Discussion</td>
<td>54</td>
</tr>
<tr>
<td>5</td>
<td>Feature Map Transform Coding for Energy-Efficient CNN Inference</td>
<td>55</td>
</tr>
<tr>
<td>5.1</td>
<td>Motivation</td>
<td>55</td>
</tr>
<tr>
<td>5.2</td>
<td>Introduction</td>
<td>56</td>
</tr>
<tr>
<td>5.3</td>
<td>Transform-Domain Compression</td>
<td>57</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Implementation</td>
<td>60</td>
</tr>
<tr>
<td>5.4</td>
<td>Experimental Results</td>
<td>63</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Eigenvalues analysis</td>
<td>65</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Block shape and size</td>
<td>65</td>
</tr>
<tr>
<td>5.5</td>
<td>Hardware Implementation</td>
<td>67</td>
</tr>
<tr>
<td>5.6</td>
<td>Discussion</td>
<td>69</td>
</tr>
<tr>
<td>6</td>
<td>CAT: Compression-Aware Training for Bandwidth Reduction</td>
<td>71</td>
</tr>
<tr>
<td>6.1</td>
<td>Motivation</td>
<td>71</td>
</tr>
<tr>
<td>6.2</td>
<td>Introduction</td>
<td>72</td>
</tr>
<tr>
<td>6.3</td>
<td>Method</td>
<td>74</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Entropy encoding and rate regularization</td>
<td>74</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Differentiable entropy-reducing loss</td>
<td>75</td>
</tr>
<tr>
<td>6.4</td>
<td>Experimental Results</td>
<td>77</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Ablation study</td>
<td>78</td>
</tr>
<tr>
<td>6.5</td>
<td>Discussion</td>
<td>82</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Classification accuracy on CIFAR-10 of the ResNet-18 architecture quantized using UNIQ with different number of quantization stages during training. .................................................. 20

2.2 **Performance vs. complexity of different quantized neural networks.** Performance is measured as top-1 accuracy on ImageNet; complexity is estimated in number of bit operations. Network architectures are denoted by different marker shapes; quantization methods are marked in different colors. Multiple instances of the same method (color) and architecture (shape) represent different level of quantization. Note that our method performs better for ResNet-50, comparable for ResNet-34, and worse for ResNet-18, this is due to the fact that in Apprentice [MM18] the baselines (full-precision) models are better for smaller models. Our model also performs well for the smaller more efficient MobileNet architecture which is known to be an harder problem. The figure should be viewed in color. .................................................. 20

2.3 k-quantile quantization to discrete integer levels ............................................ 23

2.4 **Logic utilization and power consumption vs. number of bit operations.** A significant correlation between proposed BOPs metric and both logic utilization and power consumption supports the hypothesis that BOPs is a good proxy to both. Note that LUTs here refer to the basic building block of FPGA rather than our implementation of multiplication. .... 24
3.1 A visualization of the loss surface for the pair of layers using one batch of 512 images (ResNet18 on ImageNet). X and Y-axis are quantization ranges of those layers where we estimate the cross-entropy loss (z-axis). The colored dots mark the quantization range found by optimizing different p-norm metrics. Each layer quantized in a way that minimizes the p-norm distance between the quantized tensor to its original full precision counterpart (e.g., MSE distortion). (a) One can see high coupling between two quantization ranges, making layer-wise optimization sub-optimal. (b) A zoom-in of sub-figure (a) to the point where the optimal cross-entropy loss (red cross) clearly distinguished from the optimized p-norm solutions.

3.2 Visualization of the loss surface as a function of quantization ranges of two subsequent layers of ResNet18. At higher bit-width, with more fine-grained quantization Fig. 3.2c, the interactions between layers are relatively small, making layer-wise optimization possible. On the other hand, at lower bit-width Fig. 3.2a, with an increase of the quantization noise, the interactions between two layers become tangible. Due to the high coupling, the per layer (per axis) optimization results will entirely depend on the initial point and potentially sub-optimal.

3.3 Accuracy of ResNet-50 quantized to 2 and 4 bits, respectively. Quantization steps are chosen such that they minimize the $L_p$ norm of the quantization error for different values of $p$.

3.4 The $L_p$ norm of the quantization error generated by optimizing for different values of $p$ according to Eq. (3.15). For different $L_p$ metrics, the optimal quantization step is different.

3.5 Absolute value of the Hessian matrix of the loss function with respect to quantization steps calculated over 15 layers of ResNet-18. Higher values at a diagonal of the Hessian at 2-bit quantization suggest that the minimum is sharper than at 4 bits. Non-diagonal elements provide an indication of the coupling between parameters of different layers: closer layers generally exhibit stronger interactions.

3.6 Loss of ResNet-18 quantized with different quantization steps. The orange line shows quadratic interpolation: (a) with respect to the distance from optimal quantization step $\Delta^*$ and (b) on the trajectory defined by $L_p$ norm minimization.
3.7 Intuition about the LAPQ algorithm in two dimensions. The visualization shows the loss as a function of quantization ranges of two layers on synthetic data. Yellow dots correspond to the quantization step size \( \{\Delta_p\} \), which minimizes the \( L_p \) norm of the quantization error. \( \{\Delta_p\} \) build a trajectory in the vicinity of the minimum \( \Delta^* \). The optimal quantization step size on that trajectory \( \Delta_p^* \) is used as a starting point for a joint optimization algorithm (Powell’s). Vectors \( \{d_1, d_2, d_3\} \) demonstrate the first iteration of Powell’s method that approaches the global minimum \( \Delta^* \).

3.8 Accuracy of ResNet-18 for different sized calibration sets at various quantization levels.

4.1 Results of grid search in both cases. Blue line connects homogeneous configurations, colored points are heterogeneous configurations. Error bars are for 0.6827 confidence interval.

4.2 Results of search in both cases. Each point represents a configuration proposed by search.

5.1 High-level flow diagram of the encoder-decoder chain. PCA and BN are folded into the convolution weights (denoted by *), resulting in a single convolution (boxed in grey).

5.2 Vector quantization in 2D case. (a) A pair of correlated channels on a scatter plot. All values in a cell are mapped to the center of the cell; hence, small cells induce small quantization noise. Several bins are empty (red cells); (b) Decorrelation improves utilization since the cells are smaller now; (c) Forcing equal bin size along all dimensions further improves utilization. Instead of restricting both channel dynamic range to be divided into same number of bins, we use uniform bin size along all dimensions. VLC allows to further compress the representation since the channels with smaller dynamic range have are mapped mostly to a few most probable bins.

5.3 Rate-distortion curves for ResNet-18, ResNet-50, ResNet-101, Inception V3 and MobileNetV2 architectures with 8- (blue) and 4-bit (red) weight quantization. Distortion is evaluated in terms of top-1 accuracy on ImageNet. Dashed lines represent rates obtained by Huffman VLC, while solid lines represent theoretical rates (entropy).
5.4 Ablation study of the proposed encoder on ResNet-18. Left: rate-distortion curve with different encoder configurations. Theoretical rates are reported; top-1 accuracy is used as the distortion measure. Right: theoretical memory rate in bits per value achieved for different levels of PCA truncation for baseline and 0.5% lower than baseline top-1 accuracy.

5.5 Analysis of the eigenvalues ratio that are needed to achieve energy ratio, means cumulative sum of the eigenvalues.

5.6 The influence of the block shape on the top-1 validation accuracy (left) and MSE (right) of ResNet-18 on ImageNet. Our experiments show that for the same size, the most efficient shape is $1 \times 1 \times C$, taking advantage of the correlations across different feature maps at the same spatial location. Theoretical rates are shown.

5.7 Top-1 accuracy on ResNet18 ImageNet vs. power consumption of our hardware implementation. Each point represents a different quantization rate.

6.1 Pre-activation distributions of one layer in ResNet-18. (a) Evolution at different epochs. As training progresses, the probability of non-positive pre-activation values increases, zeroing more post-ReLU values. The sharp peak at zero reduces entropy and thus improves compressibility. (b) Effect of entropy regularization. Without regularization, the distribution has much heavier tails and thus has higher entropy. As regularization increases, the probability of extreme values is significantly reduced. The entropy penalty $\lambda$ was selected so that the overall accuracy is not affected. The compression ratio in the strongly regularized case is 2.23 times higher compared to the unregularized baseline.

6.2 Comparison with other methods: EPBC ([CRB19]) and GF ([GHR18]) in (a) ResNet-34 and (b) MobilenetV2. Different marker sizes refer to different activation bitwidths before compression. For GF, the compression rate was averaged only over compressed layers.

6.3 Tradeoff between rate and accuracy for (a) ResNet-18 and (b) ResNet50 in weight compression. The activations are quantized to 8 bits for fair comparison. “No fine-tuning” refers to pre-trained model with activations and weights quantized to 8 bits and the Huffman coding applied to the weights.

6.4 Tradeoff between rate and accuracy for different values of $\lambda$ (ranged between 0 and 0.3) in (a) ResNet-18 and (b) MobileNetV2. In ResNet-18, the activations are quantized to 5 bits; in MobileNet we show results for activation quantized to 6 and 8 bits.
6.5 Soft entropy with different sample sizes compared to a real entropy of a single batch. The shaded region covers the standard deviation over three runs.

7.1 Area ($A$) vs. bitwidth ($b$) for a $3 \times 3$ PE with a single input and output channel. All weights and activations use the same bitwidth and the accumulator width is 4 bit larger, which is enough to store the result. The quadratic fit is $A = 12.39b^2 + 86.07b - 14.02$ with goodness of fit $R^2 = 0.999977$.

7.2 Area ($A$) vs. BOPS ($B$) for a $3 \times 3$ PE with a single input and output channel and variable bitwidth. The linear fit is $A = 1.694B + 153.46$ with goodness of fit $R^2 = 0.9989$.

7.3 Area ($A$) vs. BOPS ($B$) for a $3 \times 3$ PE with variable input ($n$) and output ($m$) feature dimensions, and variable bitwidth. Weights and activations use the same bitwidth and the accumulator width is set to $\log_2(9m) \cdot b_w \cdot b_a$.

7.4 single-port RAM area vs. data-bits

7.5 dual-port RAM area vs. data-bits

7.6 Per-layer memory access pattern

7.7 Roofline example. In the case of App1, memory bandwidth prevents the program from achieving its expected performance. In the case of App2, the same happens due to limited computational resources. Finally, App3 represents a program that could achieve its maximum performance on a given system.

7.8 OPS roofline: $3 \times 3$ kernel, 256 input and output $14 \times 14$ features, 1mm$^2$ accelerator with 800MHz frequency, with DDR of 2.4GHz with 64 bit data bus.

7.9 OPS roofline: $3 \times 3$ kernel, 64 input and output $56 \times 56$ features, 6mm$^2$ accelerator with 100MHz frequency, with DDR of 2.4GHz with 64 bit data bus.

7.10 All-to-all topology with $n \times m$ processing elements.

7.11 Systolic array of processing elements

7.12 Area ($A$) vs. BOPS ($B$) for a systolic array of $3 \times 3$ PEs with variable input ($n$) and output ($m$) feature dimensions, and variable bitwidth. Weights and activations use the same bitwidth and the accumulator width is set to $\log_2(9m) \cdot b_w \cdot b_a$.

7.13 ResNet-18 roofline analysis for all layers. Red dots are the performance required by the layer, and green dots are the equivalent performance using partial-sum computation. The blue curves connect points corresponding to the same layer and are displayed only for convenience.

7.14 VGG-15 on Eyeriss [Che16] Hardware
7.15 Comparison of BOPS and “compute cost” [MM18] predictive power. BOPS – 5% error, “compute cost” – 15%. 101

8.1 Accuracy on CIFAR-10 under PGD attack as a function of $\epsilon$. The smooth CNI model is WideResNet-28-4, and the model for MART is WideResNet-34-10. The smoothing uses $M = 64$ and $\sigma = 0.25$. Both models were trained using PGD attacks with $k = 10$ and $\epsilon = 8/255$. 115

8.2 Accuracy on CIFAR-10 as a function of injected noise strength on clean data (left) and under PGD attack (right). The model is ResNet-20 defended with CNI. The smoothing uses $M = 2^n$, $n \in \{0, \ldots, 9\}$. Shaded area is a standard deviation of the results over three runs. 117

8.3 Accuracy on CIFAR-10 under PGD, EPGD and SmoothAdv$_{\text{PGD}}$ attacks as a function of number of attack steps, $k \cdot M$. The model is WideResNet-28-4 defended with CNI. The smoothing uses $M = 1, 8$ and $\sigma = 0.24$. EPGD and SmoothAdv$_{\text{PGD}}$ attacks averaged the gradients over $M_{\text{backward}} = 8$ samples in each attack iteration. All the attacks run with 5 random restarts. Shaded area is a standard deviation of the results over three runs. 118

8.4 Accuracy on CIFAR-10 as a function of injected noise strength on clean data (left) and under PGD attack (right). The model is ResNet-20 defended with CNI and fine-tuned on PGD with $k = 56$. The smoothing uses $M = 2^n$, $n \in \{0, \ldots, 9\}$. Shaded area is a standard deviation of the results over three runs. 119

8.5 Accuracy on CIFAR-10 as a function of injected noise strength on clean data (left) and under PGD attack (right). The model is ResNet-20 defended with CNI, with different smoothing and fine-tuning methods. The smoothing uses $M = 8$ for all the methods. All the attacks run with 5 random restarts. The shaded area is a standard deviation of the results over three runs. 120

8.6 Accuracy on CIFAR-10 under PGD or EPGD attacks as a function of number of attack steps, $k \cdot m$. The model for MART is WideResNet-34-10 and the rest are WideResNet-28-4, all models are fine-tuned on PGD with $k = 80$. EPGD attacks are only used when specified, and the gradients are averaged over $M_{\text{backward}} = 8$ samples in each attack iteration. The smoothing uses $M = 8$ and $\sigma = 0.25$. All the attacks run with 5 random restarts. Shaded area is a standard deviation of the results over three runs. 120
8.7 Accuracy on CIFAR-100 under PGD or EPGD attacks as a function of number of attack steps, $k \cdot M$. The model is WideResNet-28-4 defended with CNI and fine-tuned on EPGD with $k = 10$ and $M = 8$. EPGD attacks are only used when specified, and the gradients are averaged over $M_{\text{backward}} = 8$ samples in each attack iteration. The smoothing uses $M = 8$ and $\sigma = 0.25$. All the attacks run with 5 random restarts. Shaded area is a standard deviation of the results over three runs.

9.1 An adversarial example from the test set of the Twitter dataset. An adversarially-crafted post perturbs the representation of the attacker node. This perturbation causes a misclassification of the target victim node, although they are not even direct neighbors.

9.2 Effectiveness of our SINGLE attack compared to the allowed $\epsilon_\infty$ (on PubMed).

9.3 Test accuracy under our SINGLE attack, compared to $\epsilon_0$, the maximal ratio of perturbed features.

9.4 Test accuracy compared to the distance between the attacker and the victim, when trained with $L = 8$ layers.
Abstract

Deep neural networks (DNN) have became a common tool for solving complex tasks in various fields such as computer vision, natural language processing, and recommendation systems. Despite the recent progress made in enhancing DNN performance, two major obstacles hinder their practical applicability in some applications: (i) their energy-expensive deployment on embedded platforms, and (ii) their amenability to malicious adversarial perturbations. In this thesis, we present our works focusing on different aspects of both these problems. Chapters 2 and 3 present a training-aware and post-training quantization approaches, which present the DNNs parameters and feature maps represented in fixed low-bit representations. Chapter 4 introduces a neural architectural search that allows to find optimal quantization bitwidths of neural network parameters for given complexity constraint. Chapters 5 and 6 present two entropy coding-based methods for reducing inference-time memory bandwidth requirements. The first method does not require any fine-tuning, while the second does and, in exchange, provides significant further bandwidth reduction with negligible additional complexity or accuracy reduction. Chapter 7 presents a simple framework that helps to design efficient hardware for quantized neural networks. In addition, in chapter 8 we show how quantization techniques can inspire new approaches to better cope with adversarial attacks, as well as demonstrate how an adversarially pre-trained classifier could boost adversarial robustness by smoothing between different levels of input noise. Finally, Chapter 9 introduces a simple single-node minimal attribute changing perturbation that can attack social graph-based DNNs, in a significantly more harmful way than the previously studied edge-based attacks.
## Abbreviations and Notations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIC</td>
<td>Application-Specific Integrated Circuit</td>
</tr>
<tr>
<td>CAD</td>
<td>Computer-Aided Design</td>
</tr>
<tr>
<td>CNN</td>
<td>convolutional Neural Network</td>
</tr>
<tr>
<td>DDR</td>
<td>Double Data Rate (Memory)</td>
</tr>
<tr>
<td>DL</td>
<td>Deep Learning</td>
</tr>
<tr>
<td>FLOPS</td>
<td>Floating point Operations</td>
</tr>
<tr>
<td>FMA</td>
<td>Fused Multiply-Add</td>
</tr>
<tr>
<td>GOPS</td>
<td>Giga Operations</td>
</tr>
<tr>
<td>HCM</td>
<td>Hardware aware Complexity Metric</td>
</tr>
<tr>
<td>HDL</td>
<td>Hardware Description Language</td>
</tr>
<tr>
<td>IP</td>
<td>Intellectual Property</td>
</tr>
<tr>
<td>MAC</td>
<td>Multiply Accumulate</td>
</tr>
<tr>
<td>NN</td>
<td>Neural Network</td>
</tr>
<tr>
<td>OPS</td>
<td>Operations</td>
</tr>
<tr>
<td>PE</td>
<td>Processing Engine</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>SRAM</td>
<td>Static Random Access Memory</td>
</tr>
<tr>
<td>TOPS</td>
<td>Tera Operations</td>
</tr>
<tr>
<td>TSMC</td>
<td>Taiwan Semiconductor Manufacturing Company</td>
</tr>
<tr>
<td>VLSI</td>
<td>Very Large-Scale Integration</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Deep Neural Networks (DNNs) have achieved spectacular success in various applications and fields such as image classification [SZ15], speech recognition [HSP+19], semantic analysis [VSP+17] etc. Despite state-of-the-art results achieved by neural networks models, two main drawbacks hinder DNNs from being practical in various applications: energy-expensive deployment on embedded platforms and their amenability to malicious adversarial perturbation. This thesis present several works that tackle both these problem. These works can be divided into methods targeting efficient DNN deployment on custom hardware accelerators and improving neural network robustness.

**Efficient DNN deployment.** The main approach for achieving DNN compression is to apply quantization of neural network weights and activation. This approach can be divided into quantization-aware training [HCS+16, HCS+17], and post-training regime [CKYK19, BNHS18]. Extending earlier quantization-aware training approaches, we propose a scheme that directly minimizes the quantization error by injecting noise that fits the real DNN parameter’s distribution and emulates the quantization error occurring during the quantization process. We also discover that the loss landscape is non-separable and have a sharp-minima for low-bit scenario. This is the main reason of significant degradation of accuracy in previously proposed post-training methods based on layer-wise optimization. To tackle this, we propose global joint optimization which find the optimal parameters that minimize the quantization error, allowing the method to achieve decent performance even for lower than 4-bit quantization in post-training regimes. In addition, we propose a novel neural network compression method that directly decreases the access for memory devices during DNN execution. As shown by [YCS17], memory access constitute a significant part of total power consumption of hardware DNN accelerators. We propose two configurations of the memory-compression scheme. The first configuration relies on PCA dimensional reduction on top of loss-less Huffman compression working in the post-training regime and the second configuration introduces a dedicated loss term that minimizes the mutual information among layer feature maps. To complete our investigation of the effectiveness of compression meth-
ods, we propose a hardware framework that analyzes the efficiency of quantized neural network hardware implementation, indicating how to improve the specific pitfalls in the design. Following our discussion on the importance of neural network complexity and its impact on hardware implementation efficiency, we propose a differentiable neural architectural search scheme. This method enables us to find an optimal quantization bandwidth configuration for a given complexity, which is appliable both for filter and layer granularity.

**Neural network robustness** Previously [SZS\textsuperscript{+}13, SLJ\textsuperscript{+}15] proposed an adversarial training method which aimed at improving adversarial robustness for image classification tasks. This approach relies on a long training regime and, in general, cannot be scaled for different settings of the adversarial attacks whiteout additional fine-tinning. Inspired by the previously proposed method of injecting noise for quantization-error minimization, we proposed a method that improves the adversarial robustness of the adversarial trained classifier without any additional fine-tuning by multiple sampling of the DNN input images with different low-magnitude noise finding the optimal value over classifier output. In addition, we explore Graph Neural Networks’ (GNNs) adversarial robustness. Most work in this field has focused on designing new GNN variants and applying them to a growing number of domains. However, only a few works, however, have explored the vulnerability of GNNs to realistic adversarial examples.

Consider the following scenario: a malicious user or bot joins a social network such as Twitter or Facebook. The malicious user mocks the behavior of a benign user, establishes connections with other users, and submits benign posts. After some time, the user submits a new adversarially crafted post, which might seem irregular but is overall benign. If the social network uses a GNN-based model to detect malicious users, the new adversarial post changes the representation of the user as seen by the GNN. As a result, another specific benign user gets blocked from the network; alternatively, another malicious user submits an inciteful or racist post – but does not get blocked. We show the feasibility of such a troublesome scenario: A single attacker node can perturb its own representation, such that another node will be misclassified as a label of the attacker’s choice. In the next chapters we provide a detailed explanation of the aforementioned approaches including extensive evaluation using common datasets from different domains such as computer vision, social graphs and semantic analysis. Below we summarize the contributions made by this thesis.

### 1.1 Contribution

This thesis makes several contributions:

- We propose a quantization-aware method that directly minimizes the quantization error outperforms previous methods in trade-off complexity vs accuracy.
• We present a post-quantization scheme that manages to reduce the number of parameter bits remarkably, e.g., to 3 to 4-bits without a significant reduction in performance.

• We present an early-stage hardware framework that allows us to identify the efficiency of the neural network implementation.

• We propose an alternative compression-aware method that decreases the required memory bandwidth during neural network execution.

• We show that multiple sampling of DNN inputs coupled with low-magnitude noise significantly improves adversarial robustness on top of an adversarially trained classifier.

• We show that a minimal change in an arbitrarily selected node can lead to misclassification of each node in social graph datasets.
Chapter 2

UNIQ: Uniform Noise Injection for Non-Uniform Quantization of Neural Networks

2.1 Motivation

We present a novel method for neural network quantization. Our method, named UNIQ, emulates a non-uniform $k$-quantile quantizer and adapts the model to perform well with quantized weights by injecting noise to the weights at training time. As a by-product of injecting noise to weights, we find that activations can also be quantized to as low as 8-bit with only a minor accuracy degradation. Our non-uniform quantization approach provides a novel alternative to the existing uniform quantization techniques for neural networks. We further propose a novel complexity metric of number of bit-operations performed (BOPs), we show this metric has a linear relation with logic utilization and power. We suggest evaluating the trade-off of accuracy vs. complexity (BOPs). The proposed method, when evaluated on ResNet18/34/50 and MobileNet on ImageNet, outperforms the prior art both in the low complexity regime and the high accuracy regime. We demonstrate the practical applicability of this approach, by implementing our non-uniformly quantized CNN on FPGA.
2.2 Introduction

Deep neural networks are widely used in many fields today including computer vision, signal processing, computational imaging, image processing, and speech and language processing [HDY\textsuperscript{+}12, LXLZ15, CPK\textsuperscript{+}18]. Yet, a major drawback of these deep learning models is their storage and computational cost. Typical deep networks comprise millions of parameters and require billions of multiply-accumulate (MAC) operations. In many cases, this cost renders them infeasible for running on mobile devices with limited resources. While some applications allow moving the inference to the cloud, such an architecture still incurs significant bandwidth and latency limitations.

A recent trend in research focuses on developing lighter deep models, both in their memory footprint and computational complexity. The main focus of most of these works, including this one, is on alleviating complexity at inference time rather than simplifying training. While training a deep model requires even more resources and time, it is usually done offline with large computational resources.

One way of reducing computational cost is quantization of weights and activations. Quantization of weights also reduces storage size and memory access. The bit widths of activations and weights affect linearly the amount of required hardware logic; reducing both bitwidths by a factor of two reduces the amount of hardware by roughly a factor of four. (A more accurate analysis of the effects of quantization is presented in Section 2.4.2). Quantization allows fitting bigger networks into a device, which is especially critical for embedded devices and custom hardware. On the other hand, activations are passed between layers, and thus, if different layers are processed separately, activation size reduction reduces communication overheads.

Deep neural networks are usually trained and operate with both the weights and the activations represented in single-precision (32-bit) floating point. A straightforward uniform quantization of the pre-trained weights to 16-bit fixed point representation has been shown to have a negligible effect on the model accuracy [GAGN15]. In the majority of the applications, further reduction of precision quickly degrades performance; hence, nontrivial techniques are required to carry it out.

2.2.1 Neural Networks on custom hardware

When implementing systems involving arbitrary precision, FPGAs and ASICs are a natural selection as target device due to their customizable nature. It was already shown that there is a lot of redundancy when using floating point representation in Neural Network (NN). Therefore, custom low-precision integer representation can be used with little impact to the accuracy. Due to the steadily increasing on-chip memory size (tens of megabytes) and the integration of high bandwidth memory (hundreds of megabytes), it is feasible to fit all the parameters inside an ASIC or FPGA, when using low bitwidth. Besides the obvious advantage of reducing the latency, this approach has several advantages: power consumption reduction and smaller resource utilization,
which in addition to DSP blocks and LUTs, also includes routing resource. The motivation of quantizing the activations is similar to that of the parameters. Although activations are not stored during inference, their quantization can lead to major saving in routing resources which in turn can increase the maximal operational frequency of the fabric, resulting in increased throughput.

Contribution. Most previous quantization solution for neural networks assume a uniform distribution of the weights, which is non-optimal due to the fact these rather have bell-shaped distributions [HMD16]. To facilitate this deficiency, we propose a $k$-quantile quantization method with balanced (equal probability mass) bins, which is particularly suitable for neural networks, where outliers or long tails of the bell curve are less important. We also show a simple and efficient way of reformulating this quantizer using a “uniformization trick”.

In addition, we introduce a novel method for training a network that performs well with quantized values. This is achieved by adding noise (at training time) to emulate the quantization noise introduced at inference time. The uniformization trick renders exactly the injection of uniform noise and alleviates the need to draw the noise from complicated and bin-dependent distributions. While we limit our attention to the $k$-quantile quantization, the proposed scheme can work with any threshold configuration while still keeping the advantage of uniformly distributed noise in every bin.

Since the calculations of low-precision networks are composed of integer operations, their computation complexity cannot be measured by FLOPS. Therefore, we propose a novel metric for complexity of fixed-point quantized NN models, called Bit Operations (BOPs). We show that this metric has a linear relation with logic utilization and power. Measuring the complexity of each method by bit operations (BOPS), our UNIQ strategy compares favorably, in the accuracy vs. complexity trade-off, to other leading methods such as Distillation [MM18] and MLQ [XWZ+18]. In addition, our method performs well on smaller models targeting mobile devices, represented in the paper by MobileNetV1. Those network are known to be harder to quantize, due to lower redundancy in parameters.

Finally, we suggest a look-up table approach for hardware implementation of non-uniformly quantized CNNs. We implement it for an FPGA and show it to be more power efficient and with higher throughput compared to the usual low-bit integers arithmetic.
2.3 Non-uniform quantization by uniform noise injection

To present our uniform noise injection quantization (UNIQ) method for training a neural network amenable to operation in low-precision arithmetic, we start by outlining several common quantization schemes and discussing their suitability for deep neural networks. Then, we suggest a training procedure where during training time uniform random additive noise is injected into weights simulating the quantization error. The scheme aims at improving the quantized network performance at inference time, when regular deterministic quantization is used.

2.3.1 Quantization

Let $X$ be a random variable drawn from some distribution described by the probability density function $f_X$. Let $T = \{t_i\}$ with $t_0 = -\infty$, $t_k = \infty$ and $t_{i-1} < t_i$ be a set of thresholds partitioning the real line into $k$ disjoint intervals (bins) $\{[t_{i-1}, t_i]\}_{i=1}^k$, and let $Q = \{q_i\}_{i=1}^k$ be a set of $k$ representation levels. A quantizer $Q_{T,Q}$ is a function mapping each bin $[t_{i-1}, t_i]$ to the corresponding representation level $q_i$. We denote the quantization error by $E = X - Q_{T,Q}(X)$. The effect of quantization can be modeled as the addition of random noise to $X$; the noise added to the $i$-th bin admits the conditional distribution $(X - q_i) | X \in [t_{i-1}, t_i]$.

Most papers on neural network quantization focus on the uniform quantizer, which has a constant bin width $t_i - t_{i-1} = \Delta$ and $q_i = (t_{i-1} + t_i)/2$, it is known to be optimal (in the sense of the mean squared error $\mathbb{E}E^2$, where the expectation is taken with respect to the density $f_X$) in the case of uniform distribution. Yet, since $X$ in neural networks is not uniform but rather bell shaped [HMD16], in the general case the optimal choice, in the $\ell_2$ sense, is the $k$-means quantizer. Its name follows the property that each representation level $q_i$ coincides with the $i$-th bin centroid (mean w.r.t. $f_X$). While finding the optimal $k$-means quantizer is known to be an NP-hard problem, heuristic procedures such as the Lloyd-Max algorithm [Llo82] usually produce a good approximation. The $k$-means quantizer coincides with the uniform quantizer when $X$ is uniformly distributed.

While being a popular choice in signal processing, the $k$-means quantizer encounters severe obstacles in our problem of neural network quantization. Firstly, the Lloyd-Max algorithm has a prohibitively high complexity to be used in every forward pass. Secondly, it is not easily amenable to our scheme of modeling quantization as the addition of random noise, as the noise distribution at every bin is complex and varies with the change of the quantization thresholds. Finally, our experiments shown in Section 2.4.3 in the sequel suggest that the use of the $\ell_2$ criterion for quantization of deep neural classifier does not produce the best classification results. The weights in such networks typically assume a bell-shaped distribution with tails exerting a great effect on the mean squared error, yet having little impact on the classification accuracy.

Based on empirical observations, we conjecture that the distribution tails, which
$k$-means is very sensitive to, are not essential for good model performance at least in classification tasks. As an alternative to $k$-means, we propose the $k$-quantile quantizer characterized by the equiprobable bins property, that is, $P(X \in [t_{i-1}, t_i]) = 1/k$. The property is realized by setting $t_i = F_X^{-1}(i/k)$, where $F_X$ denotes the cumulative distribution function of $X$ and, accordingly, its inverse $F_X^{-1}$ denotes the quantile function. The representation level of the $i$-th bin is set to the bin median, $q_i = \text{med}\{X | X \in [t_{i-1}, t_i]\}$. It can be shown that in the case of a uniform $X$, the $k$-quantile quantizer coincides with the $k$-level uniform quantizer.

The cumulative distribution $F_X$ and the quantile function $F_X^{-1}$ can be estimated empirically from the distribution of weights, and updated in every forward pass. Alternatively, one can rely on the empirical observation that the $\ell_2$-regularized weights of each layer tend to follow an approximately normal distribution [BCKW15]. To confirm that this is the case for the networks used in the paper, we analyzed the distribution of weights. An example of a layer-wise distribution of weights is shown in the supplementary material. Relying on this observation, we can estimate $\mu$ and $\sigma$ per each layer and use the CDF of the normal distribution (and its inverse, the normal quantile function).

Using the fact that applying a distribution function $F_X$ of $X$ to itself results in a uniform distribution allows an alternative construction of the $k$-quantile quantizer. We apply the transformation $U = F_X(X)$ to the input converting it into a uniform random variable on the interval $[0, 1]$. Then, a uniform $k$-level quantizer (coinciding with the $k$-quantile quantizer) is applied to $U$ producing $\hat{U} = Q_{\text{uni}}(U)$; the result is transformed back into $\hat{X} = F_X^{-1}(\hat{U})$ using the inverse transformation. We refer to this procedure as the uniformization trick. Its importance will become evident in the next section.

### 2.3.2 Training quantized neural networks by uniform noise injection

The lack of continuity, let alone smoothness, of the quantization operator renders impractical its use in the backward pass. As an alternative, at training we replace the quantizer by the injection of random additive noise. This scheme suggests that instead of using the quantized value $\hat{w} = Q_{T,Q}(w)$ of a weight $w$ in the forward pass, $\hat{w} = w + e$ is used with $e$ drawn from the conditional distribution of $(W - q_i) | W \in [t_{i-1}, t_i]$ described by the density

$$f_E(e) = \frac{f_W(e + q_i)}{\int_{t_{i-1}}^{t_i} f_W(w) \, dw} \quad (2.1)$$

defined for $e \in [t_{i-1} - q_i, t_i - q_i]$ and vanishing elsewhere. The bin $i$ to which $w$ belongs is established according to its value and is fixed during the backward pass. Quantization of the network activations is performed in the same manner.

The fact that the parameters do not directly undergo quantization keeps the model differentiable. In addition, gradient updates in the backward pass have an immediate impact on the forward pass, in contrast to the directly quantized model, where small
updates often leave the parameter in the same bin, leaving it effectively unchanged.

While it is customary to model the quantization error as noise with uniform distribution $f_{E}$ [GN98], this approximation breaks in the extremely low precision regimes (small number of quantization levels $k$) considered here. Hence, the injected noise has to be drawn from a potentially non-uniform distribution which furthermore changes as the network parameters and the quantizer thresholds are updated.

To overcome this difficulty, we resort to the uniformization trick outlined in the previous sub-section. Instead of the $k$-quantile quantizer $w' = Q(w)$, we apply the equivalent uniform quantizer to the uniformized variable, $\hat{w} = F_W^{-1}(Q_{uni}(F_W(w)))$. The effect of the quantizer can be again modeled using noise injection, $\hat{w} = F_W^{-1}(F_W(w) + e)$, with the cardinal difference than now the noise $e$ is uniformly distributed on the interval $[-\frac{1}{2k}, \frac{1}{2k}]$ (estimating quantization error distribution).

Usually, quantization of neural networks is either used for training a model from scratch or applied post-training as a fine-tuning stage. Our method, as will be demonstrated in our experiments, works well in both cases. Our practice shows that best results are obtained when the learning rate is reduced as the noise is added; we explain this by the need to compensate for noisier gradients.

### 2.3.3 Gradual quantization

The described method works well “as is” for small- to medium-sized neural networks. For deeper networks, the basic method does not perform as well, most likely due to errors arising when applying a long sequence of operations where more noise is added at each step. We found that applying the scheme gradually to small groups of layers works better in deep networks. In order to perform gradual quantization, we split the network into $N$ blocks $\{B_1, ..., B_N\}$, each containing about same number of consecutive layers. We also split our budget of training epochs into $N$ stages. At the $i$-th stage, we quantize and freeze the parameters in blocks $\{B_1, ..., B_{i-1}\}$, and inject noise into the parameters of $B_i$. For the rest of the blocks $\{B_{i+1}, ..., B_N\}$ neither noise nor quantization is applied.

This approach is similar to one proposed by [XWZ+18]. This way, the number of parameters into which the noise is injected simultaneously is reduced, which allows better convergence. The deeper blocks gradually adapt to the quantization error of previous ones and thus tend to converge relatively fast when the noise is injected into them. For fine-tuning a pre-trained model, we use the same scheme, applying a single epoch per stage. An empirical analysis of the effect of the different number of stages is presented in the supplementary material. This process can be performed iteratively, restarting from the beginning after the last layer has been trained. Since this allows earlier blocks to adapt to the changed values of the following ones, the iterative process yields an additional increase in accuracy. Two iterations were performed in the reported experiments.
2.3.4 Activations quantization

Activations quantization is also beneficial in lowering the arithmetic complexity and can help decreasing the communication overhead in the distributed model case. We observed that a by-product of training with noisy weights is that the model becomes less sensitive to a certain level of quantization of the activations. In training time, like in the case of the weights, we apply the CDF \( F_A(a) \) to uniformize the activations, but we do not apply the inverse CDF and treat the uniformization as part of the non-linear activation function. The reason for dropping the inverse CDF is that we observed similar accuracy performance with and without it. We observed that relying solely on the CDF as non-linearity cause accuracy performance degradation so we apply both CDF and ReLU. In inference time, we again apply the CDF followed by uniform quantization. We use the same quantizer for activations as used for weights with the slight modification that the first bin captures all the negative values \([-\infty, 0]\) and assign then the value 0 (the effect of ReLU activation).
2.4 Experimental evaluation

We performed an extensive performance analysis of the UNIQ scheme compared to the current state-of-the-art methods for neural network quantization. The basis for comparison is the accuracy vs. the total number of bit operations in visual classification tasks on the ImageNet-1K [RDS+15] dataset. CIFAR-10 dataset [Kri09a] is used to evaluate different design choices made, while the main evaluation is performed on ImageNet-1K.

MobileNet [HZC+17] and ResNet-18/34/50 [HZRS16a] architectures are used as the baseline for quantization. MobileNet is chosen as a representative of lighter models, which are more suited for a limited hardware setting where quantization is also most likely to be used. ResNet-18/34/50 is chosen due to its near state-of-the-art performance and popularity, which makes it an excellent reference for comparison.

We adopted the number of bit operations (BOPs) metric to quantify the network arithmetic complexity. This metric is particularly informative about the performance of mixed-precision integer arithmetic especially in hardware implementations on FPGAs and ASICs.

Training details For quantizing a pre-trained model, we train with SGD for a number of epochs equal to the number of trainable layers (convolution and fully connected). We also follow the gradual process described above with the number of stages set to the number of trainable layers. The learning rate is $10^{-4}$, momentum 0.9 and weight decay $10^{-4}$. In all experiments, unless otherwise stated, we fine-tuned a pre-trained model taken from an open-sourced repository of pre-trained models.\footnote{https://github.com/Cadene/pretrained-models.pytorch}

2.4.1 Performance of quantized networks on ImageNet

Table 2.4 compares the ResNet and MobileNet performance with weights and activations quantized to several levels using UNIQ and other leading approaches reported in the literature. For baseline, we use a full-precision model with 32 bit weights and activations. Note that the common practice of not quantizing first and last layers significantly increases the network complexity in BOPs. In contrast, our model performs quite well even when we quantize all the layers, and thus achieves lower complexity requirements with a higher bitwidth. Note also the diminishing impact of the weights bitwidth on the BOP complexity as explained hereafter.

We found UNIQ to perform well also with the smaller MobileNet architecture. This is in contrast to most of the previous methods that resort to larger models and doubling the number of filters [PPA18], thus quadrupling the number of parameters, e.g., from 11 to 44 million for ResNet-18.
2.4.2 Accuracy vs. complexity trade-off

Since custom precision data types are used for the network weights and activations, the number of MAC operations is not an appropriate metric to describe the computational complexity of the model. Therefore, we use the BOPs metric quantifying the number of bit operations. Given the bitwidth of two operands, it is possible to approximate the number of bit operations required for a basic arithmetic operation such as addition and multiplication. The proposed metric is useful when the inference is performed with fixed-point operations on custom hardware like FPGAs or ASICs. Both are a natural choice for quantized networks, due to the use of look-up tables (LUTs) and dedicated MAC (or more general DSP) units, which are efficient with custom data types.

An important phenomenon that can be observed in Table 2.4 is the non-linear relation between the number of activation and weight bits and the resulting network complexity in BOPs. To quantify this effect, let us consider a single convolutional layer with \( b_w \)-bit weights and \( b_a \)-bit activations containing \( n \) input channels, \( m \) output channels, and \( k \times k \) filters. The maximum value of a single output is about \( 2^{b_a+b_w}nk^2 \), which sets the accumulator width in the MAC operations to \( b_o = b_a + b_w + \log_2 nk^2 \). The complexity of a single output calculation consists therefore of \( nk^2 b_a \)-wide \( \times b_w \)-wide multiplications and about the same amount of \( b_o \)-wide additions. This yields the total layer complexity of

\[
\text{BOPs} \approx mnk^2(b_a b_w + b_a + b_w + \log_2 nk^2).
\]  

(2.2)

Note that the reduction of the weight and activation bitwidth decreases the number of BOPs as long as the factor \( b_a b_w \) dominates the factor \( \log_2 nk^2 \). Since the latter factor depends only on the layer topology, this point of diminishing return is network architecture-dependent.

Table 2.1: UNIQ accuracy on CIFAR-10 for different bitwidth.

<table>
<thead>
<tr>
<th>Activation bits</th>
<th>4</th>
<th>8</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight bits</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>88.1</td>
<td>90.88</td>
<td>89.14</td>
</tr>
<tr>
<td>4</td>
<td>89.5</td>
<td>91.5</td>
<td>89.70</td>
</tr>
<tr>
<td>32</td>
<td>88.52</td>
<td>91.32</td>
<td>92.00</td>
</tr>
</tbody>
</table>

Table 2.2: UNIQ with different quantization methods (ResNet-18 top-1 accuracy on CIFAR-10, 3-bit weights)

<table>
<thead>
<tr>
<th>Quantizer used</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline (unquantized)</td>
<td>92.00</td>
</tr>
<tr>
<td>( k )-quantile</td>
<td>91.30</td>
</tr>
<tr>
<td>( k )-means</td>
<td>85.80</td>
</tr>
<tr>
<td>Uniform</td>
<td>84.93</td>
</tr>
</tbody>
</table>

Another factor that must be incorporated into the BOPs calculation is the cost of fetching the parameters from an external memory. Three assumptions are made in the approximation of this cost: firstly, we assume that on-chip memory is used where memory access time is comparable to arithmetic operations; secondly, we assume that each parameter is only fetched once from an external memory; thirdly, the cost of
fetching a \( b \)-bit parameter is assumed to be \( b \) BOPs. Given a neural network with \( n \) parameters all represented in \( b \) bits, the memory access cost is simply \( nb \).

Figure 2.2 and Table 2.4 display the performance-complexity trade-offs of various neural networks trained using UNIQ and other methods to different levels of weight and activation quantization. Notice that since we quantize the first and last layers, our ResNet-34 network has better computational complexity and better accuracy compared to all competing ResNet-18 networks. The same holds for our ResNet-50 compared to all competing ResNet-34.

2.4.3 Ablation study

Accuracy vs. quantization level. We tested the effect of training ResNet-18 on CIFAR-10 with UNIQ for various levels of weight and activation quantization. Table 2.1 reports the results. We observed that for such a small dataset the quantization of activations and weights helps avoid over-fitting and the results for quantized model come very close to those with full precision.

Comparison of different quantizers. In the following experiment, different quantizers were compared within the uniform noise injection scheme.

The bins of the uniform quantizer were allocated evenly in the range \([-3\sigma, 3\sigma]\) with \(\sigma\) denoting the standard deviation of the parameters. For both the \(k\)-quantile and the \(k\)-means quantizers, normal distribution of the weights was assumed and the normal cumulative distribution and quantile functions were used for the uniformization and deuniformization of the quantized parameter. The \(k\)-means and uniform quantizers used a pre-calculated set of thresholds translated to the uniformized domain. Since the resulting bins in the uniformized domain had different widths, the level of noise was different in each bin. This required an additional step of finding the bin index for each parameter approximately doubling the training time.

The three quantizers were evaluated in a ResNet-18 network trained on the CIFAR-10 dataset with weights quantized to 3 bits \((k = 8)\) and activations computed in full precision \((32\) bit). Table 2.2 reports the obtained top-1 accuracy. \(k\)-quantile quantization outperforms other quantization methods and is only slightly inferior to the full-precision baseline. In terms of training time, the \(k\)-quantile quantizer requires about 60\% more time to train for \(k = 8\); this is compared to around 280\% increase in training time required for the \(k\)-means quantizer. In addition, \(k\)-quantile training time is independent on the number of quantization bins as the noise distribution is same for every bin while the other methods require separate processing of each bin, increasing the training time for higher bitwidths.

Training from scratch vs. fine-tuning. Both training from scratch (that is, from random initialization) and fine-tuning have their advantages and disadvantages. Training from
scratch takes more time but requires a single training phase with no extra training epochs, at the end of which a quantized model is obtained. Fine-tuning, on the other hand, is useful when a pre-trained full-precision model is already available; it can then be quantized with a short re-training.

Table 2.3 compares the accuracy achieved in the two regimes on a narrow version of ResNet-18 trained on CIFAR-10 and 100. 5-bit quantization of weights only and 5-bit quantization of both weights and activations were compared. We found that both regimes work equally well, reaching accuracy close to the full-precision baseline.

Table 2.3: Top-1 accuracy (in percent) on CIFAR-10 and 100 of a narrow version on ResNet-18 trained with UNIQ from random initialization vs. fine-tuning a full-precision model. Number of bits is reported as (weights,activations).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Bits</th>
<th>Full training</th>
<th>Fine-tuning</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td>5.32</td>
<td>93.8</td>
<td>90.9</td>
<td>92.0</td>
</tr>
<tr>
<td></td>
<td>5.5</td>
<td>91.56</td>
<td>91.21</td>
<td></td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>5.32</td>
<td>66.54</td>
<td>65.73</td>
<td>66.3</td>
</tr>
<tr>
<td></td>
<td>5.5</td>
<td>65.29</td>
<td>65.05</td>
<td></td>
</tr>
</tbody>
</table>

**Accuracy vs. number of quantization stages.** We found that injecting noise to all layers simultaneously does not perform well for deeper networks. As described in Section 2.3.3, we suggest splitting the training into $N$ stages, such that at each stage the noise is injected only into a subset of layers.

To determine the optimal number of stages, we fine-tuned ResNet-18 on CIFAR-10 with a fixed 18 epoch budget. Bit width was set to 4 for both the weights and the activations.

Figure 2.1 reports the classification accuracy as a function of the number of quantization stages. Based on these results, we conclude that the best strategy is injecting noise to a single layer at each stage. We follow this strategy in all experiments conducted in this chapter.
Figure 2.1: Classification accuracy on CIFAR-10 of the ResNet-18 architecture quantized using UNIQ with different number of quantization stages during training.

Figure 2.2: **Performance vs. complexity of different quantized neural networks.** Performance is measured as top-1 accuracy on ImageNet; complexity is estimated in number of bit operations. Network architectures are denoted by different marker shapes; quantization methods are marked in different colors. Multiple instances of the same method (color) and architecture (shape) represent different level of quantization. Note that our method performs better for ResNet-50, comparable for ResNet-34, and worse for ResNet-18, this is due to the fact that in Apprentice [MM18] the baselines (full-precision) models are better for smaller models. Our model also performs well for the smaller more efficient MobileNet architecture which is known to be an harder problem. The figure should be viewed in color.
Table 2.4: Complexity-accuracy trade-off of various DNN architectures quantized using different techniques. Complexity is reported in number of bit operations as explained in the text. Number of bits is reported as (weights, activations), (weights+, activations+) indicates that the first and last layers are not quantized (full-precision). Model size is calculated as the sum of parameters sizes in bits. Accuracy is top-1 accuracy on ImageNet. For each DNN architecture, rows are sorted in increasing order complexity.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Method</th>
<th>Bits (w,a)</th>
<th>Model size [Mbit]</th>
<th>Complexity [GBOPs]</th>
<th>Accuracy (% top-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNet</td>
<td>UNIQ</td>
<td>4,8</td>
<td>16.8</td>
<td>25.1</td>
<td>66.00</td>
</tr>
<tr>
<td>MobileNet</td>
<td>UNIQ</td>
<td>5,8</td>
<td>20.8</td>
<td>30.5</td>
<td>67.50</td>
</tr>
<tr>
<td>MobileNet</td>
<td>QSM [SFZ+ 18a]</td>
<td>8+,8+</td>
<td>33.6</td>
<td>46.7</td>
<td>68.01</td>
</tr>
<tr>
<td>MobileNet</td>
<td>UNIQ</td>
<td>8,8</td>
<td>33.6</td>
<td>46.7</td>
<td>68.25</td>
</tr>
<tr>
<td>MobileNet</td>
<td>Baseline</td>
<td>32,32</td>
<td>135.2</td>
<td>626</td>
<td>68.20</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>UNIQ</td>
<td>4,8</td>
<td>46.4</td>
<td>81.5</td>
<td>67.02</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>Apprentice [MM18]</td>
<td>2+,8+</td>
<td>39.2</td>
<td>83</td>
<td>67.6</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>UNIQ</td>
<td>5,8</td>
<td>58.4</td>
<td>99.5</td>
<td>68.00</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>Apprentice [MM18]</td>
<td>4+,8+</td>
<td>61.4</td>
<td>114</td>
<td>70.40</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>UNIQ</td>
<td>4+,8+</td>
<td>61.4</td>
<td>114</td>
<td>69.12</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>UNIQ</td>
<td>5+,8+</td>
<td>72.6</td>
<td>130</td>
<td>69.5</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>Apprentice [MM18]</td>
<td>2+,32</td>
<td>39.2</td>
<td>256</td>
<td>68.50</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>IQN [ZYG+17]</td>
<td>5+,32</td>
<td>58.4</td>
<td>315</td>
<td>68.89</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>MLQ [XWZ+18]</td>
<td>5+,32</td>
<td>58.4</td>
<td>315</td>
<td>69.09</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>Distillation [PPA18]</td>
<td>4+,32</td>
<td>61.6</td>
<td>367</td>
<td>64.20</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>Baseline</td>
<td>32,32</td>
<td>374.4</td>
<td>1924</td>
<td>69.60</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>Apprentice [MM18]</td>
<td>2+,8+</td>
<td>59.3</td>
<td>135</td>
<td>71.5</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>UNIQ</td>
<td>4,8</td>
<td>87.2</td>
<td>166</td>
<td>71.09</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>Apprentice [MM18]</td>
<td>4+,8+</td>
<td>101.8</td>
<td>198</td>
<td>73.1</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>UNIQ</td>
<td>4+,8+</td>
<td>101.8</td>
<td>198</td>
<td>72.19</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>UNIQ</td>
<td>5,8</td>
<td>109</td>
<td>202</td>
<td>72.60</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>UNIQ</td>
<td>5+,8+</td>
<td>123.1</td>
<td>234</td>
<td>73.22</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>Apprentice [MM18]</td>
<td>2+,32</td>
<td>59.3</td>
<td>396</td>
<td>72.8</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>UNIQ</td>
<td>4,32</td>
<td>101.8</td>
<td>516</td>
<td>73.1</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>Baseline</td>
<td>32,32</td>
<td>698</td>
<td>3903</td>
<td>73.4</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>Apprentice [MM18]</td>
<td>2+,8+</td>
<td>112.9</td>
<td>113</td>
<td>72.8</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>UNIQ</td>
<td>4,8</td>
<td>102.4</td>
<td>134</td>
<td>74.37</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>UNIQ</td>
<td>4+,8+</td>
<td>102.4</td>
<td>167</td>
<td>75.1</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>Apprentice [MM18]</td>
<td>4+,8+</td>
<td>160</td>
<td>167</td>
<td>74.7</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>Apprentice [MM18]</td>
<td>2+,32</td>
<td>112.8</td>
<td>344</td>
<td>74.7</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>UNIQ</td>
<td>4,32</td>
<td>102.4</td>
<td>423</td>
<td>75.09</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>Baseline</td>
<td>32,32</td>
<td>819.5</td>
<td>3232</td>
<td>76.02</td>
</tr>
</tbody>
</table>
2.5 Hardware Implementation

2.5.1 MAC operation for non-uniformly quantized data

From hardware implementation perspective, uniform quantization has low arithmetic complexity since the quantization is homomorphic and thus integer multiplication and addition can be used. However, non-uniform quantization is not homomorphic (e.g., product of elements mapped to 2 and 3 is not equal to element mapped to 6), and thus it is impossible to use integer operations anymore. A simple, but rather expensive approach would be to use floating point operations. To still benefit advantages of low-precision integer calculations along with non-uniform quantization, we suggest using look-up table approach: instead of calculating the product of two values during the inference, we pre-compute the result and store it in a table. Thus, any arithmetic operation requires a single memory access, at the expense of storing multiplication table.

In the inference phase, each layer will have its own $2^{BW_w}$ discrete values for weights, represented in floating-point format. Each of these discrete levels is associated with an integer index. For activations we use the training-set statistics of a specific layer to pre-compute the thresholds and their mapping to an integer index representation. Let $BW_w$, $BW_a$, $BW_p$ be number of bits for weights, activation and fixed-point representation of their product, respectively. For each layer, we construct a look-up table (LUT) with $2^{BW_w + BW_a}$ entries. Let $W_q$ and $A_q$ be two sets of discrete floating-point values of weights and activations, respectively. Let us also denote the outer product operator as $\otimes$. Table entries are $BW_p$ bits wide and the table is calculated using MinMax scaling as follows:

$$Q(W_q \otimes A_q) = \left\lfloor \frac{W_q \otimes A_q}{\max(W_q \otimes A_q) - \min(W_q \otimes A_q)} \left(2^{BW_p} - 1\right) \right\rfloor,$$

where $[\cdot]$ denotes the rounding operation. Since the entries in the LUT are represented in fixed-point representation, they can be summed up like integers to form the MAC result before the activation function and quantization are applied (by using pre-computed thresholds and their mapping to integers indexes).

An illustration of this function, for activation with $\mu = 900$ and $\sigma = 900$, is present in figure 2.3. From this we can clearly see that the distance on y-axis are equal which implies equiprobable bins. The intersection of the vertical green lines with the x-axis represent the integer thresholds which are loaded to the custom device, during inference, and define the boundaries between each bin.

2.5.2 Logic utilization and power consumption on FPGA

It is not reasonable to implement popular architectures such as ResNet and MobileNet, in a dataflow manner where all the layers are implemented on a single FPGA. The
more common and scalable approach is to implement a generic MAC calculation logic which can support multiple types of layers and multiple sizes. This MAC calculator is iteratively configured and called according to the current layer. We perform static power and logic utilization analysis for a single $256 \times 256 \times 3 \times 3$ representative convolution layer. We have implemented the proposed MAC calculation logic for both LUT-based (for non-uniform quantization) and DSP-based (for uniform quantization) approaches, on Intel’s Arria 10 FPGA. Table 5.3 reports logic utilization and power consumption, as reported by the Quartus Prime software for Intel FPGA synthesis and place & route. LUT-based realization outperforms the DSP-based one both in terms of maximal frequency (up to 60% higher). As for power consumption for the fixed frequency, LUT-based solution shows advantage for low bitwidth. However, since the LUT size grows exponentially with the number of bits, the approach is less efficient for higher bitwidths and the power consumption growth significantly.

To justify the use of the BOPs metric, we investigate the correlation between BOPs, logic utilization and power consumption. From Figure 2.4 we can observe a linear relation between our computational complexity metric and logic utilization and power consumption. This allows a more accurate system design, based on computational complexity vs. logic utilization and power trade-off.
Table 2.5: Uniform (DSPs) and Non-Uniform (LUTs) Logic utilization and power consumption on Intel’s Arria 10 FPGA

<table>
<thead>
<tr>
<th>Bits (w,a)</th>
<th>Method</th>
<th>LUTs</th>
<th>DSPs</th>
<th>BRAM [Kbits]</th>
<th>Frequency [MHz]</th>
<th>Power [mW]</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4,4)</td>
<td>LUTs</td>
<td>13353</td>
<td>0</td>
<td>2236</td>
<td>240 (max)</td>
<td>7133</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>150 (fixed)</td>
<td>6297</td>
</tr>
<tr>
<td></td>
<td>DSPs</td>
<td>15781</td>
<td>1152</td>
<td>0</td>
<td>185 (max)</td>
<td>7039</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>150 (fixed)</td>
<td>6900</td>
</tr>
<tr>
<td>(5,5)</td>
<td>LUTs</td>
<td>13353</td>
<td>0</td>
<td>8945</td>
<td>240 (max)</td>
<td>7145</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>150 (fixed)</td>
<td>6304</td>
</tr>
<tr>
<td></td>
<td>DSPs</td>
<td>18086</td>
<td>1152</td>
<td>0</td>
<td>185 (max)</td>
<td>7723</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>150 (fixed)</td>
<td>7221</td>
</tr>
<tr>
<td>(6,6)</td>
<td>LUTs</td>
<td>13353</td>
<td>0</td>
<td>35782</td>
<td>240 (max)</td>
<td>11369</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>150 (fixed)</td>
<td>8905</td>
</tr>
<tr>
<td></td>
<td>DSPs</td>
<td>21016</td>
<td>1152</td>
<td>0</td>
<td>185 (max)</td>
<td>8001</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>150 (fixed)</td>
<td>7539</td>
</tr>
</tbody>
</table>

Figure 2.4: Logic utilization and power consumption vs. number of bit operations. A significant correlation between proposed BOPs metric and both logic utilization and power consumption supports the hypothesis that BOPs is a good proxy to both. Note that LUTs here refer to the basic building block of FPGA rather than our implementation of multiplication.
2.6 Discussion

We introduced UNIQ – a training scheme for quantized neural networks. The scheme is based on the uniformization of the distribution of the (intended to be quantized) parameters, injection of additive uniform noise, followed by the de-uniformization of the obtained result. The scheme is amenable to efficient training by backpropagation in full-precision arithmetic, and achieves maximum efficiency with the $k$-quantile (balanced) quantizer that was investigated in this thesis.

To set a common basis for comparison, we proposed a novel measure of complexity called bit operations (BOPs). We have shown that this metric has a quasi-linear relation with logic utilization and power, which facilitates better accuracy-complexity trade-offs. We demonstrate that UNIQ achieves improved results for quantized neural networks on ImageNet. Our solution outperforms any other quantized network in terms of accuracy versus complexity for MobileNetV1 and ResNet-50 architectures, and achieves comparable results to current SOTA method [MM18] for ResNet-34.

In addition, we implemented our non-uniformly quantized network on FPGA and demonstrate that its power consumption is similar to that of a uniformly-quantized network operating with integer arithmetics.

While this work considered a setting in which all parameters have the same bitwidth, more complicated bit allocations will be explored in following studies. The proposed scheme is not restricted to the $k$-quantile quantizer discussed in this work, but rather applies to any quantizer. In the general case, the noise injected into each bin is uniform, but its variance changes with the bin width.
Chapter 3

Loss Aware Post-training Quantization

3.1 Motivation

Neural network quantization enables the deployment of large models on resource-constrained devices. Current post-training quantization methods fall short in terms of accuracy for INT4 (or lower) but provide reasonable accuracy for INT8 (or above). In this work, we study the effect of quantization on the structure of the loss landscape. We show that the structure is flat and separable for mild quantization, enabling straightforward post-training quantization methods to achieve good results. We show that with more aggressive quantization, the loss landscape becomes highly non-separable with steep curvature, making the selection of quantization parameters more challenging. Armed with this understanding, we design a method that quantizes the layer parameters jointly, enabling significant accuracy improvement over current post-training quantization methods.
Figure 3.1: A visualization of the loss surface for the pair of layers using one batch of 512 images (ResNet18 on ImageNet). X and Y-axis are quantization ranges of those layers where we estimate the cross-entropy loss (z-axis). The colored dots mark the quantization range found by optimizing different p-norm metrics. Each layer quantized in a way that minimizes the p-norm distance between the quantized tensor to its original full precision counterpart (e.g., MSE distortion). (a) One can see high coupling between two quantization ranges, making layer-wise optimization sub-optimal. (b) A zoom-in of sub-figure (a) to the point where the optimal cross-entropy loss (red cross) clearly distinguished from the optimized p-norm solutions.

3.2 Introduction

Deep neural networks (DNNs) are a powerful tool that have shown unmatched performance in various tasks in computer vision, natural language processing and optimal control, to mention only a few. The high computational resource requirements, however, constitute one of the main drawbacks of DNNs, hindering their massive adoption on edge devices. With the growing number of tasks performed on edge devices, e.g., smartphones or embedded systems, and the availability of dedicated custom hardware for DNN inference, the subject of DNN compression has gained popularity.

One way to improve DNN computational efficiency is to use lower-precision representation of the network, also known as quantization. Most of the literature on neural network quantization involves training either from scratch [JYL19] or performing fine-tuning on a pre-trained full-precision model [YSX+19]. While training is a powerful method to compensate for accuracy loss due to quantization, it is both resource consuming and requires access to the data the model was trained on, which is not always available. Thus, it is often desirable to be able to quantize the model without training. These methods are commonly referred to as post-training quantization and usually require only a small calibration dataset. Unfortunately, current post-training methods are not very efficient, and most existing works only manage to quantize parameters to the 8-bit integer representation (INT8).
In the absence of a training set, these methods typically aim at minimizing the local error introduced during the quantization process (e.g., round-off errors). A popular approach to minimizing this error has been to clip the tensor outliers. This means that peak values will incur a larger error, but, in total, this will reduce the distortion introduced by the limited resolution \([LHC^{+}18]\). However, these schemes suffer from two fundamental drawbacks.

Firstly, it is hard or even impossible to choose an optimal metric for the network performance based on the tensor level quantization error. In particular, even for the same task, similar architectures may favor different objectives. Secondly, the noise in earlier layers might be amplified by successive layers, creating a dependency between quantization errors of different layers. This cross-layer dependency makes it necessary to jointly optimize the quantization parameters across all network layers; however, current methods optimize them separately for each layer. Fig. 3.1 plots the challenging loss surface of this optimization process.

Below, we outline the main contributions of the present work along with the organization of the remaining sections.

- First we consider current layer-by-layer quantization methods where the quantization step size within each layer is optimized to accommodate the dynamic range of the tensor while keeping it small enough to minimize quantization noise. Although these methods optimize the quantization step size of each layer independently of the other layers, we observe strong interactions between the layers, explaining their suboptimal performance at the network level.

- Accordingly, we consider network quantization as a multivariate optimization problem where the layer quantization step sizes are jointly optimized to minimize the cross-entropy loss. We observe that layer-by-layer quantization identifies solutions in a small region around the optimum, where degradation is quadratic in the distance from it. We provide analytical justification as well as empirical evidence showing this effect.

- Finally, we propose to combine layer-by-layer quantization with multivariate quadratic optimization. Our method is shown to significantly outperform state-of-the-art methods on two different challenging tasks and six DNN architectures.
Figure 3.2: Visualization of the loss surface as a function of quantization ranges of two subsequent layers of ResNet18. At higher bit-width, with more fine-grained quantization Fig. 3.2c, the interactions between layers are relatively small, making layer-wise optimization possible. On the other hand, at lower bit-width Fig. 3.2a, with an increase of the quantization noise, the interactions between two layers become tangible. Due to the high coupling, the per layer (per axis) optimization results will entirely depend on the initial point and potentially sub-optimal.

### 3.3 Loss landscape of quantized DNNs

In this section, we introduce the notion of separability of the loss function. We study the separability and the curvature of the loss function and show how quantization of DNNs affect these properties. Finally, we show that during aggressive quantization, the loss function becomes highly non-separable with steep curvature, which is unfavorable for existing post-training quantization methods. Our method addresses these properties and makes post-training quantization possible at low bit quantization.

We focus on symmetric uniform quantization with a fixed number of bits $M$ for all layers and quantization step size $\Delta$ that maps a value $x \in \mathbb{R}$ into a discrete representation,

$$Q_{\Delta, M}(x) = \begin{cases} 
-2^{M-1} \Delta & x < -2^{M-1} \Delta \\
\left\lfloor \frac{x}{\Delta} \right\rfloor \Delta & |x| \leq 2^{M+1} \Delta \\
+2^{M-1} \Delta & x > +2^{M-1} \Delta.
\end{cases} \quad (3.1)
$$

By constraining the range of $x$ to $[-c, c]$, the connection between $c$ and $\Delta$ is given by:

$$\Delta = \frac{2c}{2^{M-1}}. \quad (3.2)$$

In the case of activations, we limit ourselves to the ReLU function, which allows us to choose a quantization range of $[0, c]$. In such cases, the quantization step $\Delta$ is given by:

$$\Delta = \frac{c}{2^{M-1}}. \quad (3.3)$$

30
3.3.1 Separable optimization

Suppose the loss function of the network $L$ depends on a certain set of variables (weights, activations, etc.), which we denote by a vector $\mathbf{v}$. We would like to measure the effect of adding quantization noise to this set of vectors. In the following we show that for sufficiently small quantization noise, we can treat it as an additive noise vector $\epsilon$, allowing coordinate-wise optimization. However, when quantization noise is increased, the degradation in one layer is associated with other layers, calling for more laborious non-separable optimization techniques. Since the quantization is emulated with an additive noise, the loss is smooth and thus can be expanded to Taylor series:

$$\Delta L = L(\mathbf{v} + \epsilon) - L(\mathbf{v}) = \frac{\partial L}{\partial \mathbf{v}}^\top \epsilon + \epsilon^\top \frac{\partial^2 L}{\partial \mathbf{v}^2} \epsilon + O(\|\epsilon\|^3).$$

(3.4)

When the quantization error $\epsilon$ is sufficiently small, higher-order terms can be neglected so that degradation $\Delta L$ can be approximated as a sum of the quadratic functions,

$$\Delta L = \frac{\partial L}{\partial \mathbf{v}}^\top \epsilon + \epsilon^\top \frac{\partial^2 L}{\partial \mathbf{v}^2} \epsilon = \sum_i^n \frac{\partial L}{\partial v_i} \cdot \epsilon_i + \sum_{i}^{n} \sum_{j}^{n} \frac{\partial^2 L}{\partial v_i \partial v_j} \epsilon_i \cdot \epsilon_j.$$

(3.5)

One can see from Eq. (3.7) that when quantization error $\|\epsilon\|^2$ is sufficiently small, the overall degradation $\Delta L$ can be approximated as a sum of $N$ independent separable degradation processes as follows:

$$\Delta L \approx \sum_{i}^{n} \frac{\partial L}{\partial v_i} \cdot \epsilon_i.$$

(3.6)

On the other hand, when $\|\epsilon\|^2$ is larger, one needs to take into account the interactions between different layers, corresponding to the second term in Eq. (3.7) as follows:

$$QIT = \sum_i^n \sum_{j}^{n} \frac{\partial^2 L}{\partial v_i \partial v_j} \epsilon_i \cdot \epsilon_j,$$

(3.7)

where QIT refers to the quantization interaction term.

One can see that at fine-grained quantization when $\|\epsilon\|^2$ is small the overall degradation, $\Delta L$, is additively separable and can be layer-wise optimized. However, for more aggressive quantization, the layer-wise optimization without taking into account interactions between layers (QIT) will lead to suboptimal results.

In Fig. 3.2 we provide a visualization of these interactions at 2, 3 and 4 bitwidth representations.
3.3.2 Curvature

We now analyze how the steepness of the curvature of the loss function with respect to the quantization step changes as the quantization error increases. We will show that at aggressive quantization, the curvature of the loss becomes steep, which is unfavorable for the methods that aim to minimize quantization error on the tensor level.

We start by defining a measure to quantify curvature of the loss function with respect to quantization step size $\Delta$. Given a quantized neural network, we denote $\mathcal{L}(\Delta_1, \Delta_2, \ldots, \Delta_n)$ the loss with respect to quantization step size $\Delta_i$ of each individual layer. Since $\mathcal{L}$ is twice differentiable with respect to $\Delta_i$ we can calculate the Hessian matrix:

$$H[\mathcal{L}]_{ij} = \frac{\partial^2 \mathcal{L}}{\partial \Delta_i \partial \Delta_j}.$$  

(3.10)

To quantify the curvature, we use Gaussian curvature $K$, which is given by:

$$K[\mathcal{L}](\Delta) = \frac{\det (H[\mathcal{L}](\Delta))}{(\|\nabla \mathcal{L}(\Delta)\|_2^2 + 1)^2},$$  

(3.11)

where $\Delta = (\Delta_1, \Delta_2, \ldots, \Delta_n)$. We calculated the Gaussian curvature at the point that minimizes the $L_2$ norm of the quantization error and acquired the following values:

$$K[\mathcal{L}_{4 \text{ bit}}](\Delta) = 6.7 \cdot 10^{-25}$$  

(3.12)

$$K[\mathcal{L}_{2 \text{ bit}}](\Delta) = 0.58,$$  

(3.13)

This means that the flat surface for 4 bits, shown in Fig. 3.2, is a generic property.
Figure 3.4: The $L_p$ norm of the quantization error generated by optimizing for different values of $p$ according to Eq. (3.15). For different $L_p$ metrics, the optimal quantization step is different.

of the fine-grained quantization loss and not of the specific layer choice. Similarly, we conclude that coarser quantization generally has steeper curvature than more fine-grained quantization.

Moreover, the Hessian matrix provides additional information regarding the coupling between different layers. As could be expected, adjacent off-diagonal terms have higher values than distant elements, corresponding to higher dependencies between clipping parameters of adjacent layers (the Hessian matrix is presented in Fig. 3.5 in the Appendix).

When the curvature is steep, even small changes in quantization step size may change the results drastically. To justify this hypothesis experimentally, in Fig. 3.3, we evaluate the accuracy of ResNet-50 for five different quantization steps. We choose quantization steps which minimize the $L_p$ norm of the quantization error for different values of $p$.

While at 4-bit quantization, the accuracy is almost not affected by small changes in the quantization step size, at 2-bit quantization, the same changes shift the accuracy by more than 20%. Moreover, the best accuracy is obtained with a quantization step that minimizes $L_{3.5}$ and not the MSE, which corresponds to $L_2$ norm minimization.
Figure 3.5: Absolute value of the Hessian matrix of the loss function with respect to quantization steps calculated over 15 layers of ResNet-18. Higher values at a diagonal of the Hessian at 2-bit quantization suggest that the minimum is sharper than at 4 bits. Non-diagonal elements provide an indication of the coupling between parameters of different layers: closer layers generally exhibit stronger interactions.

3.3.3 Hessian of the loss function

To estimate dependencies between clipping parameters of different layers, we analyze the structure of the Hessian matrix of the loss function. The Hessian matrix contains the second-order partial derivatives of the loss $\mathcal{L}(\Delta)$, where $b\Delta$ is a vector of quantization steps:

$$H[\mathcal{L}]_{ij} = \frac{\partial^2 \mathcal{L}}{\partial \Delta_i \partial \Delta_j}. \quad (3.14)$$

In the case of separable functions, the Hessian is a diagonal matrix. This means that the magnitude of the off-diagonal elements can be used as a measure of separability. In Fig. 3.5 we show the Hessian matrix of the loss function under quantization of 4 and 2 bits. As expected, higher dependencies between quantization steps emerge under more aggressive quantization.
3.4 Loss Aware Post-training Quantization

In the previous section, we showed that the loss function $\mathcal{L}(\Delta)$, with respect to the quantization step size $\Delta$, has a complex, non-separable landscape that is hard to optimize. Now we suggest a method to overcome this difficulty. Our optimization process involves three consecutive steps.

In the first phase, we find the quantization step $\Delta_p$ that minimizes the $L_p$ norm of the quantization error of the individual layers for several different values of $p$. Then, we perform quadratic interpolation to approximate an optimum of the loss with respect to $p$. Finally, we jointly optimize the parameters of all layers acquired on the previous step by applying a gradient-free optimization method. The pseudo-code of the whole algorithm is presented in Algorithm 3.1. Fig. 3.7 provides the algorithm visualization.

3.4.1 Layer-wise optimization

Our method starts by minimizing the $L_p$ norm of the quantization error of weights and activations in each layer with respect to clipping values:

$$ e_p(\Delta) = \left( \| Q_\Delta(X) - X \|^p \right)^{1/p}. \quad (3.15) $$

Given a real number $p > 0$, the set of optimal quantization steps $\Delta_p = \{ \Delta_{1p}, \Delta_{2p}, \ldots, \Delta_{np} \}$, according to Eq. (3.15), minimizes the quantization error within each layer. In addition, optimizing the quantization error allows us to get $\Delta_p$ in the vicinity of the optimum $\Delta^*$. Different values of $p$ result in different quantization step sizes $\Delta_p$, which are optimal under the $L_p$ metric due to the trade-off between clipping and quantization error (Fig. 3.4).
Figure 3.7: Intuition about the LAPQ algorithm in two dimensions. The visualization shows the loss as a function of quantization ranges of two layers on synthetic data. Yellow dots correspond to the quantization step size \( \{ \Delta_p \} \), which minimizes the \( L_p \) norm of the quantization error. \( \{ \Delta_p \} \) build a trajectory in the vicinity of the minimum \( \Delta^* \). The optimal quantization step size on that trajectory \( \Delta^*_p \) is used as a starting point for a joint optimization algorithm (Powell’s). Vectors \( \{d_1, d_2, d_3\} \) demonstrate the first iteration of Powell’s method that approaches the global minimum \( \Delta^* \).

### 3.4.2 Quadratic approximation

Assuming a quantization step size \( \Delta \) in the vicinity of the optimal quantization step \( \Delta^* \), the loss function can be approximated with a Taylor series as follows:

\[
L(\Delta) - L(\Delta^*) = (\Delta^* - \Delta)^\top \nabla L(\Delta^*) + \frac{1}{2} (\Delta^* - \Delta)^\top H(\Delta^*)(\Delta^* - \Delta) + \mathcal{O}(\|\Delta^* - \Delta\|^3),
\]

where \( H(\Delta^*) \) is the Hessian matrix with respect to \( \Delta \). Since \( \Delta^* \) is a minimum, the first derivative vanishes and we acquire a quadratic approximation of \( L \),

\[
\Delta L = L(\Delta) - L(\Delta^*) \approx \frac{1}{2} (\Delta^* - \Delta)^\top H(\Delta^*)(\Delta^* - \Delta) \quad (3.19)
\]

Our method exploits this quadratic property for optimization. Fig. 3.6(a) demonstrates empirical evidence of such a quadratic relationship for ResNet-18 around the optimal quantization step \( \Delta^* \) obtained by our method. First, we sample a few data points \( \{ \Delta_p \} \) to get number of samples of \( L(\Delta) \) (orange points in Fig. 3.7). Empirically we found that 10 points are enough for a good approximation as can be seen in Fig. 3.6.
Then, we use the prior quadratic assumption to approximate the minimum of the $\mathcal{L}$ on that trajectory by fitting a quadratic function $f(p)$ to the sampled $\Delta_p$. Finally, we minimize $f(p)$ and use the optimal quantization step size $\Delta_p^*$ as a starting point for a gradient-free joint optimization algorithm, such as Powell’s method, to minimize the loss and find $\Delta^*$.

### 3.4.3 Joint optimization

By minimizing both the quantization error and the loss using quadratic interpolation, we get a better approximation of the global minimum $\Delta^*$. Due to steep curvature of the minimum, however, for a low bitwidth quantization, even a small error in the value of $\Delta$ leads to performance degradation. Thus, we use a gradient-free joint optimization, specifically a Powell’s method, to further optimize $\Delta_p^*$.

---

**Algorithm 3.1 LAPQ**

1. **Layer-wise optimization:**
   
   for $p = \{p_1, p_2, \ldots, p_k\}$ do
   
   $bC_p \leftarrow \arg\min_C \left(\|Q_{\Delta,c}(x) - x\|^p\right)^{1/p}$ for every layer
   
   end for

2. **Quadratic approximation:**
   
   $\Delta_p^* \leftarrow \arg\min_{\Delta_p} \mathcal{L}(\Delta_p)$ using quadratic interpolation of $\Delta_p$

3. **Joint optimization (Powell’s):**
   
   Starting point $t_0 \leftarrow \Delta_p^*$.

   Initial direction vector $D = \{d_1, d_2, \ldots, d_N\}$.

   while not converged do
   
   for $k = 1 \ldots N$ do
   
   $\lambda_k \leftarrow \arg\min_\lambda \mathcal{L}(t_{k-1} + \lambda d_k)$

   $t_k \leftarrow t_{k-1} + \lambda d_k$

   end for

   for $j = 1 \ldots N - 1$ do
   
   $d_j \leftarrow d_{j+1}$

   end for

   $d_N \leftarrow t_N - t_0$

   $\lambda_N \leftarrow \arg\min_\lambda \mathcal{L}(t_N + \lambda d_N)$

   $t_0 \leftarrow t_0 + \lambda_N d_N$

   end while

   $\Delta^* \leftarrow t_0$

   return $\Delta^*$

---

At every iteration, we optimize the set of parameters, initialized by $\Delta_p^*$. Given a set of linear search directions $D = \{d_1, d_2, \ldots, d_N\}$, the new position $\Delta_{t+1}$ is expressed by the linear combination of the search directions as following $\Delta_t + \sum_i \lambda_id_i$. The new displacement vector $\sum_i \lambda_id_i$ becomes part of the search directions set, and the search vector, which contributed most to the new direction, is deleted from the search directions set. In practice, in our experiments it convergence in a few hundreds of
iterations. For further details, see Algorithm 3.1.
3.5 Experimental Results

In this section we conduct extensive evaluations and offer a comparison to prior art of the proposed method on two challenging benchmarks, image classification on ImageNet and a recommendation system on NCF-1B. In addition, we examine the impact of each part of the proposed method on the final accuracy. In all experiments we first calibrate the optimal clipping values on a small held-back calibration set using our method and then evaluate the validation set.

3.5.1 ImageNet

We evaluate our method on several CNN architectures on ImageNet. We select a calibration set of 512 random images for the optimization step. The size of the calibration set defines the trade-off between generalization and runtime (the analysis is given in Section 3.5.3). Following the convention [BLC$^+$18a, YSX$^+$19], we do not quantize the first and last layers.

Many successful methods of post-training quantization perform finer parameter assignment, such as group-wise [MKD$^+$17], channel-wise [BNHS18], or filter-wise [CKYK19] quantization, requiring special hardware support and additional computational resources. Finer parameter assignment appears to provide definite improvement, independently of the underlying methods used, but requires more complex and resource-demanding hardware. In contrast with those approaches, our method performs layer-wise quantization, which is simple to implement on any hardware that supports low precision integer operations. Thus, we do not include the methods with finer assignment in our comparison study. We apply bias correction, as proposed by [BNHS18], on top of the proposed method. In Tables 3.1, 3.2 and 3.6 we compare our method with several other layer-wise quantization methods, as well as the minimal MSE baseline. In most cases, our method significantly outperforms all the competing methods, showing acceptable performance even for 4-bit quantization.

3.5.2 NCF-1B

In addition to the vision models, we evaluated our method on a recommendation system task, specifically on a Neural Collaborative Filtering (NCF) [XQL19] model. We use mlperf$^1$ implementation to train the model on the MovieLens-1B dataset. Similarly to the ImageNet, the calibration set of 50k random user/item pairs is significantly smaller than both the training and validation sets.

In Table 3.3 we present results for the NCF-1B model compared to the MMSE method. Even at 8-bit quantization, NCF-1B suffers from significant degradation when using the naive MMSE method. In contrast, LAPQ achieves near baseline accuracy with 0.5% degradation from FP32 results.

---

$^1$https://github.com/mlperf/training/tree/master/recommendation/pytorch
3.5.3 Ablation study

Initialization The proposed method comprises of two steps: layer-wise optimization and quadratic approximation as the initialization for the joint optimization. In Table 3.4, we show the results for ResNet-18 under different initializations and their results after adding the joint optimization. LAPQ suggest a better initialization for the joint optimization.

Bias correction Prior research [BNHS18] has shown that CNNs are sensitive to quantization bias. To address this issue, we perform bias correction of the weights as proposed by [BNHS18], which can easily be combined with LAPQ, in all our CNN experiments. In Table 3.5 we show the effect of adding bias correction to the proposed method and compare it with MMSE. We see that bias correction is especially important in compact models, such as MobileNet.

Calibration set size Calibration set size reflects the balance between the running time and the generalization. To determine the required size, we ran the proposed method on ResNet-18 for various calibration set sizes and different bitwidths. As shown in Fig. 3.8, a calibration set size of 512 is a good choice to balance this trade-off.
Table 3.1: Comparison with other methods on ResNet-18, ResNet-50, and MobileNet V2. MMSE refers to minimization of MSE. W refers to the weights bitwidth and A to the activations bitwidth.

<table>
<thead>
<tr>
<th>Model</th>
<th>W/A</th>
<th>Method</th>
<th>Accuracy(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>32 / 32</td>
<td>FP32</td>
<td>69.7</td>
</tr>
<tr>
<td></td>
<td>8 / 8</td>
<td>LAPQ (Ours)</td>
<td>69.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DFQ [NBBW19]</td>
<td>69.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMSE</td>
<td>69.0</td>
</tr>
<tr>
<td></td>
<td>8 / 4</td>
<td>LAPQ (Ours)</td>
<td>68.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DUAL [CKYK19]</td>
<td>68.38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACIQ [BNHS18]</td>
<td>65.528</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMSE</td>
<td>68.0</td>
</tr>
<tr>
<td></td>
<td>8 / 3</td>
<td>LAPQ (Ours)</td>
<td>66.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACIQ [BNHS18]</td>
<td>52.476</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMSE</td>
<td>63.3</td>
</tr>
<tr>
<td></td>
<td>4 / 4</td>
<td>LAPQ (Ours)</td>
<td>60.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACIQ [BNHS18]</td>
<td>4.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KLD [Mig17]</td>
<td>31.937</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMSE</td>
<td>43.6</td>
</tr>
<tr>
<td>ResNet-50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>32 / 32</td>
<td>FP32</td>
<td>76.1</td>
</tr>
<tr>
<td></td>
<td>8 / 4</td>
<td>LAPQ (Ours)</td>
<td>74.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DUAL [CKYK19]</td>
<td>73.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACIQ [BNHS18]</td>
<td>68.92</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMSE</td>
<td>74.0</td>
</tr>
<tr>
<td></td>
<td>8 / 3</td>
<td>LAPQ (Ours)</td>
<td>70.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACIQ [BNHS18]</td>
<td>51.858</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMSE</td>
<td>66.3</td>
</tr>
<tr>
<td></td>
<td>4 / 4</td>
<td>LAPQ (Ours)</td>
<td>70.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACIQ [BNHS18]</td>
<td>3.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KLD [Mig17]</td>
<td>46.19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMSE</td>
<td>36.4</td>
</tr>
<tr>
<td>MobileNet v2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>32 / 32</td>
<td>FP32</td>
<td>71.8</td>
</tr>
<tr>
<td></td>
<td>8 / 8</td>
<td>LAPQ (Ours)</td>
<td>71.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DFQ [NBBW19]</td>
<td>71.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMSE</td>
<td>71.6</td>
</tr>
<tr>
<td></td>
<td>4 / 32</td>
<td>LAPQ (Ours)</td>
<td>59.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMSE</td>
<td>47.6</td>
</tr>
<tr>
<td></td>
<td>4 / 4</td>
<td>LAPQ (Ours)</td>
<td>52.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMSE</td>
<td>26.0</td>
</tr>
</tbody>
</table>
Table 3.2: Comparison with other methods on ResNet-101 and Inception-V3. MMSE refers to minimization of MSE. W refers to the weights bitwidth and A to the activations bitwidth.

<table>
<thead>
<tr>
<th>Model</th>
<th>W/A</th>
<th>Method</th>
<th>Accuracy(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-101</td>
<td>32 / 32</td>
<td>FP32</td>
<td>77.3</td>
</tr>
<tr>
<td></td>
<td>8 / 4</td>
<td>LAPQ (Ours)</td>
<td>73.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DUAL [CKYK19]</td>
<td>74.26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACIQ [BNHS18]</td>
<td>66.966</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMSE</td>
<td>72.0</td>
</tr>
<tr>
<td></td>
<td>8 / 3</td>
<td>LAPQ (Ours)</td>
<td>65.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACIQ [BNHS18]</td>
<td>41.46</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMSE</td>
<td>56.7</td>
</tr>
<tr>
<td></td>
<td>4 / 4</td>
<td>LAPQ (Ours)</td>
<td>59.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACIQ [BNHS18]</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KLD [Mig17]</td>
<td>49.948</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMSE</td>
<td>9.8</td>
</tr>
<tr>
<td>Inception-V3</td>
<td>32 / 32</td>
<td>FP32</td>
<td>77.2</td>
</tr>
<tr>
<td></td>
<td>8 / 4</td>
<td>LAPQ (Ours)</td>
<td>75.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DUAL [CKYK19]</td>
<td>73.06</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACIQ [BNHS18]</td>
<td>66.42</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMSE</td>
<td>74.3</td>
</tr>
<tr>
<td></td>
<td>8 / 3</td>
<td>LAPQ (Ours)</td>
<td>64.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACIQ [BNHS18]</td>
<td>31.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMSE</td>
<td>54.1</td>
</tr>
<tr>
<td></td>
<td>4 / 4</td>
<td>LAPQ (Ours)</td>
<td>38.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACIQ [BNHS18]</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KLD [Mig17]</td>
<td>1.84</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMSE</td>
<td>2.2</td>
</tr>
</tbody>
</table>

Table 3.3: Hit rate of NCF-1B applying our method, LAPQ, and MMSE. W refers to the weights bitwidth and A to the activations bitwidth.

<table>
<thead>
<tr>
<th>Model</th>
<th>W/A</th>
<th>Method</th>
<th>Hit rate(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCF 1B</td>
<td>32/32</td>
<td>FP32</td>
<td>51.5</td>
</tr>
<tr>
<td></td>
<td>32/8</td>
<td>LAPQ (Ours)</td>
<td>51.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMSE</td>
<td>51.1</td>
</tr>
<tr>
<td></td>
<td>8/32</td>
<td>LAPQ (Ours)</td>
<td>51.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMSE</td>
<td>33.4</td>
</tr>
<tr>
<td></td>
<td>8/8</td>
<td>LAPQ (Ours)</td>
<td>51.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMSE</td>
<td>33.5</td>
</tr>
</tbody>
</table>
Table 3.4: Accuracy of ResNet-18 with different initializations for the joint optimization. LW refers to only applying layer-wise optimization for $p = 2$. LW + QA refers to the proposed initialization that includes layer-wise optimization and quadratic approximation. W refers to the weights bitwidth and A to the activations bitwidth.

<table>
<thead>
<tr>
<th>W / A</th>
<th>Method</th>
<th>Accuracy (%)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Initial</td>
<td>Joint</td>
<td></td>
</tr>
<tr>
<td>4 / 4</td>
<td>Random</td>
<td>0.1</td>
<td>7.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LW</td>
<td>43.6</td>
<td>57.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LW + QA</td>
<td>54.1</td>
<td>60.3</td>
<td></td>
</tr>
<tr>
<td>32 / 2</td>
<td>Random</td>
<td>0.1</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LW</td>
<td>33</td>
<td>50.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LW + QA</td>
<td>48.1</td>
<td>50.7</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.5: Effect of applying bias correction on top of LAPQ for ResNet-18, ResNet-50 and MobileNet-V2. LAPQ significantly outperforms naive minimization of the mean square error (MSE). Notice the importance of bias correction on MobileNet-V2. W refers to the weights bitwidth and A to the activations bitwidth.

<table>
<thead>
<tr>
<th>W</th>
<th>A</th>
<th>ResNet-18</th>
<th>ResNet-50</th>
<th>MobileNet-V2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LAPQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>4</td>
<td>68.8%</td>
<td>74.8%</td>
<td>65.1%</td>
</tr>
<tr>
<td>32</td>
<td>2</td>
<td>51.6%</td>
<td>54.2%</td>
<td>1.5%</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>62.6%</td>
<td>69.9%</td>
<td>29.4%</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>58.5%</td>
<td>66.6%</td>
<td>21.3%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LAPQ + bias correction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>63.3%</td>
<td>71.8%</td>
<td>59.4%</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>60.3%</td>
<td>70.0%</td>
<td>52.2%</td>
</tr>
<tr>
<td>FP32</td>
<td>69.7%</td>
<td>76.1%</td>
<td>71.8%</td>
<td></td>
</tr>
</tbody>
</table>
Table 3.6: Comparison with other methods.

<table>
<thead>
<tr>
<th>Model</th>
<th>W/A</th>
<th>Method</th>
<th>Accuracy(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>32/32 FP32</td>
<td>LAPQ (Ours)</td>
<td>69.7</td>
</tr>
<tr>
<td></td>
<td>8/2</td>
<td>ACIQ [BNHS18]</td>
<td>51.6</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>32/32 FP32</td>
<td>LAPQ (Ours)</td>
<td>76.1</td>
</tr>
<tr>
<td></td>
<td>8/2</td>
<td>ACIQ [BNHS18]</td>
<td>54.2</td>
</tr>
<tr>
<td></td>
<td>4/32</td>
<td>OCS [ZGB+19]</td>
<td>71.8</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>32/32 FP32</td>
<td>LAPQ (Ours)</td>
<td>77.3</td>
</tr>
<tr>
<td></td>
<td>8/2</td>
<td>ACIQ [BNHS18]</td>
<td>29.8</td>
</tr>
<tr>
<td></td>
<td>4/32</td>
<td>MMSE</td>
<td>66.5</td>
</tr>
</tbody>
</table>
3.6 Discussion

We have analyzed the loss function of quantized neural networks. At low precision, the function is non-separable, with steep curvature, which is unfavorable for existing post-training quantization methods. Accordingly, we have introduced Loss Aware Post-training Quantization (LAPQ), which jointly optimizes all quantization parameters by minimizing the loss function directly. We have shown that our method outperforms current post-training quantization methods. Also, our method does not require special hardware support such as channel-wise or filter-wise quantization. In some models, LAPQ is the first to show compatible performance in 4-bit post-training quantization regime with layer-wise quantization, almost achieving a the full-precision baseline accuracy.
Chapter 4

Towards learning of filter-level heterogeneous compression of CNNs

4.1 Motivation

Recently, deep learning has become a de facto standard in machine learning with convolutional neural networks (CNNs) demonstrating spectacular success on a wide variety of tasks. However, CNNs are typically very demanding computationally at inference time. One of the ways to alleviate this burden on certain hardware platforms is quantization relying on the use of low-precision arithmetic representation for the weights and the activations. Another popular method is the pruning of the number of filters in each layer. While mainstream deep learning methods train the neural networks weights while keeping the network architecture fixed, the emerging neural architecture search (NAS) techniques make the latter also amenable to training. In this chapter, we formulate optimal arithmetic bit length allocation and neural network pruning as a NAS problem, searching for the configurations satisfying a computational complexity budget while maximizing the accuracy. We use a differentiable search method based on the continuous relaxation of the search space proposed by [LSY19]. We show, by grid search, that heterogeneous quantized networks suffer from a high variance which renders the benefit of the search questionable. For pruning, improvement over homogeneous cases is possible, but it is still challenging to find those configurations with the proposed method.
4.2 Introduction

Convolutional neural networks (CNNs) have become a main solution for computer vision tasks. However, high computation requirements complicate their usage in low-power systems.

The method presented in this chapter is designed to tackle two aspects of complexity reduction: quantization and pruning. Some recent works demonstrated that using 16—or even 8—bit representations do not harm accuracy of NNs [GAGN15, JKC+18a, LHC+18]. To reduce both runtime and power consumption, researches investigated further reduction of bitwidth, up to a single bit [HCS+16, BBL+18, DCLM19], which is impossible with naive techniques.

Alternatively, one could use fewer convolutional filters by proportionally reducing their number [SHZ+18a] or by pruning insignificant ones [LDS90, LKD+17]. Recently, slimmable networks [YYX+19] offered a method to simultaneously train multiple instances of a CNN with different filter count.

Both methods compress the network with some parameter $\alpha$ representing the bitwidth or the percentage of filters pruned. Homogeneous configurations use same value of $\alpha$ along the network, e.g. same bitwidth of parameters in each layer. Many quantization works employ simple heterogeneous configurations with different bitwidth in first or last layer [ZWN+16, ZYYH18, HHS18, MEA+18, CCW+18]. Some works studied a layer-wise quantization granularity [LUW17, LTA18]. Heterogeneous configurations are often found in pruning too.

Recent works studied quantization with filter-wise quantization granularity [WWZ+18, YEP+18, CMZ+18, LLKJ19, GZM+19] and pruning [LMZ+19, YH19a] using NAS techniques. In this thesis we study the opportunities of compression of the network with filter-wise granularity.
4.3 Method

Differentiable search method  For CNN with \( L \) convolutional layers with \( C_\ell \) filters in layer \( \ell = 1, \ldots, L \), let \( T_\ell \) denote the set of compression operations that can be applied to one filter. Our goal is to find an optimal assignment of the said operations to each filter.

To do that, we aim to learn a probability of each operation to be chosen, similarly to [LSY19]. Referring to the operations in \( T_\ell \) simply by their index, \( i = 1, \ldots, |T_\ell| \), we assign to each operation \( i \) in layer \( \ell \) a parameter \( \alpha_{\ell i} \), and denote by \( \hat{\alpha}_{\ell i} = f(\alpha_{\ell i}) \) the probability to choose this particular operation. We also denote by \( \alpha_\ell = (\alpha_{\ell 1}, \ldots, \alpha_{\ell |T_\ell|})^T \) the vector of parameters in a given layer \( \ell \), and by the pseudomatrix \( \alpha = (\alpha_1, \ldots, \alpha_L) \) the corresponding parameters of the entire network.

To reduce the search space size, instead of assigning operations to each filter independently we do it on layer level. Let \( a_{\ell i} \) denote the number of filters in the layer \( \ell \) to which the operation \( i \) is applied. We refer to the vector \( a_\ell = (a_{\ell 1}, \ldots, a_{\ell |T_\ell|})^T \) with the elements summing to \( C_\ell \) defining the choice of the operations in layer \( \ell \) as to the configuration of the layer. We denote the configuration of the entire network by the pseudo-matrix \( a \).

Let \( A_\ell \) be a random vector sampling which yields a specific configuration \( a_\ell \) for layer \( \ell \). The probability of the network configuration \( a \) is given by

\[
p(a|\alpha) = \prod_{\ell=1}^L \Pr(A_\ell = a_\ell).
\]

(4.1)

Note that the latter probability depends on the \( \hat{\alpha}_\ell \)'s, which, in turn, depend on the \( \alpha_\ell \)'s. The neural network is thus fully defined by the configuration \( a \) and the weights \( \omega \).

Let \( \mathcal{L}(a;\omega) \) denote the loss of the particular configuration \( a \) with the weights \( \omega \). The expected loss over all network configurations with the same weights is given by

\[
J(\alpha;\omega) = \mathbb{E}_a \mathcal{L}(a;\omega) = \sum_a p(a|\alpha) \mathcal{L}(a;\omega),
\]

(4.2)

where the sum is taken over all possible configurations. The goal of the search is to minimize the latter loss over \( \alpha \) and \( \omega \), which is carried out using gradient steps.

Note that compared to the regular neural network training (optimization over \( \omega \) with a single configuration), the number of additional optimization variables (\( \alpha \)) remains relatively modest. In sharp contrast, the number of configurations required to compute \( J \) is exponentially large. For this reason, we approximate the gradient \( g = \nabla_{\alpha} J \) by sampling a subset \( S \) of possible configurations:

\[
g_{\alpha_{\ell i}} \approx \hat{g}_{\alpha_{\ell i}} = \frac{1}{|S|} \sum_{a \in S} \mathcal{L}(a;\omega) \cdot (a_{\ell i} - C_\ell \cdot \hat{\alpha}_{\ell i}).
\]

(4.3)
The sample size $|S|$ governs the tradeoff between the complexity of the training and the estimator variance. The gradient with respect to the weights $\omega$ is computed as usual.

**Loss**  We use bit operations (BOPs) [BSZ+18] as a complexity metric in the case of quantization. BOPs refers to the number of bit operations needed to perform inference. Since the bitwidth of operands might be different, we extend the definition of [BSZ+18] to this case. The exact derivation of the BOPs metric is provided ?? . Neither FLOPs nor BOPs predict the runtime of the network [TCP+18, CZH19, LZPF19], but can still be used as a proxy to the performance.

Let us denote the complexity of layer $\ell$ by $B_\ell(a)$; the metric is defined in ?? in the Appendix for the quantized CNN case, and simply equals to the MAC count of the layer in all other cases. We define the computational complexity loss as

$$L_{\text{com}}(a) \triangleq \sigma \left( \sum_{\ell=1}^{L} B_\ell(a) \right), \quad (4.4)$$

where $\sigma$ is some increasing function. Note that $L_{\text{com}}(a)$ only depends on the network configuration and penalizes configurations $a$ with high complexity. In particular, $\sigma$ can be a function of the ratio between $a$ arithmetic complexity and the complexity of some target homogeneous configuration, which allows to set a target complexity for the search.

The combined loss $L(a; \omega)$ appearing in (4.3) is a linear combination of the standard loss used to train the network w.r.t. the weights, $L_{\text{acc}}(a; \omega)$, and the complexity loss,

$$L(a; \omega) = L_{\text{acc}}(a; \omega) + \lambda \cdot L_{\text{com}}(a), \quad (4.5)$$

embodying the tradeoff between the network accuracy and complexity.
4.4 Quantized NAS

Quantization was one of the evaluated compression methods. We used ResNet-20 [HZRS16b] on CIFAR-10 [Kri09b]. The network was quantized with NICE [BLC18b], with the operations set $T_\ell$ consisting of tuples $(b_w, b_a)$ of weight and output activation bitwidths, respectively. $A_\ell \sim \text{Multinomial} \left( C_\ell, \left( \hat{\alpha}_{\ell1}, \hat{\alpha}_{\ell2}, \ldots, \hat{\alpha}_{\ell|T_\ell|} \right) \right)$ is multinomial random variable, with the probabilities $\hat{\alpha}_\ell$ obtained from $\alpha$ using softmax.

Sampling a layer configuration $a_\ell = (a_{\ell1}, \ldots, a_{\ell|T_\ell|})$ induces a specific structure over the filters. For $T_\ell = \{t_1, t_2, \ldots, t_{|T_\ell|}\}$, we apply quantization with bitwidth tuple $t_1$ on the first $a_{\ell1}$ filters, $t_2$ on the next $a_{\ell2}$ filters and so on.

In our experiments, we selected the set $T_\ell = \{(2, 2), (2, 4), (3, 3), (8, 8)\}$ for all the layers. A few configurations were trained multiple times under the same conditions. We conclude that though the search yields well-performing configurations (Fig. 4.2a), the variance of the accuracy is high, as shown in Fig. 4.1a. Thus, is impossible to establish whether the configurations would be good in a different realization.
4.5 Slimmable NAS

Another compression method we considered is a reduction of number of filters in convolutional layers. In particular, we used slimmable networks framework [YYX+19, YH19b], in which networks with the same architecture but different amount of filters are trained simultaneously with the same weights. The operations set $T_\ell = \{1, \ldots, C_\ell\}$ represents the number of filters in a layer $\ell$. We set $A_\ell \sim \text{Binomial}(C_\ell - 1, \hat{\alpha}_\ell)$ as binomial a random variable, and use a sigmoid normalization of the distribution parameters,

$$\hat{\alpha}_\ell = \frac{\exp\{\alpha_\ell\}}{\exp\{\alpha_\ell\} + 1}$$  \hspace{1cm} (4.6)

Sampling a configuration from $A_\ell$ determines the number of filters in the layer.

Similarly to Section 4.4, we explored the search space by evaluating ResNet-20 configurations. As shown in Fig. 4.1b, the variance is relatively low, though it is still higher for heterogeneous configurations. Points with statistically significant improvement over homogeneous configuration were also found.

Figure 4.1: Results of grid search in both cases. Blue line connects homogeneous configurations, colored points are heterogeneous configurations. Error bars are for 0.6827 confidence interval.

**Basic search method** At each iteration, we sample a set of configurations $S_k$ from current distribution $\hat{\alpha}^k$ for gradient estimation. To improve the loss evaluation we duplicate the current network weights and fine-tune each configuration $a \in S_k$ for 5 epochs.

We define the expected configuration $A^{\hat{\alpha}}$ such that $A^{\hat{\alpha}}_\ell = \text{round}(E[A_\ell])$. The network weights are trained over 5 configurations: 4 homogeneous ones ($\{0.25, 0.5, 0.75, 1.0\}$) and one defined by $A^{\hat{\alpha}}$. Since samples from $\hat{\alpha}^k$ would be close to
the expectation of their distribution, $\omega$ should be a better starting point to train the sampled configurations.

**Resetting the $\omega$** We noticed that after few iterations of the network weights updates on $A \cup A^{\hat{\alpha}_k}$, the validation loss of $A^{\hat{\alpha}_k}$ was high compared to the homogeneous configurations in $A$. We conjectured that the network overfits to the homogeneous configurations which are kept same while $A^{\hat{\alpha}_k}$ changes. To avoid the overfitting, we reinitialize $\omega$ after each iteration.

**Disabling weight-sharing** In addition, the overly short fine-tuning leads to inaccurate configuration evaluation. Thus instead of sharing and fine-tuning $\omega$ we trained each configuration $a$, individually, with individual weights set $\omega_1, 2, \ldots, L$, from scratch.

**Interpolation loss** To achieve the goal of improvement over homogeneous configurations, we tried to compare the heterogeneous configuration cross-entropy with the expected one, by defining the loss as a difference from interpolation of known homogeneous configurations. The results are shown on Fig. 4.2b

Figure 4.2: Results of search in both cases. Each point represents a configuration proposed by search.
4.6 Discussion

in this thesis, we studied the feasibility of using NAS-like algorithms, and in particular differentiable NAS [LSY19], for the reduction of CNN complexity by means of filter-wise quantization and layer-wise pruning. In both cases, we applied our method for ResNet-20 on CIFAR-10, on which it took only 36 GPU-hours to converge.

For filter-wise quantization, after acquiring nominal improvement over the homogeneous baseline, we found out that the variance of heterogeneous configurations is too high to warrant a significant improvement, which we confirmed using partial grid search. Unfortunately, previous studies on bitwidth allocation or architecture-quantization search [WWZ+18, YEP+18, CMZ+18, LLKJ19, GZM+19] did not report the variance of the results, making meaningful comparison impossible. For layer-wise pruning, we obtained more stable results, with the grid search confirming the possibility of improvement over the baseline homogeneous configurations. However, the heterogeneous configurations found by NAS did not significantly outperform the baseline.

We conclude that future work should focus on loss design and better loss estimators, such as Gumbel softmax [JGP16]. Successfully transferring the architecture to a more challenging use case (e.g. ImageNet) remains another important challenge.
Chapter 5

Feature Map Transform Coding for Energy-Efficient CNN Inference

5.1 Motivation

Convolutional neural networks (CNNs) achieve state-of-the-art accuracy in a variety of tasks in computer vision and beyond. One of the major obstacles hindering the ubiquitous use of CNNs for inference on low-power edge devices is their high computational complexity and memory bandwidth requirements. The latter often dominates the energy footprint on modern hardware. In this thesis, we introduce a lossy transform coding approach, inspired by image and video compression, designed to reduce the memory bandwidth due to the storage of intermediate activation calculation results. Our method does not require fine-tuning the network weights and halves the data transfer volumes to the main memory by compressing feature maps, which are highly correlated, with variable length coding. Our method outperforms previous approach in term of the number of bits per value with minor accuracy degradation on ResNet-34 and MobileNetV2. We analyze the performance of our approach on a variety of CNN architectures and demonstrate that FPGA implementation of ResNet-18 with our approach results in a reduction of around 40% in the memory energy footprint, compared to quantized network, with negligible impact on accuracy. When allowing accuracy degradation of up to 2%, the reduction of 60% is achieved.
5.2 Introduction

Deep neural networks have established themselves as the first-choice tool for a wide range of applications. Neural networks have shown phenomenal results in a variety of tasks in a broad range of fields such as computer vision, computational imaging, and image and language processing. Despite deep neural models impressive performance, the computation and computational requirements are substantial for both training and inference phases. So far, this fact has been a major obstacle for the deployment of deep neural models in applications constrained by memory, computational, and energy resources, such as those running on embedded systems.

To alleviate the energy cost, custom hardware for neural network inference, including FPGAs and ASICs, is actively being developed in recent years. In addition to providing better energy efficiency per arithmetic operation, custom hardware offers more flexibility in various strategies to reduce the computational and storage complexity of the model inference, for example by means of quantization [BSZ+18, HCS+18, JKC+18b] and pruning [HMD16, LWK18, TKTH18]. In particular, quantization to very low precision is especially efficient on custom hardware where arbitrary precision arithmetic operations require proportional resources. To prevent accuracy degradation, many approaches have employed training the model with quantization constraints or modifying the network structure.

A recent study [YCES17] has shown that almost 70% of the energy footprint on such hardware is due to data movement to and from the off-chip memory. Amounts of data typically need to be transferred to and from the RAM and back during the forward pass through each layer, since the local memory is too small to store all the feature maps. By reducing the number of bits representing these data, existing quantization techniques reduce the memory bandwidth considerably. However, to the best of our knowledge, none of these methods exploit the high amount of interdependence between the feature maps and spatial locations of the compute activations.

Contributions. In this thesis, we propose a novel scheme based on transform-domain quantization of the neural network activations followed by lossless variable length coding. The method does not require neither backpropogation nor training data except for a one calibration batch. We demonstrate that this approach reduces memory bandwidth by 40% when applied in the post-training regime (i.e., without fine-tuning) with small computational overhead and no accuracy degradation. Relaxing the accuracy requirements increases bandwidth savings to 60%. Moreover, we outperform previous methods in term of number bit per value with minor accuracy degradation. A detailed evaluation of various ingredients and parameters of the proposed method is presented. We also demonstrate a reference hardware implementation that confirms a reduction in memory energy consumption during inference.
Figure 5.1: High-level flow diagram of the encoder-decoder chain. PCA and BN are folded into the convolution weights (denoted by \( \ast \)), resulting in a single convolution (boxed in grey).

Figure 5.2: Vector quantization in 2D case. (a) A pair of correlated channels on a scatter plot. All values in a cell are mapped to the center of the cell; hence, small cells induce small quantization noise. Several bins are empty (red cells); (b) Decorrelation improves utilization since the cells are smaller now; (c) Forcing equal bin size along all dimensions further improves utilization. Instead of restricting both channel dynamic range to be divided into same number of bins, we use uniform bin size along all dimensions. VLC allows to further compress the representation since the channels with smaller dynamic range have are mapped mostly to a few most probable bins.

5.3 Transform-Domain Compression

In what follows, we briefly review the basics of lossy transform coding. For a detailed overview of the subject, we refer the reader to (author?) [Goy01]. Let \( \mathbf{x} = (x_1, \ldots, x_n) \) represent the values of the activations of a NN layer in a block of size \( n = W \times H \times C \) spanning, respectively, the horizontal and the vertical dimensions and the feature channels. Prior to being sent to memory, the activations, \( \mathbf{x} \), are encoded by first undergoing an affine transform, \( \mathbf{y} = \mathbf{T} \mathbf{x} = \mathbf{T}(\mathbf{x} - \mu) \); the transform coefficients are quantized by a scalar quantizer, \( Q_\Delta \), whose strength is controlled by the step size \( \Delta \), and subsequently coded by a lossless variable length coder (VLC). We refer to the length in bits of the resulting code, normalized by \( n \) as to the average rate, \( R_\Delta \).
Figure 5.3: Rate-distortion curves for ResNet-18, ResNet-50, ResNet-101, Inception V3 and MobileNetV2 architectures with 8- (blue) and 4-bit (red) weight quantization. Distortion is evaluated in terms of top-1 accuracy on ImageNet. Dashed lines represent rates obtained by Huffman VLC, while solid lines represent theoretical rates (entropy).

Figure 5.4: Ablation study of the proposed encoder on ResNet-18. Left: rate-distortion curve with different encoder configurations. Theoretical rates are reported; top-1 accuracy is used as the distortion measure. Right: theoretical memory rate in bits per value achieved for different levels of PCA truncation for baseline and 0.5% lower than baseline top-1 accuracy.

decode the activation vector, a variable length decoder (VLD) is applied first, followed by the inverse quantizer and the inverse transform. The resulting decoded activation, \( \hat{x} = T^{-1}Q_{\Delta}^{-1}(Q_{\Delta}(Tx)) \), typically differs from \( x \); the discrepancy is quantified by a distortion, \( D_{\Delta} \). The functional relation between the rate and the distortion is controlled by the quantization strength, \( \Delta \), and is called rate-distortion curve.

Classical rate-distortion analysis in information theory assumes the MSE distortion, \( D = \frac{1}{n}\|x - \hat{x}\|_2^2 \). While in our case the measure of distortion is the impact on the task-specific loss, we adopt the Euclidean distortion for two reasons: firstly, it is well-studied and leads to simple expressions for the quantizer; and, secondly, computing loss requires
Figure 5.5: Analysis of the eigenvalues ratio that are needed to achieve energy ratio, means cumulative sum of the eigenvalues.

Figure 5.6: The influence of the block shape on the top-1 validation accuracy (left) and MSE (right) of ResNet-18 on ImageNet. Our experiments show that for the same size, the most efficient shape is $1 \times 1 \times C$, taking advantage of the correlations across different feature maps at the same spatial location. Theoretical rates are shown.

A crucial observation justifying transform coding is the fact that significant statistical dependence usually exists between the $x_i$ [CAG+16]. We model this fact by asserting that the activations are jointly Gaussian, $x \sim \mathcal{N}(\mu, \Sigma)$, with the covariance matrix $\Sigma$, whose diagonal elements are denoted by $\sigma^2_i$. Statistical dependence corresponds to
Figure 5.7: Top-1 accuracy on ResNet18 ImageNet vs. power consumption of our hardware implementation. Each point represents a different quantization rate.

non-zero off-diagonal elements in $\Sigma$. The affinely transformed $y = T(x - \mu)$ is also Gaussian with the covariance matrix $\Sigma' = T^T \Sigma T$. The distortion is minimized over orthonormal matrices by $T = \Sigma^{-1/2}$ diagonalizing the covariance [Goy01]. The latter is usually referred to as the Karhunen-Loeve transform (KLT) or principal component analysis (PCA). The corresponding minimum distortion is $D^*(R) = \frac{\pi e^2}{6} \det(\Sigma)^{1/n^2} 2^{-2R}$. Since the covariance matrix is symmetric, $T$ is orthonormal, implying $T^{-1} = T^T$.

In Fig. 5.2, a visualization of 2D vector quantization is shown. For correlated channels (Fig. 5.2a), many 2D quantization bins are not used since they contain no values. Linear transformation (Fig. 5.2b) provides improved quantization error for correlated channels by getting rid of those empty bins.

5.3.1 Implementation

In what follows, we describe an implementation of the transform coding scheme at the level of individual CNN layers. The convolutional layer depicted in Fig. 5.1 comprises a bank of convolutions (denoted by $\ast$ in the Figure) followed by batch normalization (BN) that is computed on an incoming input stream. The output of BN is a 3D tensor that is subdivided into 3D blocks to which the transform coding is applied. Each such block is sent to an encoder, where it undergoes PCA, scalar quantization and VLC.
stream at the VLC output has a lower rate than the raw input and is accumulated in the external memory. Once all the output of the layer has been stored in the memory, it can be streamed as the input to the following layer. For that purpose, the inverse process is performed by the decoder: a VLD produces the quantized levels that are scaled back to the transform domain, and an inverse PCA is applied to reconstruct each of the activation blocks. The layer non-linearity is then applied, and the activations are used as an input to the following layer. While the location of the nonlinearity could also precede the encoder, our experiments show that the described scheme performs better.

**Linear transform** We have explored different sizes of blocks for the PCA transform and found $1 \times 1 \times C$ to be optimal (the ablation study is shown in Section 5.4.2). Moreover, this choice allows to optimize the calculation of the transformation: applying same linear transformation to every $1 \times 1 \times C$ block is a convolution with $1 \times 1$ kernel, which can be calculated very efficiently. This allows further optimization: as depicted in Fig. 5.1, the convolution bank of the layer, BN and PCA can be folded [JKC+18b] into a single operation, offering also an advantage in the arithmetic complexity.

The PCA matrix is pre-computed, as its computation requires the expensive eigen-decomposition of the covariance matrix. The covariance matrix is estimated on a small batch of (unlabeled) training or test data and can be updated online. Estimation of the covariance matrix for all layers at once is problematic since quantizing activations in the $l$-th layer alters the input to the $l + 1$-st layer, resulting in a biased estimation of the covariance matrix in the $l + 1$-st layer. To avoid this, we calculate the covariance matrix layer by layer, gradually applying the quantization: at iteration $i$, first $i - 1$ layers perform PCA transformation and only $i$ layer covariance matrix estimation is updated. The PCA matrix is calculated after quantization of the weights is performed, and is itself quantized to 8 bits.

**Quantization** For transformed feature maps we use a uniform quantization, where the dynamic range is determined according to the channel with the highest variance. Since all channels have an equal quantization step, entropy of the low-variance channels is significantly reduced.

**Variable length coding** The theoretical rate associated with a discrete random variable, $Y$ (the output of the quantizer), is given by its entropy $H(X) = -\mathbb{E} \log_2 X = -\sum_i p(x_i) \log_2 p(x_i)$. This quantity constitutes the lower bound on the amount of information required for lossless compression of $Y$. We use Huffman codes, which are a practical variable length coding method [Szp00], achieving the rates bounded by $H(X) \leq R \leq H(X) + 1$ (see Fig. 5.3 for a comparison of the theoretical rates to the ones attained by Huffman codes).
1×1 and grouped convolutions While for regular 3×3 convolutions the computational overhead is small, there are two useful cases in which this is not true: 1×1 and grouped convolutions. For 1×1 convolutions the overhead is higher: the transformation requires as much computation as the convolution itself. Nevertheless, it can still be feasible in the case of energy-efficient computations. In the case of grouped convolutions, it is impossible to fold the transformation inside the convolution. However, in the common case when the grouped convolution is followed by a regular one, we can change the order of operations: we perform BN, activation and transformation before writing to the memory. This way, the inverse transformation can be folded inside the following convolution.
Table 5.1: Comparison with EBPC [CB19]. While EBPC does not affect performance of the network, our method allows better compression by exploiting rate-distortion tradeoff.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Method</th>
<th>Activations (avg. number of bits per value)</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-34</td>
<td>EBPC</td>
<td>3.33</td>
<td>73.3%</td>
</tr>
<tr>
<td></td>
<td>Our method</td>
<td>3.9</td>
<td>72.9%</td>
</tr>
<tr>
<td></td>
<td>Our method</td>
<td><strong>3.11</strong></td>
<td>72.1%</td>
</tr>
<tr>
<td>MobileNetV2</td>
<td>EBPC</td>
<td>3.64</td>
<td>71.7%</td>
</tr>
<tr>
<td></td>
<td>Our method</td>
<td>3.8</td>
<td>71.6%</td>
</tr>
<tr>
<td></td>
<td>Our method</td>
<td><strong>3.25</strong></td>
<td>71.4%</td>
</tr>
</tbody>
</table>

5.4 Experimental Results

We evaluate the proposed framework on common CNN architectures that have achieved high performance on the ImageNet benchmark. The inference contains 2 stages: a calibration stage, on which the linear transformation is learned based on a single batch of data, and the test stage.

**Full model performance** We evaluated our method on different CNN architectures: ResNet-18, 50, 101 [HZRS16b]; MobileNetV2 [SHZ+18b]; and Inception V3 [SVI+16]. Specifically, MobileNetV2 is known to be unfriendly to activation quantization [SFZ+18b]. Performance was evaluated on ImageNet dataset [RDS+15] on which the networks were pre-trained. The proposed method was applied to the outputs of all convolutional layers, while the weights were quantized to either 4 or 8 bits (two distinct configurations) using the method proposed by (author?) [BNHS18]. Rates are reported both in terms of the entropy value and the average length of the feature maps compressed using Huffman VLC in Fig. 5.3. We observed that higher compression is achieved for covariance matrices with fast decaying eigenvalues describing low-dimensional data. A full analysis can be found in Section 5.4.1.

**Comparison to other methods** We compare the proposed method with other post-training quantization methods: ACIQ [BNHS18], GEMMLOWP [JW17], and KLD [Mig17]. Note that our method can be applied on top of any of them to further reduce the memory bandwidth. For each method, we varied the bitwidth and chose the smallest one that attained top-1 accuracy within 0.1% from the baseline and measured the entropy of the activations. Our method reduces, in average, 36% of the memory bandwidth relatively to the best competing methods; the full comparison can be found in Table 5.2.

As for other memory bandwidth reduction methods, our method shows better performance than (author?) [CB19] at the expense of performance degradation (Table 5.1).
Table 5.2: Comparison of our method against three known post-training quantization methods ((i) ACIQ [BNHS18]; (ii) GEMMLOWP [JW17]; (iii) KLD [Mig17]. We report the smallest bit per value for which degradation is at most 0.1% of the baseline.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Weights (bits)</th>
<th>Method</th>
<th>Activations (avg number of bits per value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-50</td>
<td>8</td>
<td>GEMMLOWP</td>
<td>6.88</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACIQ</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KLD</td>
<td>6.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Our method</td>
<td>4.15</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>GEMMLOWP</td>
<td>6.93</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACIQ</td>
<td>6.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KLD</td>
<td>6.52</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Our method</td>
<td>4.25</td>
</tr>
<tr>
<td>Inception V3</td>
<td>8</td>
<td>GEMMLOWP</td>
<td>6.93</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACIQ</td>
<td>6.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KLD</td>
<td>6.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Our method</td>
<td>4.3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>GEMMLOWP</td>
<td>6.97</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACIQ</td>
<td>6.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KLD</td>
<td>6.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Our method</td>
<td>4.6</td>
</tr>
<tr>
<td>MobileNetV2</td>
<td>8</td>
<td>GEMMLOWP</td>
<td>8.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACIQ</td>
<td>7.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KLD</td>
<td>7.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Our method</td>
<td>3.8</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>GEMMLOWP</td>
<td>8.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACIQ</td>
<td>7.85</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KLD</td>
<td>8.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Our method</td>
<td>4</td>
</tr>
</tbody>
</table>

The performance difference is smaller in MobileNetV2, since mobile architectures tend to be less sparse [PXB18], making RLE less efficient. While the method proposed by (author?) [GHR18] requires fine-tuning, our method, although introducing computational overhead, can be applied to any network without such limitations. In addition, similarly to [GHR18] it is possible to compress only part of the layers in which the activation size is most significant.
Ablation study  An ablation study was performed using ResNet-18 to study the effect of different ingredients of the proposed encoder-decoder chain. The following settings were compared:

- only quantization of the feature maps with standard uniform quantization (Q);
- applying PCA transformation to the feature maps and quantizing the latter (PCA→Q);
- applying quantization to the feature maps and then compressing them using VLC, without PCA (Q→VLC);
- the full suggested method (PCA→Q→VLC).

The resulting rate-distortion curves are compared in Fig. 5.4 (left). Our results confirm the previous result of (author?) [Cha18], suggesting that VLC applied to quantized activations can significantly reduce memory bandwidth. They further show that a combination with PCA makes the improvement dramatically bigger. In addition, we analyze the effect of truncating the least significant principal components, which reduces the computational overhead of PCA. Fig. 5.4 (right) shows the tradeoff between the computational and memory complexities, with baseline accuracy and 0.5% below the baseline.

In Table 5.2 we show the comparison of our method with other post-training quantization method. For fair comparison, we add to all compared methods a VLC and show the minimum amount of information that is required to be transferred to the memory.

5.4.1 Eigenvalues analysis

The eigenvalues of the covariance matrix is a measure of the dispersal of the data. If high energy ratio, means the cumulative sum of eigenvalues divided by the total sum, can be expressed with small part of the eigenvalues, the data is less dispersal and therefore more compressible. In figure 5.5 we analyze the covariance energy average ratio in all layers of different architectures. The interesting conclusion is that the ability of compression with the suggested algorithm is correlated with the covariance energy average ratio , means that for new architectures we can look only at the energy ratio of the activation to measure our ability of compression.

5.4.2 Block shape and size

Fig. 5.6 shows the rate-distortion curves for blocks of the same size allocated differently to each of the three dimensions; the distortion is evaluated both in terms of the MSE and the network classification accuracy. The figure demonstrates that optimal performance for high accuracy is achieved with $1 \times 1 \times C = n$ blocks, suggesting that the correlation between the feature maps is higher than that between spatially adjacent activations.
For lower accuracy, bigger blocks are even more efficient, but the overhead of 4 times bigger block is too high. Experiments reported later in the paper set the block size to values between 64 to 512 samples.
Table 5.3: Logic utilization and memory energy consumption of layers of various widths on Intel’s Stratix10 FPGA. Clock frequency was fixed at 160MHz for each design. In LUTs and DSP we present the % of total resources. In Power and Bandwidth we present the total number (% saving comparison to regular quantization)

<table>
<thead>
<tr>
<th># channels</th>
<th>Method</th>
<th>LUTs</th>
<th>DSPs</th>
<th>Energy (µJ)</th>
<th>Bandwidth (Gbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>Quantization</td>
<td>19K</td>
<td>960</td>
<td>225.93</td>
<td>1.28</td>
</tr>
<tr>
<td></td>
<td>Q+VLC</td>
<td>19K</td>
<td>960</td>
<td>173.44 (-23%)</td>
<td>0.96 (-25%)</td>
</tr>
<tr>
<td></td>
<td>Q+VLC+PCA</td>
<td>19.5K (+5%)</td>
<td>1056(+10%)</td>
<td>100.6 (-44%)</td>
<td>0.68 (-46.8%)</td>
</tr>
<tr>
<td>128</td>
<td>Quantization</td>
<td>43K</td>
<td>2240</td>
<td>112.96</td>
<td>1.28</td>
</tr>
<tr>
<td></td>
<td>Q+VLC</td>
<td>43K</td>
<td>2240</td>
<td>148 (-17%)</td>
<td>1.04 (-18.7%)</td>
</tr>
<tr>
<td></td>
<td>Q+VLC+PCA</td>
<td>45K(+4.3%)</td>
<td>2366(+5.6%)</td>
<td>117 (-35%)</td>
<td>0.83 (-35.1%)</td>
</tr>
<tr>
<td>256</td>
<td>Quantization</td>
<td>91K</td>
<td>4800</td>
<td>56.5</td>
<td>1.28</td>
</tr>
<tr>
<td></td>
<td>Q+VLC</td>
<td>91K</td>
<td>4800</td>
<td>46.8 (-17.2%)</td>
<td>1.03 (-19.5%)</td>
</tr>
<tr>
<td></td>
<td>Q+VLC+PCA</td>
<td>93K(+4%)</td>
<td>5059(+5.4%)</td>
<td>103 (-42.8%)</td>
<td>0.73 (-42.9%)</td>
</tr>
<tr>
<td>512</td>
<td>Quantization</td>
<td>182K</td>
<td>9600</td>
<td>28.2</td>
<td>1.28</td>
</tr>
<tr>
<td></td>
<td>Q+VLC</td>
<td>182K</td>
<td>9600</td>
<td>23.9 (-15.6%)</td>
<td>1.05 (-17.9%)</td>
</tr>
<tr>
<td></td>
<td>Q+VLC+PCA</td>
<td>186K(+2%)</td>
<td>10051(+4.7%)</td>
<td>91 (-49.5%)</td>
<td>0.66 (-48.4%)</td>
</tr>
</tbody>
</table>

5.5 Hardware Implementation

In order to verify the practical impact of the proposed approach of reducing feature map entropy to save total energy consumption, we implemented the basic building blocks of a pre-trained ResNet-18, with weights and activation quantized to 8 bit, on Intel Stratix-10 FPGA, part number 1SX280LU3F50I2VG. From logic utilization and memory energy consumption (exact numbers are shown in Table 5.3) of convolutional layers we conclude that our method add minor computational overhead in contrast to significant reduction in memory energy consumption. Fig. 5.7 shows total energy consumption for a single inference of ResNet-18 on ImageNet. In particular, our method is more efficient for higher accuracies, where the redundancy of features is inevitably higher. We further noticed that our approach reached real-time computational inference speed (over 40 fps). The reduction in memory bandwidth can be exploited by using cheaper, slower memory operating at lower clock speeds, which may further reduce its energy footprint. Source files for hardware implementation can found at reference implementation.

We have implemented ResNet-18 using a Stratix-10 FPGA by Intel, part number 1SX280LU3F50I2VG. The memory used for energy calculation is the Micron 4Gb x16 - MT41J256M16. Current consumption of the DDR was taken from the data sheet for read and write operation, and was used to calculate the energy required to transfer the feature maps in each layer. The script for calculating the energy consumption accompanies reference implementation.

In our design each convolutional layer of ResNet-18 is implemented separately. In
each layer we calculate 1 pixel of 1 output feature each clock. For example, the second
layer has 64 input and 64 output feature maps, thus it takes $64 \times (56 \times 56)$ clock cycles
to calculate the output before moving to the next layer.

We read the input features only once by caching the pixels and reusing them from
internal memory, and only reloading the weights of the filters in the current layer.
5.6 Discussion

This chapter presented a proof-of-concept of energy optimization in NN inference hardware by lossy compression of activations prior to their offloading to the external memory. Our method uses transform-domain coding, exploiting the correlations between the activation values to improve their compressibility, reducing bandwidth by approximately 25% relative to VLC and approximately 40% relative to an 8-bit baseline without accuracy degradation and by 60% relative to an 8-bit baseline with less than 2% accuracy degradation. The computational overhead required for additional linear transformation is relatively small and the proposed method can be easily applied on top of any existing quantization method.
Chapter 6

CAT: Compression-Aware Training for Bandwidth Reduction

6.1 Motivation

One major obstacle hindering the ubiquitous use of CNNs for inference is their relatively high memory bandwidth requirements, which can be the primary energy consumer and throughput bottleneck in hardware accelerators. Inspired by quantization-aware training approaches, we propose a compression-aware training (CAT) method that involves training the model to allow better compression of weights and feature maps during neural network deployment. Our method trains the model to achieve low-entropy feature maps, enabling efficient compression at inference time using classical transform coding methods. CAT significantly improves the state-of-the-art results reported for quantization evaluated on various vision and NLP tasks, such as image classification (ImageNet), image detection (Pascal VOC), and sentiment analysis (CoLa). For example, on ResNet-18, we achieve near baseline ImageNet accuracy with an average representation of only 1.5 bits per value with 5-bit quantization. Moreover, we show that entropy reduction of weights and activations can be applied together, further improving bandwidth reduction.
6.2 Introduction

The desire for reduced bandwidth and compute requirements of deep learning models has driven research into quantization [HCS+17, YSX+19, LDX+19, GLJ+19], pruning [LDS90, LKD+17, MMT+19], and sparsification [GEH19, DZ19].

In particular, quantization works usually focus on scalar quantization of the feature maps: mapping the activation values to a discrete set \( \{q_i\} \) of size \( L \). Such representation, while being less precise, is especially useful in custom hardware, where it allows more efficient computations and reduces the memory bandwidth requirement. In this work, we focus on the latter, which has been shown to dominate the energy footprint of CNN inference on custom hardware [YCES17]. We show that the quantized activation values \( \{q_i\} \) can further be coded to reduce memory requirements.

Raw quantized data require \( \lceil \log_2(L) \rceil \) bits per value for storage, which quantity can be reduced by compressing the feature maps. In particular, in the case of element-wise compression of independent identically distributed values, the lower bound of the amount of bits per element is given by the entropy [Sha48]:

\[
H(bq) = - \sum_{i=1}^{L} p(q_i) \log_2 p(q_i)
\]

(6.1)

of the quantized values \( \{q_i\} \), where \( p(q_i) \) denotes the probability of \( q_i \).

In this work, we take a further step by manipulating the distribution of the quantization values so that the entropy \( H(q) \) is minimized. To that end, we formulate the training problem by augmenting the regular task-specific loss (the cross-entropy classifier loss in our case) with the feature map entropy serving as a proxy for the memory rate. The strength of the latter penalty is controlled through a parameter \( \lambda > 0 \). Fig. 6.1 demonstrates the effect of the entropy penalty on the compressibility of the intermediate activations.

In contrast to previous works that employed entropy encoders [AMT+17, ACA19] for weight compression, we focus on compression of activations. Activations are responsible for a significant part of the memory I/O during inference [YCS17, SSMM18], and their efficient encoding provides significant benefits in terms of power. Nevertheless, we show that the proposed method is compatible with the previously proposed weight compression approaches.

**Contributions.** Our paper makes several contributions. Firstly, we introduce Compression-Aware Training (CAT), a novel technique for memory bandwidth reduction. The method works by introducing a loss term that penalizes the entropy of the activations at training time and by applying entropy encoding (e.g., Huffman coding) on the resulting activations at inference time. Similar to many successful and widely-used techniques in deep learning, the proposed scheme is almost straightforward to implement.
Figure 6.1: Pre-activation distributions of one layer in ResNet-18. (a) **Evolution at different epochs.** As training progresses, the probability of non-positive pre-activation values increases, zeroing more post-ReLU values. The sharp peak at zero reduces entropy and thus improves compressibility. (b) **Effect of entropy regularization.** Without regularization, the distribution has much heavier tails and thus has higher entropy. As regularization increases, the probability of extreme values is significantly reduced. The entropy penalty $\lambda$ was selected so that the overall accuracy is not affected. The compression ratio in the strongly regularized case is 2.23 times higher compared to the unregularized baseline.

Since the only overhead of the method at inference time is entropy encoding, the improvement is universal for any hardware implementation, being especially efficient on computationally optimized ones, where memory I/O dominates the energy footprint [YCES17, JYP+17].

We demonstrate a two- to fourfold memory bandwidth reduction for multiple architectures: MobileNetV2 and ResNet on the ImageNet visual recognition task, SSD512 on the PASCAL VOC object detection task and BERT on CoLa sentiment analysis task. We also investigate several differentiable loss functions that lead to activation entropy minimization and show a few alternatives that lead to the same effect. Moreover, applying the same regularization to both the weights and the activations allows a further reduction in the memory bandwidth with only minor loss of accuracy. In addition, we show that entropy reduction of weights and activation can be efficiently applied together, further improving bandwidth reduction.

Finally, we analyze the method’s rate–distortion tradeoff, achieving even stronger compression at the expense of a minor reduction in accuracy: for ResNet-18, we manage to achieve entropy inferior to one bit per value, at the expense of losing 2% of the top-1 accuracy.
6.3 Method

We consider a feed-forward DNN $F$ composed of $L$ layers. Each subsequent layer processes the output of the previous one: $x^i = F_i(x^{i-1})$, using the parameters $b_{w_i} \in \mathbb{R}^{N_i \times N_{i-1}}$. We denote by $x^0 = x$ and $x^L = y$ the input and output of the network, respectively, and the number of elements of $x^i$ as $N_i$. The parameters $b_{w}$ of the network are learned by minimizing $\mathcal{L}(x, y; b_{w}) + \lambda \mathcal{R}(b_{w})$, with the former term $\mathcal{L}$ being the task loss, and the latter term $\mathcal{R}$ being a regularizer (e.g., $\|b_{w}\|_2$) inducing some properties on the parameters $b_{w}$.

6.3.1 Entropy encoding and rate regularization

Entropy encoders are a family of lossless data encoders that compress each symbol independently. In this case, assuming i.i.d. distribution of the input, it has been shown that optimal code length is $-\log_b p_i$, where $b$ is the size of the alphabet and $p_i$ is the probability of the $i$th symbol [Sha48]. Thus, for a discrete random variable $X$, we define an entropy $H(X) = -\mathbb{E} \log_2 X = -\sum_i p(x_i) \log_2 p(x_i)$, which is a lower bound on the amount of information required for lossless compression of $X$. The expected total space required to encode the message is $N \cdot H$, where $N$ is the number of symbols. Since we encode the activations with the entropy encoder before writing them into memory, we would like to minimize the entropy of the activations to improve the compression rate.

One example of an entropy encoder is Huffman coding – a prefix coding that assigns shorter codes to the more probable symbols. The simplicity along with the high compression rate [Szp00], bounded by $H(X) \leq R \leq H(X) + 1$, renders it especially useful in performance-critical applications. The comparison of Huffman coding rates to entropy in case of activation compression is shown in Fig. 6.4. Other entropy encoders, such as arithmetic coding or asymmetric numeral systems [DTGD15], can provide even better compression rates; for instance, for large enough inputs, arithmetic
coding achieves optimal rates. These schemes, however, require more computational resources for encoding and are harder to implement.

### 6.3.2 Differentiable entropy-reducing loss

Since the empirical entropy is a discrete function, it is not differentiable and thus cannot be directly minimized with gradient descent. Nevertheless, there exist a number of differentiable functions which either approximate entropy or have same minimizer. Thus, we optimize

\[ L = L_p + \lambda L_H, \]

where \( L_p \) is a target loss function and \( L_H \) is some regularization that minimizes the entropy.

**Soft entropy** First, we consider the differentiable entropy estimation suggested by [AMT+17]. We start from the definition of the entropy,

\[ H(X) = -\sum_i p(x_i) \log(p(x_i)) \]

\[ p(x_i) = \frac{|\{x|x = q_i\}|}{N}, \]

where \( b_q \) is a vector of quantized values. Let \( m \) be an index of the bin to which the current value is mapped, and \( bQ \) a one-hot encoding of this index, i.e.,

\[ q_m = \arg \min_{q_i \in Q} |x - q_i| = \arg \max_{q_i \in Q} (-|x - q_i|) \]

\[ bQ_i = \delta_{im}, \]

where \( \delta_{im} \) denotes Kronecker’s delta. To make the latter expression differentiable, we can replace argmax with softmax:

\[ \tilde{b}Q(x) = \text{softmax}(-|bx - b_q|, T), \]

where \( T \) is the temperature, and \( \tilde{b}Q(x) \to bQ(x) \) as \( T \to 0 \). Finally, the soft entropy \( \hat{H} \) is defined as

\[ \hat{H}(X) = -\sum_i \hat{p}(x_i) \log(\hat{p}(x_i)) \]

\[ \hat{p}(x_i) = \frac{\sum_j bQ_i(x_j)}{N}. \]

To improve both memory requirements and time complexity of the training, we calculate the soft entropy only on part of the batch, reducing the amount of computation and the gradient tensor size. In particular, we try to take each \( k^{th} \) pixel of every feature map or a random subset of the activation, both leading to the same performance.
We empirically confirm that this choice gives a reasonable approximation of the real entropy (Section 6.4.1).

**Compressibility loss** An alternative loss promoting entropy reduction was proposed by [ACA19] under the name of compressibility loss and based on earlier work by [Hoy04]:

$$L_c = \frac{\|b_x\|_1}{\|b_x\|_2}. \tag{6.10}$$

This loss has the advantage of computational simplicity, and has been shown both theoretically and practically to promote sparsity and low entropy in input vectors. While originally applied to the weights of the network, here we apply the same loss to the activations. As shown in Section 6.4.1, both the soft entropy and the compressibility loss lead to similar results.

We summarize the proposed method for reducing memory bandwidth as follows: at training time, we fine-tune (training from scratch is also possible but was not performed in our experiments) the pre-trained network $\mathcal{F}$ with the regularized loss (6.2), with $L_H = \sum N_i \cdot \hat{H}(x^i)$ in the case of differentiable entropy and $L_H = \sum L_c(x^i)$ in the case of compressibility loss, where the sum is running over network layers. At test time, we apply entropy coding on the activations (on a per layer basis) before writing them to memory, thus reducing the amount of memory transactions. In contrast to [CBB⁺19], who avoided fine-tuning by using test-time transformation to reduce entropy, our method does not require complex transformations at test time because it induces low entropy during training.
6.4 Experimental Results

We evaluate the proposed scheme on common CNN architectures for image classification on ImageNet [RDS+15] (ResNet-18/34/50, MobileNetV2), object detection on Pascal VOC [EVGW+10] (SSD512 [LAE+16]) as well as Transformers (BERT) [DCLT19] for sentimental analysis on CoLA [WSB18].

To perform an evaluation in conditions close to real-life setups, we chose to quantize the weights to 8 bits and activations to the range of 4–8 bits. Adding a complicated quantization method to CAT would require more resources and might add additional noise to the measurements, without contributing much to the analysis of CAT. Thus, we chose a simple quantization method and did not attempt to achieve state-of-the-art quantization performance for the baseline.

The weights were initialized with a pre-trained model and quantized with uniform quantization using the shadow weights, i.e. applying updates to a full precision copy of quantized weights [HCS+17, RORF16]. The activations were clipped with a learnable parameter and then uniformly quantized as suggested by [BLC+18c]. Similarly to previous works [ZNZ+16, RORF16], we used the straight-through estimator [BLC13] to approximate the gradients. We quantize all layers in the network, in contrast to the common practice of leaving the first and last layers in high precision [ZNZ+16, BLC+18c].

For optimization, we used SGD with a learning rate of $10^{-4}$, momentum 0.9, and weight decay $4 \times 10^{-5}$ for up to 30 epochs (usually, 10 to 15 epochs were sufficient for convergence). Our initial choice of temperature was $T = 10$, which performed well. We tried to apply exponential scheduling to the temperature [JGP16], but it did not have any noticeable effect on the results.

In Fig. 6.2 we compare our method with EPBC ([CRB19]) and GF ([GHR18]). EPBC is based on a lossless compression method that maintains the full precision accuracy while reducing the bit rate to approximately 3.5 bits/value in both models. GF, on the other hand, provides strong compression at the expense of larger accuracy degradations. In addition, [GHR18] compressed only part of the layers. Unlike these two methods, CAT allows more flexible tradeoff between compression and accuracy. CAT shows better results in ResNet-34 and shows either better accuracy or compression for MobileNetV2. We also ran our method on additional architectures: ResNet-18, ResNet-50, and SSD512 with VGG backbone; the results are listed in Table 6.1. Even though we cannot directly compare detection results with [GHR18], the drop in accuracy is lower in our case. Additional experimental results are presented in the Appendix.

In Table 6.2 we evaluate our method on BERT. Since those models are more challenging for quantization, we used 8-bit quantization for both weights and activations. We used sentiment analysis task on CoLa dataset for evaluation. The primary performance metric in this case is Matthews correlation coefficient (MCC). CAT was able to

---

1Our code is based on an implementation by [Li18].
Table 6.1: Results for ResNet-18, ResNet-50, and SSD512. We include the results by [GHR18] for the SSD512 model on the same task but with a different backbone, for which we obtain a better compression with a lower accuracy degradation. Compute denotes the activation bitwidth used for arithmetic operations. Memory denotes the average number of bits for memory transactions (after compression). Compression ratio denotes the reduction in representation size. Weight bitwidth is 8 except for the full-precision experiments. Additional results are provided in the Appendix.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Compute (bits)</th>
<th>Memory (bits)</th>
<th>Compression ratio</th>
<th>Top-1 accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18, CAT</td>
<td>32</td>
<td>32</td>
<td>1</td>
<td>69.70</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.5</td>
<td>3.33</td>
<td>69.20</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.51</td>
<td>2.65</td>
<td>68.08</td>
</tr>
<tr>
<td>ResNet-50, CAT</td>
<td>32</td>
<td>32</td>
<td>1</td>
<td>76.1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.60</td>
<td>3.125</td>
<td>74.90</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.78</td>
<td>2.25</td>
<td>74.50</td>
</tr>
<tr>
<td>SSD512-SqueezeNet</td>
<td>32</td>
<td>32</td>
<td>1</td>
<td>68.12</td>
</tr>
<tr>
<td>[GHR18]</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>64.39</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>62.09</td>
</tr>
<tr>
<td>SSD512-VGG, CAT</td>
<td>32</td>
<td>32</td>
<td>1</td>
<td>80.72</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>2.334</td>
<td>2.57</td>
<td>77.49</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.562</td>
<td>2.56</td>
<td>77.43</td>
</tr>
</tbody>
</table>

Table 6.2: Results of compression of both weights and activations for sentiment analysis task on CoLa dataset using BERT model. The notation is the same as in Table 6.1, as compared to full precision baseline and Huffman coding without regularization. We provide standard deviation over different samples in form mean ± std.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Compute weights (bits)</th>
<th>Memory, embeddings weights (bits)</th>
<th>Memory, fully connected weights (bits)</th>
<th>MCC (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>32</td>
<td>32</td>
<td>4.17 ± 0.10</td>
<td>49.23</td>
</tr>
<tr>
<td>No regularization</td>
<td>8</td>
<td>5.06</td>
<td>5.1</td>
<td>49.00</td>
</tr>
<tr>
<td>CAT</td>
<td>8</td>
<td>2.64 ± 0.07</td>
<td>2.97 ± 0.08</td>
<td>49.10</td>
</tr>
</tbody>
</table>

compress feature maps to ~ 3-bit, while Huffman coding alone requires almost twice higher bandwidth. In both cases there is only minor performance degradation.

6.4.1 Ablation study

Rate-Accuracy Tradeoff  The proposed algorithm tries to balance between the compression and the accuracy of the network by means of the parameter λ in Eq. (6.2). To evaluate this tradeoff, we trained ResNet-18 and MobileNetV2 with different values of λ in the range of 0 – 0.3, with results shown in Fig. 6.4. Increasing the value of the regularization term results in weights that produce lower-entropy activations and thus allow a better compression rate, at the expense of accuracy degradation. We show the
Figure 6.3: Tradeoff between rate and accuracy for (a) ResNet-18 and (b) ResNet50 in weight compression. The activations are quantized to 8 bits for fair comparison. “No fine-tuning” refers to pre-trained model with activations and weights quantized to 8 bits and the Huffman coding applied to the weights.

Figure 6.4: Tradeoff between rate and accuracy for different values of $\lambda$ (ranged between 0 and 0.3) in (a) ResNet-18 and (b) MobileNetV2. In ResNet-18, the activations are quantized to 5 bits; in MobileNet we show results for activation quantized to 6 and 8 bits.

values of the theoretical entropy and the average bitwidth of the Huffman-coded activations. The main advantage of Huffman coding is computational efficiency: even the naive implementation of the Huffman coding introduces only 4% overhead for inference time. For high bitwidth, Huffman coding is close ($\sim$3% overhead) to the theoretical entropy, while for lower entropy there is a larger difference (in particular, Huffman coding is bounded from below by 1 bit per value) – in this case, different lossless coding schemes such as arithmetic coding or asymmetric numeral systems [DTGD15] can provide better results.

**Robustness** To check the robustness of our method, we performed several runs with the same hyperparameters and a different random seed. The results, reported in Table 6.3, suggest that the method is robust and stable under random initialization.
Table 6.3: Mean and standard deviation over five runs of ResNet-18 and ResNet-34 with 5 bit compute.

<table>
<thead>
<tr>
<th>Network</th>
<th>Accuracy, % (mean ± std)</th>
<th>Memory, bits (mean ± std)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>69.122 ± 0.016</td>
<td>1.5150 ± 0.0087</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>73.025 ± 0.095</td>
<td>1.7875 ± 0.033</td>
</tr>
</tbody>
</table>

Table 6.4: Performance of soft entropy (6.8) and compressibility loss (6.10) on ResNet-18.

<table>
<thead>
<tr>
<th>Compute, bits</th>
<th>Loss</th>
<th>Accuracy</th>
<th>Memory, bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>entropy</td>
<td>67.86%</td>
<td>1.43</td>
</tr>
<tr>
<td>4</td>
<td>comp.</td>
<td>67.84%</td>
<td>1.50</td>
</tr>
<tr>
<td>5</td>
<td>entropy</td>
<td>69.49%</td>
<td>1.79</td>
</tr>
<tr>
<td>5</td>
<td>comp.</td>
<td>69.36%</td>
<td>1.73</td>
</tr>
</tbody>
</table>

Figure 6.5: Soft entropy with different sample sizes compared to a real entropy of a single batch. The shaded region covers the standard deviation over three runs.

**Soft entropy vs. compressibility loss** Replacing the soft entropy with a different entropy-minimizing loss has a minor effect on the results (Table 6.4). This suggests that the desired effect is the result of an entropy reduction rather than a particular form of regularization promoting it.

**Batch size** We noticed that training ResNet-50 on a single GPU mandated the use of small batches, leading to performance degradation. Increasing the batch size from 16 to 64 without other changes increased accuracy by more than 0.5% with an entropy increase of less than 0.1 bits/value.

**Sample size in soft entropy calculation** To check whether the number of values used to calculate soft entropy is enough, we ran a soft entropy evaluation on a single tensor.
Table 6.5: Results of compression of both weights and activations for ResNet-18 and ResNet-50. The notation is the same as in Table 6.1. We add the baseline (“Memory, original”) that employs Huffman coding but no regularization to pre-trained model. We provide standard deviation over different samples in form mean ± std.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Compute (bits)</th>
<th>Memory, original (bits)</th>
<th>Memory, CAT (bits)</th>
<th>Top-1 accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>weights</td>
<td>activ.</td>
<td>weights</td>
<td>activ.</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>6</td>
<td>5.1</td>
<td>3.5</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>6</td>
<td>4.6</td>
<td>3.6</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>5</td>
<td>4.6</td>
<td>2.95</td>
</tr>
</tbody>
</table>

and compared it to real values. Since the tensors are large (hundreds of thousands of elements), even 5% of the values already provide a reasonable approximation of the real entropy, as shown in Fig. 6.5.

**Weight compression** We also show that the proposed method can be applied along with weight compression. First, we performed experiments with regularization applied only to weights. Fig. 6.3 shows the result for ResNet-18 and ResNet-50 and 6.2 show the result of BERT on CoLa dataset. Both activations and weights are quantized to 8 bits. The regularization provides a significant improvement, reducing the entropy of weights to ~ 2.5 bits without a loss in accuracy or MCC score. A further increase in regularization allows us to achieve 2-bit entropy with less than 1% loss in accuracy.

Finally, we apply the entropy reduction to both weights and activations. The results are presented in Table 6.5. With less than 1% loss in accuracy, we get an almost twofold improvement over unregularized version in both weight and activation entropy.
6.5 Discussion

Quantization of activations reduces memory access costs that are responsible for a significant part of the energy footprint of DNN accelerators. Conservative quantization approaches, known as post-training quantization, take a model trained for full precision and directly quantize it to 8-bit precision. These methods are simple to use and allow for quantization with limited data. Unfortunately, post-training quantization below 8 bits usually incurs significant accuracy degradation. Quantization-aware training approaches involve some sort of training either from scratch [HCS+17], or as a fine-tuning step from a pre-trained floating point model [HMD16]. Training usually compensates significantly for a model’s accuracy loss due to quantization.

In this work, we take a further step and propose a compression-aware training method to aggressively compress activations to as low as 2 average bit/value representations without harming accuracy. Our method optimizes the average bit per value needed to represent activation values by minimizing the entropy. We demonstrate the applicability of our approach and its compatibility with other compression methods, such as quantization and weight entropy reduction, on classification tasks using the MobileNetV2 and various ResNet models, as well as an object detection task using the model SSD512 and sentimental analysis task using BERT model. Also, we show that entropy reduction of weights and activations can be applied together, further improving bandwidth reduction. Because of the low overhead, the method provides universal improvement for any custom hardware, being especially useful for accelerators with efficient computations, where memory transfers are a significant part of the energy budget. We show that the effect is universal among loss functions and robust to random initialization.
Chapter 7

Early-Stage Neural Network Hardware Performance Analysis

7.1 Motivation

Due to the significant success of convolutional neural network (CNN) approaches in various tasks, including image recognition and generation, the demand for running NNs in embedded environments has increased significantly in recent years. The task of achieving high accuracy on resource-restricted devices, however, is still considered challenging, mainly due to the vast number of design parameters that need to be balanced. While quantization of CNN parameters leads to a reduction of power and area, it can also generate unexpected changes in the balance between communication and computation. This change is hard to evaluate, and the lack of balance may lead to lower utilization of either memory bandwidth or computational resources, thereby reducing performance. This chapter introduces a hardware performance analysis framework for identifying bottlenecks in the early stages of CNN hardware design. We demonstrate how the proposed method can help evaluate different architecture alternatives of resource-restricted CNN accelerators (e.g., part of real-time embedded systems) early in design stages and thus prevent making design mistakes.
7.2 Introduction

Domain-specific systems were found to be very efficient, in general, and when developing constrained devices such as IoT, in particular. A system architect of such devices must consider hardware limitations (e.g., bandwidth and local memory capacity), algorithmic factors (e.g., accuracy and representation of data), and system aspects (e.g., cost, power envelop, battery life, and more). Many IoT and other resource-constrained devices provide support for applications that use convolutional neural networks (CNNs). Such algorithms can achieve spectacular performance in various tasks covering a wide range of domains such as computer vision, medicine, autonomous vehicles, etc. Notwithstanding, CNNs contain a vast number of parameters and require a significant amount of computation during inference, thus monopolizing hardware resources and demanding massively parallel computation engines.

These requirements have led to great interest in using custom-designed hardware for efficient inference of CNNs that would allow the promise of neural networks to be used in real-life applications by deploying them on low-power edge devices or as part of an IP in a SoC, where silicon area is scares. Developing such systems requires a new set of design tools due to the tight entanglement between the algorithmic aspects, the chip architecture and the constraints the end product needs to meet. In particular, great efforts were made to develop low-resource CNN architectures. One example of such architectural changes is the splitting of the regular $3 \times 3$ convolutions into a channel-wise $3 \times 3$ convolution followed by a $1 \times 1$ one. Another way to reduce the computational burden is to quantize the CNN parameters (weights and activations), employing low-bit integer representation of the data instead of the expensive floating point representation. Recent quantization-aware training schemes [YSX+19, JYL19] achieve near-baseline accuracy for as low as 2-bit quantization. The benefit of quantizing the CNN is twofold: both the number of gates required for each multiply-accumulate (MAC) operation and the amount of routing are reduced. Also, quantization reduces the overhaul bandwidth needs from external or internal memory. The decision regarding which algorithm to choose may depend on the architecture (e.g., FPGA or ASIC), the accuracy requirements, and their impact on performance and power. Thus, the architect needs to make these fundamental decisions early in the developing process and no existing tool can help predict these design factors ahead of time. The impact of the high-level architecture of the accelerator, e.g., the shape of CNN layers and the bitwidth of the operands, on the power, the area and the performance of the final product needs to be defined and predicted at an early stage of the project. Recent research has shown that ASIC-based architectures are the most efficient solution for CNN accelerators both in datacenters and in real-time platforms [HLM+16, SHK+14]. Accordingly, we employ an implementation of a streaming ASIC-based convolutional engine for our experiments. Nevertheless, our methodology can be applied for the evaluation of other types of architectures, such as FPGA-based accelerators [BLZ+18].
both cases, the development process includes an important trade-off between the logical
gates area, local memory area, their routing on the silicon versus the performance and
accuracy of the resulting system. Especially on SoC IC, these trade-offs have great
impact where the CNN accelerator is a small part of the entire system, the remaining
silicon ”budget” needs to be divided between execution units and local memory. Un-
fortunately, all these parameters also depend on the quantization level of the data, and
its impact on both communication and computation. To date, there is no quantitative
metric for this trade-off available at the design stage of the CNN accelerator and no
tool exists that can assist the architect to predict the impact of high level decisions on
the important design implementation parameters. Ideally, the designer would like to
have an early estimation of the chip resources required by the accelerator as well as the
performance, accuracy and power it can achieve.

A critical difficulty in trying to predict the design parameters for CNN-based sys-
tems is the lack of a proper complexity metric. Currently, the most common metric for
calculating the computational complexity of CNN algorithms is the number of MAC op-
erations denoted as OPS (or FLOPS in case of floating-point operations). This metric,
however, does not take into account the data format or additional operations performed
during the inference, such as memory accesses and communication. For that reason, the
number of FLOPS does not necessarily correlate with run-time [SCYE17] [LHC+18] or
the required amount of computational resources. This work proposes to use a differ-
ent metric for assessing the complexity of CNN-based architectures: the number of bit
operations (BOPS) as presented by [BSZ+18]. We show that BOPS is well-suited to
the task of comparing the datapath hardware resources of different architectures with
different weight and activation bitwidths.

**Contribution**  This chapter makes several contributions:

- Study the impact of CNN quantization on the hardware implementation in terms
  of computational resources and memory bandwidth considerations.

- extend the previously proposed computation complexity for quantized CNNs,
  termed BOPS [BSZ+18], with a communication complexity analysis to identify
  the performance bottlenecks that may arise from the data movement.

- Implement a quantized convolution block to demonstrate an accurate estimation
  of the power/area of the hardware accelerator.
Table 7.1: 32-bit floating-point and 32-bit fixed-point multiplier design in terms of the number of gates, area, and power.

<table>
<thead>
<tr>
<th>Multiplier</th>
<th>Gates</th>
<th>Area [µm²]</th>
<th>Power [mW]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Internal</td>
<td>Switching</td>
</tr>
<tr>
<td>Floating-Point</td>
<td>40090</td>
<td>11786</td>
<td>2.76</td>
</tr>
<tr>
<td>Fixed-Point</td>
<td>5065</td>
<td>1489</td>
<td>0.49</td>
</tr>
</tbody>
</table>

7.3 Complexity Metric

In this section, we describe our hardware-aware complexity metric (HCM), which takes into account the CNN topology, and define the design rules of efficient implementation of quantized neural networks. The HCM metric assesses two elements: the computation complexity of the data-path, which quantifies the hardware resources needed to implement the CNN computation engine on silicon, how the choice of the amount of local SRAM affects the area budget on the silicon, and the communication complexity, which defines the memory access pattern and bandwidth. We describe the hardware changes resulting from switching from a floating-point representation to a fixed-point one, and then present our computation and communication complexity metrics. All results for the fixed-point multiplication presented in this section are based on the Synopsys standard library multiplier using TSMC’s 28nm process.

7.3.1 The impact of quantization on hardware implementation

Currently, the most common representation of weights and activations for training and inference of CNNs is either 32-bit or 16-bit floating-point numbers. The fixed-point MAC operation, however, requires significantly fewer hardware resources, even for the same input bitwidth. To illustrate this fact, we generated two multipliers: one for 32-bit floating-point and the other for 32-bit fixed-point operands. The results in Table 7.1 show that a fixed-point multiplier uses approximately eight time less area, gates, and power than the floating-point counterpart. Next, we generated a Preocessing Engine (PE) that calculates a convolution with a $3 \times 3$ kernel, a basic operation in CNNs consisting of $3^2$ MAC operations per output value. After switching from floating-point to fixed-point, we explored the area of a single PE with variable bitwidth. Note that accumulator size depends on the network architecture: the maximal bitwidth of the output value is $b_w b_a + \log_2(k^2) + \log_2(n)$, where $n$ is number of input features. Since the extreme values are very rare, however, it is often possible to reduce the accumulator width without harming the accuracy of the network. Fig. 7.1 shows the silicon area of the PE as a function of the bitwidth. We performed a polynomial regression and observed a quadratic dependence of the PE area on the bitwidth, with the

---

1FPU100 from https://opencores.org/projects/fpu100
Figure 7.1: Area ($A$) vs. bitwidth ($b$) for a 3 × 3 PE with a single input and output channel. All weights and activations use the same bitwidth and the accumulator width is 4 bit larger, which is enough to store the result. The quadratic fit is $A = 12.39b^2 + 86.07b - 14.02$ with goodness of fit $R^2 = 0.9999877$.

Coefficient of determination $R^2 = 0.9999877$. This nonlinear dependency demonstrates that quantization impact a network hardware resources is quadratic: reducing bitwidth of the operands by half reduces area and, by proxy, power approximately by a factor of four.

7.3.2 Data-path

We now present the BOPS metric defined in [BSZ+18] as our computation complexity metric for the data-path circuit. In particular, we show that BOPS can be used as an estimator for the area of the data-path in an accelerator. The area, in turn, is found to be linearly related to the power in case of the PEs.

The computation complexity metric describes the amount of arithmetic “work” needed to calculate the entire network or a single layer. BOPS is defined as the number of bit operations required to perform the calculation: the multiplication of $n$-bit number by $m$-bit number requires $n \cdot m$ bit operations, while addition requires $\max(n, m)$ bit operations. In particular, [BSZ+18] show that a $k \times k$ convolutional layer with $b_a$-bit activations and $b_w$-bit weights requires

$$\text{BOPS} = mnk^2 \left(b_a b_w + b_a + b_w + \log_2(nk^2)\right) \quad (7.1)$$

bit operations, where $n$ and $m$ are, respectively, the number of input and output features of the layer. The formula takes into account the width of the accumulator required to accommodate the intermediate calculations, which depends on $n$. The BOPS of an entire network is calculated as a sum of the BOPS of the individual layers. Creating larger accelerators that can process more layers in parallel involves simply replicating
Figure 7.2: Area ($A$) vs. BOPS ($B$) for a $3 \times 3$ PE with a single input and output channel and variable bitwidth. The linear fit is $A = 1.694B + 153.46$ with goodness of fit $R^2 = 0.9989$.

Figure 7.3: Area ($A$) vs. BOPS ($B$) for a $3 \times 3$ PE with variable input ($n$) and output ($m$) feature dimensions, and variable bitwidth. Weights and activations use the same bitwidth and the accumulator width is set to $\log_2(9m) \cdot b_w \cdot b_a$.

the same individual PE design.

In Fig. 7.2, we calculated BOPS values for the PEs from Fig. 7.1 and plotted them against the area. We conclude that for a single PE with variable bitwidth, BOPS can be used to predict the PE area with high accuracy.

Next, we tested the predictive power of BOPS scaling with the size of the design. We generated several designs with variable bitwidths, $b_w = b_a \in \{4, 6, 8\}$, and variable numbers of PEs $n = m \in \{4, 8, 16\}$ used to accommodate multidimensional inputs and outputs that typically arise in real CNN layers. Fig. 7.3 shows that the area depends linearly on the BOPS for the range of two orders of magnitude of total area with goodness of fit $R^2 = 0.9980$. We conclude that since BOPS provides a high-accuracy approximation of the area and power required by the hardware, it can be used as an
early estimator. While the area of the accelerator depends on the particular design of the PE, this only affects the slope of the linear fit, since the area is still linearly dependent on the amount of PEs. An architect dealing with algorithms only can use definitions such as the number of input features and output features, kernel size etc. and get an early estimation how much power is needed to solve the network, without having any knowledge about VLSI constraints in advance. Using information such as a number of input/output features and kernel size, it is possible to immediately assess the amount of area the PEs occupy on the silicon.

7.3.3 Local Memory

Local memory is used to store the weights or activations for further re-use instead of fetching them over again from the main memory. Many accelerators designs [JYL19] starting to implement more and more local memory, enough to fit the entire network’s parameters to not use external memory what so ever. The use of local RAM is more resource expensive than using DDR as the memory for the weights and partial sums, the trade-off for the amount of local SRAM area used vs. PEs that can be placed on the same area to use and the performance gain must be carefully evaluated for each design. From Fig. 7.4 and Fig. 7.5 SRAM area is mostly linear to the amount of bits used, \( R^2 = 0.998 \) and \( R^2 = 0.916 \) for single port RAM and dual port RAM respectively. Now, we can use this linear relation to accurately estimate the area of the local SRAM just from the number of bits we decided to implement.

In order to get the complete area of the accelerator, we can use the BOPS metric to scale up our micro-design of the data-path to get the area of the PEs and derive \( \mu_D \), and for the SRAM we can derive the relation \( \mu_M \) between the area and the number of bits from the graphs of the memory specifications\(^2\) to construct the complete area equation.

\[
\text{Area} = \mu_D \cdot \text{BOPS} + \mu_M \cdot \text{sram\_bits}
\]  

(7.2)

7.3.4 Communication

Another important aspect of hardware implementation of CNN accelerators is memory communication. The transmission of data from the memory and back is often overlooked by hardware implementation papers [CES16, Che16] that focus on the raw calculation ability to determine the performance of their hardware. In many cases, there is a difference between the calculated performance and real-life performance, since real-life implementations of accelerators are often memory-bound [MHS+19, JYP+17, WDCC19].

\(^2\)The graphs show Synopsys 28nm Educational Design Kit SRAM specifications
For each layer, the total memory bandwidth is the sum of the activation and weight sizes read and written from memory. In typical CNNs used, e.g., in vision tasks, the first layers consume most of their bandwidth for activations, whereas in deeper layers that have smaller but higher-dimensional feature maps (and, consequently, a bigger number of kernels), weights are the main source of memory communication.

We assume that each PE can calculate one convolution result per clock cycle and the resulting partial sum is saved in the cache. In Fig. 7.6, we show typical memory access progress at the beginning of the convolutional layer calculation. At first stage, the weights and the first $k$ rows of the activations are read from memory at maximal possible speed to start the calculations as soon as possible. After the initial data are loaded, the unit reaches a “steady state”, in which it needs to read from the memory only one new input value per clock cycle (other values are already in the cache). We assume the processed signals to be two-dimensional (images), which additionally requires $k$ new values to be loaded in the beginning of each new row.
Note that until the weights and the first activations are loaded, no calculations are performed. The overhead bandwidth of the pre-fetch stage can be mitigated by doing work in greater batch sizes, loading the weights once and reading several inputs for the same weights. By doing this, we minimize the penalty for reading the weights compared to reading the actual input data to perform the calculation. In the case of real-time processing, however, larger batches are not possible because the stream of data needs to be completed on-the-fly. We focus on the latter real-time streaming regime in this work because of its great importance in a range of applications including automotive, security, and finance. The memory access pattern depicted in Fig. 7.6 must be kept in mind when designing the hardware, since it may limit the performance of the accelerator and decrease its power efficiency.

### 7.4 Roofline analysis

So far, we discussed the use of BOPS for the prediction of the physical parameters of the final product, such as the expected power and area. In this section, we extend the BOPS model to a system level, by introducing the OPS-based roofline model. The traditional roofline model, as introduced by [WZ19], suggests depicting the dependencies between the performance (e.g., GFLOPS/second) and the operation density (the average number of operations per information unit transferred over the memory bus). Now, for each machine we can draw “roofs”: the horizontal line that represents its computational bounds and the diagonal line that represents its maximal memory bandwidth. An example of the roofline for three applications assuming infinite compute resources and memory bandwidth is shown in Fig. 7.7. The maximum performance a machine can achieve for any application is visualized by the area below both bounds, shaded in green.

#### 7.4.1 OPS-based roofline model

Since, as indicated in Section 7.3.1, FLOPS cannot be used for efficient estimation of the complexity of quantized CNNs, we introduce a new model that is based on
Figure 7.7: Roofline example. In the case of App1, memory bandwidth prevents the program from achieving its expected performance. In the case of App2, the same happens due to limited computational resources. Finally, App3 represents a program that could achieve its maximum performance on a given system.

the BOPS metric presented in Section 7.3.2. This model, to which we refer as the OPS-based roofline model, replaces the GFLOPS/s axis of the roofline plot with a performance metric more adequate for neural networks, e.g., number of operations per second (OPS/s), and the second metric that measures the computational complexity with operations per bit (OPS/bit). Using generic operations and bits allows plotting quantized accelerators with different bitwidths on the same plot.

As an example of the proposed approach, we use two different ResNet-18 layers (a deep layer, which is computationally-intensive, and an early one, which is memory-intensive) on four different accelerator designs: 32-bit floating-point, 32-bit fixed-point, and quantized 8-bit and 4-bit fixed-point. The accelerators were implemented using standard ASIC design tools, as detailed in Section 7.5 and were built using the TSMC 28nm technology, using standard 2.4GHz DDR-4 memory with a 64-bit data bus.

7.4.2 Roofline Example 1 - Compute & Memory Bound

The first example employs an accelerator with a silicon area of $1\text{mm}^2$ and 800MHz clock speed. The task is the 11th layer of ResNet-18 that has a $3 \times 3$ kernel and 256 input and output features of dimension $14 \times 14$ each. Looking at Table 7.1, it is possibly to fit only 85 32-bit floating-point multipliers in $1\text{mm}^2$. That allows installation of 9 PEs (without taking into account the area required for the accumulators of the partial sums) and calculation of convolutions with the $3 \times 3 \times 3 \times 3$ kernel in a single clock. Using the known areas of 4-bit, 8-bit and 16-bit PEs, we extrapolate the area of the 32-bit fixed point PE to be $16676\mu\text{m}^2$. From these data, we can place 60 PEs with $7 \times 7 \times 3 \times 3$ kernels, 220 PEs with $14 \times 14 \times 3 \times 3$ kernels and 683 PEs with $26 \times 26 \times 3 \times 3$ kernels, for 32-bit, 16-bit and 8-bit fixed-point PEs, respectively, on the given area.
Table 7.2: The amount of computation (OPS/s) provided by the accelerators and memory throughput (OPS/bit) required by the 11th layer of ResNet-18.

<table>
<thead>
<tr>
<th></th>
<th>32-bit float</th>
<th>32-bit fixed</th>
<th>16-bit quant.</th>
<th>8-bit quant.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOPS/s</td>
<td>72.00</td>
<td>392.0</td>
<td>1568</td>
<td>5408</td>
</tr>
<tr>
<td>OPS/bit</td>
<td>5.82</td>
<td>5.82</td>
<td>11.63</td>
<td>23.26</td>
</tr>
</tbody>
</table>

Figure 7.8: OPS roofline: $3 \times 3$ kernel, 256 input and output $14 \times 14$ features, $1\text{mm}^2$ accelerator with 800MHz frequency, with DDR of $2.4GHz$ with 64 bit data bus.

To calculate the amount of OPS/s required by the layer, under the assumption that a full single pixel is produced every clock, we need to calculate the amount of MAC operations required to calculate one output pixel ($n \times m \times (k^2 + 1)$) and multiply it by the accelerator frequency. To calculate the OPS/bit for each design, we divide the amount of MAC operations in the layer by the total number of bits transferred over the memory bus, which includes the weights, the input and the output activations. The layer requires 524,288 TOPS/s to be calculated without stalling for memory access and computation. The available performance of the accelerators is summarized in Table 7.2 and visualised using the proposed OPS-based roofline analysis in Fig. 7.8.

In this example, the application’s requirements are out of the scope of the product definition. On one hand, all accelerators are computationally bound (all horizontal lines are below the application’s requirements), indicating that we do not have enough PEs to calculate the layer in one run. On the other hand, even if we decide to increase the computational density by using stronger quantization or by increasing the silicon area (and the cost of the accelerator), we would still hit the memory bound (represented by the diagonal line). In this case, the solution should be found at the algorithmic level or by changing the product’s targets; e.g., we can calculate the layer in parts, increase the silicon area of while decreasing the frequency in order not to hit memory wall, or
Table 7.3: The amount of computation (OPS/s) provided by the accelerators and memory throughput (OPS/bit) required by second layer of ResNet-18.

<table>
<thead>
<tr>
<th></th>
<th>32-bit float</th>
<th>32-bit fixed</th>
<th>16-bit quant.</th>
<th>8-bit quant.</th>
<th>4-bit quant.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOPS/s</td>
<td>49.00</td>
<td>324.0</td>
<td>1296</td>
<td>3969</td>
<td>11236</td>
</tr>
<tr>
<td>OPS/bit</td>
<td>9.16</td>
<td>9.16</td>
<td>18.32</td>
<td>36.64</td>
<td>73.27</td>
</tr>
</tbody>
</table>

Figure 7.9: OPS roofline: $3 \times 3$ kernel, 64 input and output $56 \times 56$ features, $6mm^2$ accelerator with 100MHz frequency, with DDR of 2.4GHz with 64 bit data bus.

decide to use another algorithm.

### 7.4.3 Roofline Example 2 - Optimal

Our second example explores the feasibility of implementing the second layer of ResNet-18 that has a $3 \times 3$ kernel and 64 input and output features of dimension $56 \times 56$. For this example, we increase the silicon area to $6mm^2$ and lower the frequency to 100MHz, as proposed earlier, and add a 4-bit quantized accelerator for comparison purposes. The layer requires 4.1 GOPS/s. The accelerators results are summarized in Table 7.3 and visualised with the OPS-based roofline analysis in Fig. 7.9.

From Fig. 7.9 we can see that our 32-bit and 16-bit accelerators are still computationally bound, while the 8-bit and 4-bit quantized accelerators meet the demands of the layer. In particular, the 8-bit accelerator is located at the border of computational ability, meaning this solution has nearly optimal resource allocation, since the hardware is fully utilized. Still, the final choice of the configuration depends on other parameters such as the accuracy of the CNN.

Both examples demonstrate that decisions made at early stages have a critical impact on the quality of the final product. For example, applying an aggressive quantization to the network or increasing the silicon size may not improve the overall per-
formance of the chip if its performance is memory-bound. From the architect’s point of view, it is important to balance between the computation and data transfer. Nonetheless, this balance can be achieved in different ways: at the micro-architecture level, at the algorithmic level or by changing the data representation. The architect may also consider (1) changing the hardware to provide faster communication (requires more power and is more expensive), (2) applying communication bandwidth compression algorithms [CBB+19], (3) using fewer number of bits to represent weights and activations (using 3- or 4-bit representation may solve the communication problem, at the cost of reducing the expected accuracy), or (4) changing the algorithm to transfer data slower (even though that solves the bandwidth issue, the possible drawback is a reduced throughput of the whole system). The proposed OPS-based roofline model helps the architect to choose alternative. After making major architectural decisions we can use BOPS to get an estimation of the impact of different design choices on the final product, such as the expected area, power, optimal operational point, etc. The next chapter will examine these design processes from the system design point of view.

7.5 HCM Metric evaluation

After introducing the use of BOPS as a metric for the hardware complexity of CNN-based algorithms and the use of the OPS-based roofline model to help the architect understand how the decisions at the algorithmic level may impact the characterizations of the final product, this section provides a holistic view of the design process of systems with CNN accelerators. We conducted an extensive evaluation of the design and the implementation of a commonly used CNN architecture for ImageNet [RDS+15] classification, ResNet-18. We also compare our metric to prior art [MM18] in terms of correspondence between complexity score to hardware utilization for CNN parameters with various bitwidths.

7.5.1 Experimental methodology

We start the evaluation of the HCM metric with a comprehensive review of the use of BOPS as part of the design and implementation process of a CNN accelerator. This section shows the trade-offs involved in the process and verifies the accuracy of the proposed model. It focuses on the implementation of a single PE since PEs are directly affected by the quantization process. The area of an individual PE depends on the chosen bitwidth, while the change in the amount of input and output features changes both the required number of PEs and the size of the accumulator. The leading example we use implemented an all-to-all CNN accelerator that can calculate \( n \) input features and \( m \) output features in parallel, as depicted in Fig. 7.10. For simplicity, we choose an equal number of input and output features. In this architecture, all the input features are routed to each of the \( m \) blocks of PEs, each calculating a single output.
feature. The implementation was done for an ASIC using the TSMC 28nm technology library, 800MHz system clock and in the nominal corner of V_{DD} = 0.81V. For the power analysis, input activity factor, and sequential activity factor, we used the value of 0.2. The tool versions are listed in Table 7.4.

Table 7.4: CAD Design Tools

<table>
<thead>
<tr>
<th>Language</th>
<th>Verilog HDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Simulation</td>
<td>ModelSim 19.1</td>
</tr>
<tr>
<td>Synthesis</td>
<td>Synopsys Design Compiler 2017.09-SP3</td>
</tr>
<tr>
<td>Place and route</td>
<td>Cadence Innovus 2019.11</td>
</tr>
</tbody>
</table>

For brevity, we present only the results of experiments at 800 MHz clock frequency. We performed additional experiments at 600 MHz and 400 MHz (obviously, neither BOPS nor the area of an accelerator depends on the chip frequency), but do not show these results. As shown in Section 7.4, lowering the frequency of the design can help to avoid the memory bound, but incurs the penalty of slower solution time.

Our results show a high correlation between the area of the design and BOPS. The choice of an all-to-all topology shown in Fig. 7.10 was made because of an intuitive understanding of how the accelerator calculates the outputs of the network. This choice, however, has a greater impact on the layout’s routing difficulty, with various alternatives such as broadcast or systolic topologies [Che16]. For example, a systolic topology, a popular choice for high-end NN accelerators [JYP+17], eases the routing complexity by using a mesh architecture. Although it reduces the routing effort and improves the flexibility of the input/output feature count, it requires a more complex control for the data movement to the PEs.

To verify the applicability of BOPS to different topologies, we also implemented a systolic array shown in Fig. 7.11, where each 1 × 1 PE is connected to 4 neighbors with the ability to bypass any input to any output without calculations. The input feature accumulator is located at the input of the PE. This topology generates natural...
4 \times 1 \text{ PEs}, but with proper control, it is possible to create flexible accelerators. In the systolic design, we generated three square arrays, 4 \times 4, 8 \times 8, and 16 \times 16, with b_w = b_a \in \{4, 6\}. The systolic array area was found to be in linear relation with BOPS, with the goodness of fit $R^2 = 0.9752$.

### 7.6 System-level design using HCM

In this section, we analyze the acceleration of ResNet-18 using the proposed metrics and show the workflow for early estimation of the hardware cost when designing an accelerator. We start the discussion by targeting an ASIC that runs at 800MHz, with 16 \times 16 \text{ PEs} and the same 2.4GHz DDR-4 memory with a 64-bit data bus as used in Section 7.4. The impact of changing these constraints is discussed at the end of the section. For the first layer, we replace the 7 \times 7 \text{ convolution} with three 3 \times 3 \text{ convolutions}, as proposed by [HWZ+16]. This allows us to simplify the analysis by
employing universal $3 \times 3$ PEs for all layers.

We start the design process by comparing different alternatives using the new proposed OPS-based-roofline analysis since it helps to explore the design trade-offs between the multiple solutions. We calculate the amount of OPS/s provided by $16 \times 16$ PEs at 800MHz and the requirements of each layer. To acquire the roofline, we need to calculate the OPS/bit, which depend on the quantization level. For ResNet-18, the current art [GLJ+19] achieves 69.56% top-1 accuracy on ImageNet for 4-bit weights and activations, which is only 0.34% less than 32-bit floating-point baseline (69.9%). Thus we decided to focus on 2-, 3- and 4-bit quantization both for weights and activations, which can achieve 65.17%, 68.66%, and 69.56% top-1 accuracy, correspondingly.

For a given bitwidth, OPS/bit is calculated by dividing the total number of operations by the total number of bits transferred over the memory bus, consisting of reading weights and input activations and writing output activations. Fig. 7.13 presents OPS-based roofline for each quantization bitwidth. Please note that for each layer we provided two points: the red dots are the performance required by the layer, and the green dots are the equivalent performance using partial-sum computation.

Fig. 7.13 clearly indicates that this accelerator is severely limited by both computational resources and lack of enough bandwidth; the system is computationally bounded, which could be inferred from the fact that it does not have enough PEs to calculate all the features simultaneously. Nevertheless, the system is also memory-bound for any quantization level, meaning that adding more PE resources would not solve the problem. It is crucial to make this observation at the early stages of the design since it means that micro-architecture changes would not be sufficient to solve the problem.

One possible solution, as presented in Section 7.4, is to divide the channels of the input and output feature maps into smaller groups, and use more than one clock cycle to calculate each pixel. In this way, the effective amount of the OPS/s required for the layer is reduced. In the case that the number of feature maps is divisible by the number of available PEs, the layer will fully utilize the computational resources, which is the case for every layer except the first one. Reducing the number of PEs, however, also reduces
the data efficiency, and thus the OPS/bit also decreases, shifting the points to the left on the roofline plot. Thus, some layers still require more bandwidth from the memory than what the latter can supply. In particular, in the case of 4-bit quantization, most of the layers are memory-bound. The only option that properly utilizes the hardware is 2-bit quantization, for which all the layers except one are within the memory bound of the accelerator. Another option for solving the problem is to either change the neural network topology being used, or add a data compression scheme on the way to and from the memory [CBB+19]. Adding compression will reduce the effective memory bandwidth requirement and allow adding more PEs in order to meet the performance requirements – at the expense of cost and power.

At this point, BOPS can be used to estimate the power and the area of each alternative for implementing the accelerator using the PE micro-design. In addition, we can explore other trade-offs, such as the influence of modifying some parameters that were fixed at the beginning: lowering the ASIC frequency will decrease the computational bound, which reduces the cost and only hurts the performance if the network is not memory-bound. An equivalent alternative is to decrease the number of PEs. Both procedures will reduce the power consumption of the accelerator as well the computational performance. The system architect may also consider changing the parameters of the algorithm being used, e.g., change the feature sizes, use different quantization for the weights and for the activations, include pruning, and more.

It is also possible to reverse design order: start with a BOPS estimate of the number of PEs that can fit into a given area, and then calculate the ASIC frequency and memory bandwidth that would allow full utilization the accelerator. This can be especially useful if the designer has a specific area or power goal.

To summarize this section, from the architecture point of view it is extremely important to be able to predict, at the early stages of the design, if the proposed (micro)architecture is going to meet the project targets. At the project exploration stage, the system architect has plenty of alternatives to choose from to make the right trade-offs or even negotiate to change the product definition and requirements. Introducing such alternatives later may be painful or even near to impossible.

### 7.6.1 HCM evaluation of Eyeriss Architecture

In this section we would like to evaluate an existing CNN hardware architecture using the our modified roofline analysis. We are going to evaluate the Eyeris [Che16] implementation of VGG-15. As before we plotted the required performance of each layer and we can see in Fig. 7.14 the needed performance, the red dots, are only going down on the roofline when calculating in cycles, as opposed to what we showed in 7.6, because of the different architectural structure. In our example, we used a weight stationery, which has an overhead when calculating in cycles for reading the input features for each set of weights. In Eyeriss architecture, they use row stationery and have enough local
memory for re-using all the weights and activations before reading additional data. By doing this, they are avoiding the overhead of re-reading the activations for each set of weights. The roofline analysis shows asymptotic PE performance and external memory bandwidth, so features like compression data drop are not shown.

Our analysis shows that using VGG-16 on Eyeriss hardware has some layers that will be memory bound. Most of the layers that are close to the memory bound may be mitigated using the proposed [Che16] compression methods and weights/activations drop that was proposed, But the first and the three last layers will suffer from poor performance compared with other layers. These results should be an input to the architect. The decision can be made that these layers will suffer from performance
Figure 7.15: Comparison of BOPS and “compute cost” [MM18] predictive power. BOPS – 5% error, “compute cost” – 15%.

degradation, but this is tolerable because other layers can perform efficiently. The main benefit of using the roofline analysis is that we can know in advance the areas where we are not optimal and may need to put the attention in the design or not. It is up to the architect of the hardware.

7.6.2 Comparison with prior metrics

In this section, we compare the BOPS [BSZ+18] metric to another complexity metric, introduced by [MM18]. A good complexity metric should have a number of properties. First, it should reflect the real cost of the design. Second, it should be possible to calculate it from micro-designs or prior design results, without needing to generate complete designs. Last, it should generalize well, providing meaningful predictions for a wide spectrum of possible design parameter values. We compare our choice of computational complexity assessment, BOPS, with the “compute cost” proposed by [MM18]. To analyze the metrics, we use our real accelerator area results from Section 7.5 and error bands of linear extrapolation of the measured values. To remind the reader, BOPS and “compute cost” are defined as follows:

\[
\text{BOPS} = mnk^2\left(b_a b_w + b_a + b_w + \log_2(nk^2)\right) \quad (7.3)
\]

\[
\text{compute\_cost} = mnk^2(b_a + b_w) \quad (7.4)
\]

The error of predicting a new point with “compute cost” is 15% within 2 orders
of magnitude, whereas using BOPS, is only 5%. As shown in Fig. 7.15, “compute cost” introduces a systematic error: each of the distinguishable groups of three points corresponding to a single value of the number of input and output features creates a separate prediction line. This may lead to higher errors in case of extrapolation from a single value of the number of input and output features or a wide range of the considered bitwidth.
7.7 Discussion

CNN accelerators are commonly used in different systems, starting from IoT and other resource-constrained devices, and ending in datacenters and high-performance computers. Designing accelerators that meet tight constraints is still a challenging task, since the current EDA and design tools do not provide enough information to the architect. To make the right choice, the architects need to understand at the early stages of the design the impact of high-level decisions they make on the final product, and to be able to make a fair comparison between different design alternatives.

In this work, we showed that one of the fundamental shortcomings of the current design methodologies and tools is the use of GFLOPS as a metric for estimating the complexity of existing hardware solutions. The first contribution of this work is the definition of the HCM as a metric for hardware complexity. We demonstrated its application to the prediction of such product characteristics as power, performance, etc.

The second contribution of the work is the introduction of the OPS-based roofline model as a supporting tool for the architect at the very early stages of the development. We showed that this model allows the comparison of different alternatives of the design and the determination of the optimality and feasibility of the solution.

Lastly, we provided several examples of realistic designs, using an actual implementation with standard design tools and a mainstream process technology. By applying the proposed metric, we could build a better system and indicate to the system architect that certain CNN architectures may better fit the constraints of a specific platform. In particular, our metric confirmed that CNN accelerators are more likely to be memory, rather than computationally bound [WDCC19, JYP+17].

Although this work is mainly focused on ASIC-based architectures, the same methodology can be applied to many other systems, including FPGA-based implementations and other system-specific domains that allow trading off accuracy and data representation with different physical parameters such as power, performance, and area.
Chapter 8

Adversarial robustness via randomization

8.1 Motivation

Deep neural networks are known to be vulnerable to malicious perturbations. Current methods for improving adversarial robustness make use of either implicit or explicit regularization, and the latter is usually based on adversarial training. Randomized smoothing, the averaging of the classifier outputs over a random distribution centered in the sample, has been shown to guarantee a classifier’s performance subject to bounded perturbations of the input. In this work, we study the application of randomized smoothing to improve performance on unperturbed data and increase robustness to adversarial attacks. We propose to combine smoothing along with adversarial training and randomization approaches, and find that doing so significantly improve over the baseline. We examine its performance on common white-box (FGSM, PGD) and black-box (transferable attack and NAttack) attacks on CIFAR-10 and CIFAR-100, and determine that for a low amount of iterations, smoothing provides a significant performance boost which persists even for perturbations with high attack norm, $\epsilon$. For example, under a PGD-10 attack on CIFAR-10 using Wide-ResNet28-4, we achieve 60.3% accuracy for infinity norm $\epsilon_\infty = \frac{8}{255}$ and 13.1% accuracy for $\epsilon_\infty = \frac{35}{255}$. Hereby outperforming previous art by 3% and 6% respectively, achieving nearly twice the accuracy on $\epsilon_\infty = \frac{35}{255}$ and even more so for perturbations with higher infinity norm.
8.2 Introduction

Deep neural networks (DNNs) are showing spectacular performance in a variety of computer vision tasks, but at the same time are susceptible to adversarial examples – small perturbations that alter the output of the network [SZS+13, GSS14]. Since the initial discovery of this phenomenon in 2013, increasingly stronger defenses [GSS14, MMS+18, XWM+19, SGI19, KHM19, ZL19, HRF19, ZYJ+19] and counterattacks [GSS14, CW17, ACW18, MMS+18, RHO+19, PMG+17, CZS+17, LLW+19] were proposed in the literature. Adversarial attacks have also been shown to occur in tasks beyond image classification where they were first discovered: in real-life object recognition [BMR+17, XZL+19, AEIK18], object detection [WLW+19], natural language processing [GLSQ18, CKG19, JJZS19], reinforcement learning [GDK+19], speech-to-text [CW18], and point cloud classification [XQL19], just to mention a few. Moreover, the adversarial examples can be used to improve the performance of the DNNs on unperturbed data [XTG+19, GRYL20, SWC+20].

Understanding the root cause of adversarial examples, how they are created, and how we can detect and prevent such attacks, is at the center of many research works. [GMF+18] argued that adversarial examples are an inevitable property of high-dimensional data manifolds rather than a weakness of specific models. Given this, an adversarial defense’s real goal is not to get rid of adversarial examples, but rather to make their search difficult.

Current defense methods are based on either implicit or explicit regularization. Explicit regularization methods aim to increase the performance under adversarial attack by directly incorporating a suitable term into the loss of the network during training, usually by incorporating adversarial examples for the dataset used in the training process. In contrast, implicit regularization methods that do not change the objective, such as variational dropout [KSW15], seek to train the network to be robust against any perturbations without taking into account adversarial examples. [FGCC19] showed that there is a strong connection between robustness to random noise corruption and adversarial robustness. Implicit regularization seeks to exploit this connection to achieve adversarial robustness. In particular, adding randomness to the network can be especially successful [LCZH18, BMCM18, HRF19], since information acquired from previous runs cannot be directly applied to a current run. Another way to utilize randomness to improve classifier robustness is through randomized smoothing [CRK19]: averaging the outputs of the classifier over some random distribution centered in the input data point. The effects of these three approaches (explicit regularization, implicit regularization, and smoothing) do not necessarily line up with or contradict each other. Thus, one could use a combination of the three when devising adversarial defenses.

Previous works that discuss randomized smoothing do so exclusively in the context of certified robustness [CRK19, SYL+19]. While certification is a very strong and desired guarantee, researchers were unable to achieve a practical \( \ell_{\infty} \) certification radii,
and it is unclear whether this is at all possible [BDMZ20, YDH+20, KLGF20]. Nevertheless, smoothing may still be employed as a practical method to improve empirical classifier performance. Therefore, we investigate the effect of smoothing for both the clean accuracy and adversarial robustness of the model. We show this effect on top of adversarial regularization methods – both implicit [HRF19, ZBN+20] and explicit [MMS+18]. Below, we outline our main contributions.

Firstly, we study the effect of randomized smoothing on empirical accuracy of adversarially trained classifiers, both on perturbed and clean data. We show that even for a small amount of samples, adversarial accuracy increases with no degradation of the clean accuracy. Also, since the performance grows with the sample size, smoothing introduces a trade-off between inference time complexity and accuracy. While this effect is present even for higher attack radii, it decreases with number of iterations, suggesting that at least some of the impact may be attributed to random gradient obfuscation.

Secondly, in addition to the combination of smoothing with adversarial training proposed by [SYL+19], we use it in tandem with methods that utilize implicit regularization based on noise injection. Combining this regularization with smoothing resulted in a substantial increase in the smooth classifier’s performance on both perturbed and clean data, achieving state-of-the-art results in various configurations. In addition, we discuss several smooth inference methods and ways to optimize a pre-trained adversarial model to improve the accuracy of the smooth classifier.

Lastly, we show a relation between training the smooth classifier and training the base model under perturbation. A similar relation was previously demonstrated for the cross-entropy loss by [CRK19]. We, on the other hand, show a relation for the 0-1 loss under weaker assumptions.
8.3 Randomized Smoothing and Implicit Regularization

In what follows, we start with definition of inference smoothing and propose three methods for the aggregation of the samples: prediction smoothing, soft prediction smoothing and weighted prediction smoothing. Finally, we show a relation between optimization of the smooth classifier and optimization of the base classifier under implicit regularization. We do not discuss adversarial perturbations in this section.

8.3.1 Inference smoothing

A smooth classifier $\tilde{f}_{b^*\theta}$ is a map assigning to an input $bx$ the class label that the base classifier $f_{b^*\theta}$ is most likely to return for $bx$ under random perturbation $\eta$.

$$\tilde{f}_{b^*\theta}(bx) = \arg \max_{y \in \mathcal{Y}} P_{\eta}[f_{b^*\theta}(bx + \eta) = y] = \arg \max_{y \in \mathcal{Y}} \int_{\mathbb{R}^n} d\eta p(\eta) I[f_{b^*\theta}(bx + \eta) = y] = \arg \max_{y \in \mathcal{Y}} \int_{\mathbb{R}^n} d\eta p(\eta) I[\arg \max_{y' \in \mathcal{Y}} f_{b^*\theta}(bx + \eta)]_{y'} = y], \quad (8.1)$$

$$= \arg \max_{y \in \mathcal{Y}} \int_{\mathbb{R}^n} d\eta p(\eta) \mathbb{I}[\arg \max_{y' \in \mathcal{Y}} f_{b^*\theta}(bx + \eta)]_{y'} = y', \quad (8.2)$$

where $b^*\eta$ is some random vector, $p$ is its density function, $[f_{b^*\theta}(bx + \eta)]_{y'}$ is $y'$ element of probability (or, equivalently, logit) vector output of $f_{b^*\theta}$. We now generalize $\tilde{f}_{b^*\theta}$ for the smoothing to be over the logits of the classifier instead of just the prediction. Let the resulting classifier be $\hat{f}_{b^*\theta}(bx)$ for some aggregation function $A_y$.

$$\tilde{f}_{b^*\theta}(bx) = \arg \max_{y \in \mathcal{Y}} \int_{\mathbb{R}^n} d\eta p(\eta) A_y(f_{b^*\theta}(bx + \eta)), \quad (8.3)$$

Notice that we can acquire Eq. (8.3) from Eq. (8.4) by taking $A_y$ to be the following indicator function:

$$A_y(f_{b^*\theta}(bx + \eta)) = \mathbb{I}[f_{b^*\theta}(bx + \eta) = y], \quad (8.5)$$

Since the integral (8.4) is intractable, it is usually approximated using the Monte Carlo method, i.e., by averaging over a number of points from the distribution sampled independently. We denote by $M$ the number of samples used for Monte Carlo approximation.

Under this approximation, the most general form of the smooth model output is

$$\hat{f}_{b^*\theta}(bx) \approx \arg \max_{y \in \mathcal{Y}} \sum_{i=1}^{M} V_y(p_i(bx)), \quad (8.6)$$

where $V_y$ is some aggregation function (corresponding to aggregation function $A$ in
Eq. (8.5), and we denote:

\[ p_i(bx) = f_{b \theta}(bx + \eta_i), \quad (8.7) \]

\[ p_{iy}(bx) = [f_{b \theta}(bx + \eta_i)]_y. \quad (8.8) \]

We propose three different implementations of smoothing, differing in their aggregation function \( V_y \): prediction smoothing, soft prediction smoothing and weighed prediction smoothing.

**Prediction smoothing.**

The simplest possible way to aggregate prediction is to perform prediction voting, i.e., to output the most frequent prediction among the samples, or, equivalently, to approximate Eq. (8.3) directly with the Monte Carlo method. In this case, \( V_y \) is the following indicator function:

\[ V_y(p_i) = \mathbb{1}\left[ \arg \max_{y' \in \mathcal{Y}} p_{iy'} = y \right]. \quad (8.9) \]

yielding the following smooth classifier

\[ \hat{f}^\mathcal{V}_y(bx) = \arg \max_{y \in \mathcal{Y}} \sum_{i=1}^M \mathbb{1}\left[ \arg \max_{y' \in \mathcal{Y}} p_{iy'} = y \right]. \quad (8.10) \]

In this form, randomized smoothing was previously discussed by [CRK19] and [SYL+19] for certified robustness. It is important to emphasize that this aggregation scheme only considers the classification of each of the \( M \) samples, discarding the predicted probabilities of each class. In what follows, we use this aggregation method unless otherwise specified. In Section 9.4 we show that in various configurations, taking \( M \) predictions into account is a viable method to increase the accuracy of the classifier on both clean and adversarially-perturbed inputs.

**Soft prediction smoothing.**

Alternatively, we can calculate the probability expectation for each class and use the results for prediction generation. In this case, we take \( V_y \) to be the softmax function:

\[ V_y(p_i) = [\text{softmax}(p_i)]_y, \quad (8.11) \]

yielding the following smooth classifier

\[ \hat{f}^\mathcal{V}_y(bx) = \arg \max_{y \in \mathcal{Y}} \sum_{i=1}^M [\text{softmax}(f_{b \theta}(bx + \eta_i))]_y. \quad (8.12) \]

This method was previously mentioned by [SYL+19] as a way to apply adversarial
training to a smooth classifier. Since the probabilities for each class are now differentiable, this allows to train the classifier end-to-end, using smooth predictions. As opposed to prediction smoothing, we now fully consider the predicted class probabilities of each of the $M$ samples.

However, soft smoothing is a weaker adversarial defense than prediction smoothing. Even if the attack is not successful, the probability of competing classes increases, which means that even an unsuccessful attack on a base model impacts the prediction of the smooth model. Therefore, we do not use this kind of aggregation.

**Weighted prediction smoothing**

The last aggregation method we propose is a compromise between the two methods previously presented, generalizing both by assigning some weight to top-k predictions.

Let us denote by $k_y(p_i)$ the rank of class $y$ as predicted by $f_{b*\theta}(bx + b*\eta_i)$, i.e., $k_y = 1$ for the most probable class, $k_y = 2$ for the second most probable one, and so on. Then, for example, the following choices of $V_y$ are possible:

$$V_y(p_i) = 2^{1-k_y(p_i)}, \text{ or }$$

$$V_{C,y}(p_i) = \begin{cases} 1 & k_y(p_i) = 1 \\ C & k_y(p_i) = 2 \\ 0 & \text{otherwise.} \end{cases}$$

yielding the following smooth classifiers,

$$\tilde{f}_{V,b*\theta}(bx) = \arg\max_{y \in Y} \sum_{i=1}^{M} 2^{1-k_y(p_i)}, \text{ or }$$

$$\tilde{f}_{V,C*,\theta}(bx) = \arg\max_{y \in Y} \sum_{i=1}^{M} 1[k_y(p_i) = 1] + C \cdot 1[k_y(p_i) = 2],$$

Note that for $C = 0$, Eq. (8.16) becomes prediction smoothing (8.10).

In particular, $V_{C,y}$ expresses the dependency on the second prominent class shown in Theorem 1 of [CRK19]. Let $f_{b*\theta} : \mathbb{R}^d \rightarrow Y$ be any deterministic or random function, $\tilde{f}_{V,b*\theta}(bx)$ be defined as in Eq. (8.10) with $b*\eta \sim \mathcal{N}(0, \sigma^2 I)$, and let $C_A, C_B$ be indices of the first and second prominent classes accordingly, then certified $L_2$ radius is given by

$$R = \frac{\sigma}{2} (\Phi^{-1}([f_{b*\theta}(bx + \eta)]_{C_A}) - \Phi^{-1}([f_{b*\theta}(bx + \eta)]_{C_B}))$$

$$\tilde{f}_{V,b*\theta}(bx + \delta) = C_A, \forall \|\delta\|_2 < R,$$

where $\Phi^{-1}$ is the inverse of the standard Gaussian CDF.
8.3.2 Relation to implicit regularization

We now show that optimizing the smooth classifier is strongly related to optimizing the base classifier under implicit regularization. This relation is our motivation for investigating the effect of smoothing over a CNI base model.

In the case of prediction smoothing (or other cases in which $V_y$ is not differentiable), we cannot train the smooth model directly. Therefore, we would like to train the base model to optimize the loss of the smooth model. For this, we estimate the loss function of the smooth classifier with an expression that can be calculated at training time.

To this end, we use the 0-1 loss of $n$ training samples. Denoting $\tilde{f}_{b*\theta}$ as in Eq. (8.3),

$$L_{01} = \sum_{i=1}^{n} \ell_{01}(y_i, \tilde{f}_{b*\theta}(bx_i)), \quad (8.19)$$

with the pointwise terms

$$\ell_{01}(y_i, \tilde{f}_{b*\theta}(bx_i)) = 1 - 1 \left[ y_i = \arg \max_{y \in \mathcal{Y}} P_y[f_{b*\theta}(bx_i + b*\eta) = y] \right], \quad (8.20)$$

is minimized over the model parameters $b*\theta$. Denoting for brevity $P_y = P_y[f_{b*\theta}(bx_i + b*\eta) = y]$, we can approximate the indicator as

$$1 \left[ y_i = \arg \max_{y \in \mathcal{Y}} P_y \right] = 1 \left[ P_{y_i} \geq \max_{y' \in \mathcal{Y} \setminus \{y_i\}} P_{y'} \right] \approx \frac{1}{2} \left[ 1 + P_{y_i} - \max_{y' \in \mathcal{Y} \setminus \{y_i\}} E_{\eta}[1_{y'=y_i}] \right], \quad (8.21)$$

where we approximate the Heaviside function $1[x > c]$ on an interval $[0, 1]$ with a better-behaving linear function $(x-c+c)/2$. The expression in Eq. (8.22) resembles the bound that [CRK19] has suggested for the radius of certification under adversarial attacks, which is previously presented in Eq. (8.17).

We now show a relation to training the base model under perturbation, similarly denoting $1_y = 1[f_{b*\theta}(bx_i + b*\eta) = y]$:

$$\ell_{01}(y_i, \tilde{f}_{b*\theta}(bx_i)) \approx 1 - \frac{1}{2} \left[ 1 + E_{\eta}[1_{y_i}] - \max_{y' \in \mathcal{Y} \setminus \{y_i\}} E_{\eta}[1_{y'}] \right] = \frac{1}{2} \left[ 1 - E_{\eta}[1_{y_i}] + \max_{y' \in \mathcal{Y} \setminus \{y_i\}} E_{\eta}[1_{y'}] \right]. \quad (8.23)$$

Written in this form, the 0-1 loss is now amenable to Monte Carlo approximation; however, working with a non-convex loss is still problematic. We, therefore, bound the

111
maximum over the expectation by the expectation of the maximum,

\[
\ell_{01}(y_i, \tilde{f}_{b*\theta}(bx_i)) \leq \frac{1}{2} \left[ 1 - E_{\eta}[1_{y_i}] + E_{\eta} \left[ \max_{y' \in Y \setminus \{y_i\}} 1_{y'} \right] \right] \\
= \frac{1}{2} E_{\eta} \left[ 1 - 1_{y_i} + \max_{y' \in Y \setminus \{y_i\}} 1_{y'} \right].
\]  

(8.24)

Since the classification events are disjoint, only one of \(1_{y'}\) is non-zero and thus we can replace the maximum over the indicators with a sum:

\[
\max_{y' \in Y \setminus \{y_i\}} 1_{y'} = \sum_{y' \in Y \setminus \{y_i\}} 1_{y'} = 1 - 1_{y_i},
\]

(8.25)

acquiring

\[
\ell_{01}(y_i, \tilde{f}_{b*\theta}(bx_i)) \leq \frac{1}{2} E_{\eta}[1 - 1_{y_i} + 1 - 1_{y_i}] = \]  

(8.26)

\[
= E_{\eta}[1 - 1_{y_i}] = E_{\eta}[\ell_{01}(y_i, f_{b*\theta}(bx_i + b \ast \eta))],
\]

(8.27)

which is the 0-1 loss of the base classifier under Gaussian perturbation. In addition, for the \(\ell\)-th layer, we can rewrite the network inference as

\[
f(bx_i) = f_2 \circ f_1(bx_i) = f_2(bx'_i),
\]

(8.28)

where \(f_1\) denotes the first \(\ell - 1\) layers and \(f_2\) stands for the rest of the network. Repeating the computation for \(f_2\) and \(bx'_i\) instead of \(f\) and \(bx_i\) shows that implicit regularization in the form of injecting Gaussian noise to the activation in any layer should help to minimize the loss of the smooth classifier. Thus, we study an application of smoothing to CNI [ZBN+20] – a family of models that inject noise as an adversarial defense, expecting that both base model and smoothing would benefit from noise injection.
Table 8.1: Results on CIFAR-10 with ResNet-20 under PGD attack. Smoothed models use \( M = 512 \). \( \sigma = 0.24 \) for Smooth CNI, \( \sigma = 0.3 \) for smooth adversarial training, and \( \sigma = 0.00 \) for the no-noise settings in both cases. For comparison with [SYL+19] we applied the smoothing to their best-performing checkpoint provided by the authors, on ResNet-110. † denotes our results based on code provided by the authors or our re-implementation. + denotes our results based on the checkpoint provided by the authors.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy, %</th>
<th>Clean</th>
<th>PGD-7</th>
<th>PGD-10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adversarial training [MMS+18]†</td>
<td></td>
<td>83.04</td>
<td>42.44</td>
<td>36.76</td>
</tr>
<tr>
<td>PNI [HRF19]</td>
<td></td>
<td>84.89</td>
<td>45.94</td>
<td>43.24†</td>
</tr>
<tr>
<td>CNI [ZBN+20]</td>
<td></td>
<td>78.48</td>
<td>48.84</td>
<td>46.52</td>
</tr>
<tr>
<td>IAAT (trained on k=10) [BGH19]</td>
<td>87.26</td>
<td>-</td>
<td>43.08</td>
<td></td>
</tr>
<tr>
<td>Randomized smoothing [SYL+19]+</td>
<td></td>
<td>67.92</td>
<td>45.88</td>
<td>45.32</td>
</tr>
<tr>
<td>Smooth adversarial training (no noise)</td>
<td></td>
<td>83.04</td>
<td>42.61</td>
<td>36.85</td>
</tr>
<tr>
<td>Smooth adversarial training</td>
<td></td>
<td>81.64</td>
<td>49.63</td>
<td>43.49</td>
</tr>
<tr>
<td>Smooth CNI (no noise)</td>
<td></td>
<td>84.46</td>
<td>53.31</td>
<td>49.79</td>
</tr>
<tr>
<td>Smooth CNI</td>
<td></td>
<td>81.98</td>
<td>56.20</td>
<td>53.10</td>
</tr>
</tbody>
</table>

8.4 Experiments

We now present an empirical evaluation of the proposed method. We start from a comparison of our method to previous art in multiple settings, followed by an extensive ablation study both of the smooth attacks and fine-tuning methods.

To investigate the effectiveness of the proposed method, we conducted experiments on CIFAR-10 and CIFAR-100 [Kri09b] under white-box and black-box attacks for ResNet-20 [HZRS15], Wide-ResNet28-4, Wide-ResNet28-10 and Wide-ResNet34-10 [ZK16].

**Experimental settings.** We considered two different base models: adversarially-trained CNNs [MMS+18] and CNI [ZBN+20]. While smoothing applied to adversarially-trained models improved adversarial robustness [SYL+19], using CNI as base model provides significantly larger performance gains.

For ResNet-20 on CIFAR-10, we used the CNI as base model trained adversarially for 400 epochs under PGD attack with \( k = 7 \). For Wide-ResNet28-4 on both CIFAR-10 and CIFAR-100, we used the CNI model trained adversarially for 100 epochs under PGD attack with \( k = 10 \). In all cases, we used the adversarial training suggested by [MMS+18] and chose the model with the highest adversarial performance on the validation set.

8.4.1 Comparison to other adversarial defenses

In each case, we compared our best-performing defense instance without fine-tuning (smooth CNI) to the current state-of-the-art. We do not include the fine-tuned models
Table 8.2: Results on CIFAR-100 with Wide-ResNet28-4 (top part), Wide-ResNet28-10 (middle) and Wide-ResNet34-10 (bottom) under FGSM and PGD attacks. Smoothing uses $M = 64$ with $\sigma = 0.04$ for Smooth CNI, and $\sigma = 0.02$ for smooth adversarial training.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy, %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Clean</td>
</tr>
<tr>
<td>Adversarial training [MMS$^{+}$18]$^\dagger$</td>
<td>56.6</td>
</tr>
<tr>
<td>Smooth adversarial training</td>
<td>56.17</td>
</tr>
<tr>
<td>PNI [HRF$^{+}$19]$^\dagger$</td>
<td>54.63</td>
</tr>
<tr>
<td>CNI [ZBN$^{+}$20]$^\dagger$</td>
<td>53.46</td>
</tr>
<tr>
<td>Smooth CNI (ours)</td>
<td>61.73</td>
</tr>
<tr>
<td>Feature Matching [LFZ$^{+}$20]</td>
<td>59.51</td>
</tr>
<tr>
<td>IAAT [BGH19]</td>
<td><strong>68.1</strong></td>
</tr>
<tr>
<td>Free [SNG$^{+}$19]</td>
<td>62.13</td>
</tr>
</tbody>
</table>

In main comparison as they were trained on attacks with higher complexity than the current state-of-the-art. For all the results, unless stated otherwise, we use the $\ell_\infty$ radius of attack $\epsilon = 8/255$.

For ResNet-20 on CIFAR-10, our smooth CNI demonstrated a consistent improvement of $\sim7\%$ over the CNI base model, which is the best existing method for this case (Table 8.1), in addition our smooth CNI substantially outperformed the Randomized smoothing [SYL$^{+}$19] baseline by $\sim10.3\%$. For Wide-ResNet28-4 on CIFAR-10, our smooth CNI showed a consistent improvement of $\sim5\%$ over the CNI base model, outperforming the best existing method by $\sim3\%$ for $k = 10$ (Table 8.3). For Wide-ResNet on CIFAR-100, the effect of smoothing diminishes for higher $k$, with our smooth CNI substantially outperforming prior art for FGSM, but falling behind for PGD with $k = 10, 20$ (Table 8.2).

Next, we study the effect of the value of $\epsilon$ on the PGD attack. Fig. 8.1 demonstrates resilience of the smoothing effect for high-$\epsilon$ attacks with a CNI base model. The degradation of existing models is significantly stronger for $\epsilon > 20/255$. For example, smooth CNI achieves 13.1% accuracy for $\epsilon = 35/255$ and 5.3% accuracy for $\epsilon = 45/255$, outperforming current state-of-the-art by 6% and 3.7% respectively. Moreover, similar effect appears of low values of $\epsilon$. We conjecture that MART [WZY$^{+}$20] is more sensitive to changes in $\epsilon$ between training and test, thus showing similar accuracy for values close $\epsilon = 8/255$, but worse performance compared to smooth CNI otherwise.

We tested our defense against black-box attacks: the transferable attack [LCLS16] and NAttack [LLW$^{+}$19]. For the transferable attack, we trained another instance of the CNI base model and used it as a source model in two configurations: PGD with and without smoothing. We present the best-performing instances of smooth CNI defense (Tables 8.4 and 8.5). For the transferable attack, we conclude that the performance of the model is independent on the smoothing of the source model. Our method showed
Table 8.3: Comparison of our method to prior art on CIFAR-10 with Wide-ResNet28-4 (top part), Wide-ResNet28-10 (middle) and Wide-ResNet34-10 (bottom) under PGD attack. smooth CNI uses $M = 64$ with $\sigma = 0.22$ during inference. † denotes our results based on code provided by the authors or our re-implementation. + denotes our results based on the checkpoint provided by the authors.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy, %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Clean</td>
</tr>
<tr>
<td>Adversarial training [MMS†18]</td>
<td>86.08</td>
</tr>
<tr>
<td>PNI [HRF19]†</td>
<td>84.54</td>
</tr>
<tr>
<td>CNI [ZBN+20]†</td>
<td>84.44</td>
</tr>
<tr>
<td>Smooth CNI (ours)</td>
<td>85.81</td>
</tr>
<tr>
<td>Feature Matching [LFZ+20]</td>
<td>85.21</td>
</tr>
<tr>
<td>Adversarial training [MMS†18]</td>
<td>87.56</td>
</tr>
<tr>
<td>PNI [HRF19]†</td>
<td>85.89</td>
</tr>
<tr>
<td>CNI [ZBN+20]†</td>
<td>86.44</td>
</tr>
<tr>
<td>IAAT [BGH19]</td>
<td>91.3</td>
</tr>
<tr>
<td>CSAT [SGI19]</td>
<td>87.65</td>
</tr>
<tr>
<td>TRADES [ZYJ+19]+</td>
<td>84.92</td>
</tr>
<tr>
<td>RNN-TRADES [XH20]</td>
<td>84.21</td>
</tr>
<tr>
<td>DAT [WMB+19]</td>
<td>85.03</td>
</tr>
<tr>
<td>MART [WZY+20]+</td>
<td>83.62</td>
</tr>
</tbody>
</table>

Figure 8.1: Accuracy on CIFAR-10 under PGD attack as a function of $\epsilon$. The smooth CNI model is WideResNet-28-4, and the model for MART is WideResNet-34-10. The smoothing uses $M = 64$ and $\sigma = 0.25$. Both models were trained using PGD attacks with $k = 10$ and $\epsilon = 8/255$. 

Technion - Computer Science Department - Ph.D. Thesis PHD-2021-05 - 2021
Table 8.4: Results on CIFAR-10 with Wide-ResNet28-4 under black-box transferable PGD attack with \( k = 10 \). PGD-s-trans denotes the setting were the attack is generated on a model with smoothing, in this case the \( \sigma \) for both generation and testing the attack is the same. We presents results for adversarial training (top) and for CNI (bottom).

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy, %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PGD</td>
</tr>
<tr>
<td>( M ) ( \sigma )</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>40.36</td>
</tr>
<tr>
<td>1 0</td>
<td>55.32</td>
</tr>
<tr>
<td>256 0.09</td>
<td>58.33</td>
</tr>
<tr>
<td>512 0.17</td>
<td>59.59</td>
</tr>
<tr>
<td>512 0.22</td>
<td>60.30</td>
</tr>
</tbody>
</table>

Table 8.5: Results on CIFAR-10 with Wide-ResNet28-4 under black-box NAttack [LLW+19]. Smooth CNI uses \( \sigma = 0.21 \) for both models. † denotes our results based on code provided by authors or our re-implementation. + denotes our results based on the checkpoint provided by the authors.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy, %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adversarial training</td>
<td>43.75</td>
</tr>
<tr>
<td>[MMS+18]†</td>
<td></td>
</tr>
<tr>
<td>MART [WZY+20]+</td>
<td>47.02</td>
</tr>
<tr>
<td>CNI [ZBN+20]†</td>
<td>42.61</td>
</tr>
<tr>
<td>Smooth CNI (ours, ( M = 8 ))</td>
<td>56.76</td>
</tr>
<tr>
<td>Smooth CNI (ours, ( M = 64 ))</td>
<td>57.00</td>
</tr>
</tbody>
</table>

a consistent improvement of \( \sim 4\% \) over the CNI base model. The NAttack comparison shows that our method substantially outperforms previous art and the baselines by at least 10%.

8.4.2 Ablation study

We now turn to ablation study of our method. In Section 8.4.2 we show the dependence of smoothing on the sampling hyper-parameters, namely the number of samples, \( M \) and the injected noise standard deviation. In Section 8.4.2 we compare the effectiveness of the PGD, EPGD and SmoothAdvPGD [SYL+19] attacks on our smooth CNI model. In Section 8.4.2 we conduct performance comparison of the smooth models. In particular, we compare our fine-tuned models to the current state-of-the-art with similar fine-tuning.

Sampling

In Fig. 8.2 we compared a different number of Monte Carlo samples: \( M = 2^n \), \( n \in \{0, \ldots, 9\} \). For each value of \( M \), we considered multiple noise standard deviation values in the range of \([0, 0.5]\). The selected upper bound corresponds to significant degradation
of the model performance. We observe that the accuracy increases for a larger $M$ across all values of noise variance. Accuracy of all models on clean data drops as the noise variance increases, achieving their maximal clean accuracy for almost zero noise variance. In addition, all the models achieve their maximal adversarial accuracy for the same value of noise variance, for which most values of $M$ show improvement in clean accuracy when compared to single sample ($M = 1$) without noise. Hence, smoothing improves both clean and adversarial robustness, with the clean-adversarial trade-off controlled by the noise variance.

**Comparison of Attacks**

In Fig. 8.3 we compared the adversarial accuracy of the PGD, EPGD and SmoothAdv$_{\text{PGD}}$ \cite{Syl\textit{+}19} attacks on the CNI base model and our smooth CNI with $M = 8$ samples. Since EPGD and SmoothAdv$_{\text{PGD}}$ attacks with $k$ PGD steps and $M$ samples require computing the gradients $k \cdot M$ times, those attacks require an amount of computation similar to PGD attack with $k \cdot M$ steps. Therefore, we compare the adversarial accuracy as a function of $k \cdot M$ (where for PGD, $M = 1$). We find that EPGD is comparable but not better than PGD, and that SmoothAdv$_{\text{PGD}}$ is less effective, achieving higher accuracy on both models for all values of $k \cdot M$.

PGD attacks are the most effective for lower $k \cdot M$, but converge to the same value as EPGD attacks without smoothing. In contrast, SmoothAdv$_{\text{PGD}}$ converge to higher accuracy for both models. It could indicate that EPGD makes better use of the information encoded in the multiple samples, so much so that it produces an attack comparable to PGD with $1/M$ of the required iterations. As such, SmoothAdv$_{\text{PGD}}$ is less useful than EPGD for empirical evaluation, and we continue to compare EPGD
Adversarial accuracy

Figure 8.3: Accuracy on CIFAR-10 under PGD, EPGD and \textit{SmoothAdv}\textsubscript{PGD} attacks as a function of number of attack steps, \(k \cdot M\). The model is WideResNet-28-4 defended with CNI. The smoothing uses \(M = 1, 8\) and \(\sigma = 0.24\). \textit{EPGD} and \textit{SmoothAdv}\textsubscript{PGD} attacks averaged the gradients over \(M_{\text{backward}} = 8\) samples in each attack iteration. All the attacks run with 5 random restarts. Shaded area is a standard deviation of the results over three runs.

with PGD.

In addition, for all three attacks types, the accuracy on smooth CNI is higher when compared to the base model. However, the improvement is more substantial for EPGD and \textit{SmoothAdv}\textsubscript{PGD}, suggesting that EOT \cite{ACW18} or similar attacks do not produce a significant decrease in the adversarial accuracy of our smooth model. Therefore, while some of our method’s effect may result from random gradient obfuscation, it cannot be easily exploited to undermine our method.

**Fine-tuning**

In addition to the adversarially-trained CNI \cite{ZBN20}, we trained two models obtained by fine-tuning the CNI base model, one of which was trained with the soft smoothing training. For comparison, the second model was similarly fine-tuned with PGD attack. For the PGD attack, we multiply \(k\) by \(M\) to get computational complexity similar to EPGD. We chose to limit the number of samples of the EPGD attack to \(M = 8\), since the further improvement of the resulting model is counterbalanced by the increase in the computational complexity.

We compare the effect of fine-tuning on our models on CIFAR-10 and CIFAR-100. Surprisingly, the results for both types of fine-tuning models were almost identical.
Table 8.6: Comparison of our method to prior art on CIFAR-10 (top) and CIFAR-100 (bottom) with Wide-ResNet28-4 under PGD attack, with and without fine-tuning. Smooth CNI uses $M = 64$ for all models and specify the $\sigma$ used. During the fine-tuning, $k = 10$ and $M = 8$ were used for the EPGD setting, and $k = 80$ for the high $k$ setting. For comparison, we show the results for current state-of-the-art [WZY+20, LFZ+20] with and without fine-tuning on high $k$.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy, %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smooth CNI ($\sigma = 0.22$)</td>
<td>Clean PGD-10 PGD-20</td>
</tr>
<tr>
<td>Smooth CNI ($k = 80$) ($\sigma = 0.28$)</td>
<td>87.24 64.30 55.64</td>
</tr>
<tr>
<td>Smooth CNI-EPGD ($\sigma = 0.3$)</td>
<td>87.53 64.31 55.19</td>
</tr>
<tr>
<td>MART [WZY+20]$^+$</td>
<td>83.62 57.3 57.1</td>
</tr>
<tr>
<td>MART ($k = 80$)$^+$</td>
<td>83.94 59.4 57.4</td>
</tr>
<tr>
<td>Smooth CNI ($\sigma = 0.04$)</td>
<td>61.72 28.57 22.53</td>
</tr>
<tr>
<td>Smooth CNI ($k = 80$) ($\sigma = 0.21$)</td>
<td>55.14 33.34 28.42</td>
</tr>
<tr>
<td>Smooth CNI-EPGD ($\sigma = 0.22$)</td>
<td>54.13 33.45 28.60</td>
</tr>
<tr>
<td>Feature Matching [LFZ+20]</td>
<td>59.51 31.68 31.11</td>
</tr>
</tbody>
</table>

Figure 8.4: Accuracy on CIFAR-10 as a function of injected noise strength on clean data (left) and under PGD attack (right). The model is ResNet-20 defended with CNI and fine-tuned on PGD with $k = 56$. The smoothing uses $M = 2^n$, $n \in \{0, \ldots, 9\}$. Shaded area is a standard deviation of the results over three runs.

As summarized in Table 8.6, in terms of adversarial accuracy, the smooth fine-tuned models improved by $\sim 4\%$ on CIFAR-10 and $\sim 5-6\%$ on CIFAR-100. In contrast, for MART fine-tuning has shown only a $\sim 2\%$ improvement on CIFAR-10. To conclude, our fine-tuned models significantly outperform the fine-tuned MART [WZY+20] and Feature Matching [LFZ+20] models, achieving state-of-the-art results for $k = 10$ on both CIFAR-10 and CIFAR-100.

In Fig. 8.4, we evaluated the impact of noise injection variance for different numbers of samples on a ResNet-20 model defended with CNI and fine-tuned on PGD with $k = 56$. We conclude that both clean and adversarial accuracy reaches are maximized.
Figure 8.5: Accuracy on CIFAR-10 as a function of injected noise strength on clean data (left) and under PGD attack (right). The model is ResNet-20 defended with CNI, with different smoothing and fine-tuning methods. The smoothing uses $M = 8$ for all the methods. All the attacks run with 5 random restarts. The shaded area is a standard deviation of the results over three runs.

Figure 8.6: Accuracy on CIFAR-10 under PGD or EPGD attacks as a function of number of attack steps, $k \cdot M$. The model for MART is WideResNet-34-10 and the rest are WideResNet-28-4, all models are fine-tuned on PGD with $k = 80$. EPGD attacks are only used when specified, and the gradients are averaged over $M_{backward} = 8$ samples in each attack iteration. The smoothing uses $M = 8$ and $\sigma = 0.25$. All the attacks run with 5 random restarts. Shaded area is a standard deviation of the results over three runs.

for zero or almost zero variance, in contrast to Fig. 8.2 where the adversarial accuracy is maximized a non-zero variance.

Fig. 8.5 demonstrates that there is a single optimal value of noise strength for each base model, independent of the smoothing method. In all the figures, we see a single maximum, which is fixed for each of the base models and unrelated to the smoothing method. It indicates that for each such model, there should be an optimal variance of the injected noise.

In Figs. 8.6 and 8.7, we show the decay of the smoothing effect for high-$k$ attacks.
both with a CNI and adversarial training as the base model. Due to this decay, the significant performance boost for lower values of \( k \) vanishes for larger values of \( k \). Moreover, the same effect appears for the CNI base model, even without the application of smoothing. For CIFAR-100, however, the gap between CNI and smooth CNI is not closed entirely, even with the extreme value of \( k = 4000 \). In addition, while for CIFAR-10 EPGD shows the same performance as PGD given enough iterations, for CIFAR-100 EPGD manages to break the smooth defense in instances where PGD is incapable of doing so.

We define a minimal \( k \) to break the defense, \( k_c \), as a value for which adversarial accuracy is at most 0.5% higher than the minimum value (as a function of \( k \)). For adversarial training on CIFAR-10, this value is \( k_c = 40 \), while with smoothing, it increases to \( k_c = 100 \). CNI pushes the minimum further (\( k_c = 500 \) without smoothing and \( k_c = 900 \) with smoothing). MART, on the other hand, is barely affected by increased number of iterations, with \( k_c = 25 \).
8.5 Discussion

This paper proposed a novel inference-only method for improving empirical adversarial robustness by taking into account the spatial information in the area of the data points. We showed that the proposed method could manage changes in $\epsilon$ better than previous state-of-the-art approaches, such as MART. However, as our theoretical results suggest, the smoothing performance is highly dependent on $k$, the number of the attack iterations. Applying smoothing over a CNI base model achieved state-of-the-art results for FGSM and PGD with $k \leq 10$ on CIFAR-10 with a decline of the robustness for large values of $k$. In addition, smoothing shows high efficiency against black-box attacks, achieving a significant increase over the current state-of-the-art for NAttack on CIFAR-10, as well as improvement over the CNI base model for transferable attacks. Moreover, for both white-box and black-box attacks, these results require only a small number of samples, $M$.

We conclude that smoothing offers a practical trade-off between the inference time and model performance for these configurations. In addition, while smoothing can be incorporated into any adversarial defense, we have seen that the CNI base model [ZBN+20], is essential for it to be effective, which aligns with our theoretical results in Eq. (8.27).

Nevertheless, as there is a noticeable decline in the performance for attacks with higher $k$ values, we believe that some of our method’s effect may indeed result from random gradient obfuscation. The usual ways of overcoming gradient obfuscation, such as estimating the expectation of the gradient [ACW18], however, did not produce a significant decrease in adversarial accuracy without substantial increase in computational complexity. Moreover, we note that the same effect appears in the base models we used [HRF19, ZBN+20] and, to the best of our knowledge, no efficient attack to exploit this kind of obfuscation has been proposed yet.
Chapter 9

Single-Node Attack for Fooling Graph Neural Networks

9.1 Motivation

Graph neural networks (GNNs) have shown broad applicability in a variety of domains. Some of these domains, such as social networks and product recommendations, are fertile ground for malicious users and behavior. We show that GNNs are vulnerable to the extremely limited scenario of a single-node adversarial attack: an attacker can force the GNN to classify any target node to a chosen label by only slightly perturbing another single arbitrary node in the graph, without being able to pick that specific attacker node. When the adversary is allowed to pick the attacker node, the attack is even more effective, reducing test accuracy by more than 70%. We also show analytically, that if the norm of the perturbation is greater than the fraction of non-zero given features, our attack is more powerful than the common attack that inserts or removes edges. Finally, we show empirically that our attack is effective across various GNN types (e.g., GCN, GraphSAGE, GAT, GIN), across different real-world datasets, and as a targeted and a non-targeted attack.
Our vision...

(a) Before attacking: the victim node \((v)\) is classified as valid.

(b) After attacking: the victim node \((v)\) is classified as invalid.

Figure 9.1: An adversarial example from the test set of the Twitter dataset. An adversarially-crafted post perturbs the representation of the attacker node. This perturbation causes a misclassification of the target victim node, although they are not even direct neighbors.

9.2 Introduction

Graph neural networks (GNNs) \cite{SGT+09, Mic09} have recently shown sharply increasing popularity due to their generality and computation efficiency \cite{DMI+15, LTBB16, KW17, HYL17, VCC+18, XHLJ19}. Graph-structured data underlie a plethora of domains such as citation networks \cite{SNB+08}, social networks \cite{LM12, RCS+17, RCS+18}, knowledge graphs \cite{TDWS17, WLLZ18, SKB+18}, and product recommendations \cite{SMBG18}. Therefore, GNNs are useful in a variety of real-world structured data.

Most work in this field has focused on designing new GNN variants and applying them to a growing number of domains. However, merely few past works have explored the vulnerability of GNNs to realistic adversarial examples. Consider the following scenario: a malicious user or a bot joins a social network such as Twitter or Facebook. The malicious user mocks the behavior of a benign user, establishes connections with other users, and submits benign posts. After some time, the user submits a new adversarially crafted post, which might seem irregular but overall benign. If the social network uses a GNN-based model to detect malicious users, the new adversarial post changes the representation of the user as seen by the GNN. As a result, another specific benign user gets blocked from the network; alternatively, another malicious user submits an inciteful or racist post — but does not get blocked. This scenario is illustrated in Figure 9.1. In this thesis, we show the feasibility of such a troublesome scenario: a single attacker node can perturb its own representation, such that another node will be misclassified as a label of the attacker’s choice.

Most prior work that explored adversarial attacks on GNNs required the perturbation to span multiple nodes, which in reality required the cooperation of multiple attackers. For example, the pioneering work of \cite{ZAG18} perturbs a set of attacker nodes; \cite{BG19a} perturb edges, and perturbing these edges in reality requires control over the set of nodes that cover these edges. In contrast, in this thesis we show the surprising effectiveness of a single-node attack.
Additionally, most previous approaches perturb the adjacency of the graph, by adding and removing edges [DLT+18, LXC+20]. [ZAG18] and [WWT+19] perturb both edges and nodes, and concluded that perturbing edges is more effective than perturbing nodes. Nonetheless, we show empirically and analytically that perturbing nodes is more harmful than perturbing edges, when evaluated in realistic and comparable settings.

In this chapter, we present the first single-node adversarial attack on GNNs. When the attacker node is chosen randomly, and the attacker is limited to perturb only a small percentage of its vector attributes (10%) or limited to small perturbation norm (0.1), our attack reduces test accuracy by (absolute) 27% on average, across multiple datasets. If the attacker can choose the attacker node, for example, by hacking into an existing account, the attack’s efficiency significantly increases: this reduces test accuracy by 71%. We present two approaches for selecting the attacker: a white-box gradient-based approach, and a black-box, model-free approach that relies on graph topology. Further, we show analytically that in expectation, perturbing a single-node using our attack is more effective than perturbing a single edge, in contrast with the conclusion of previous work. Finally, we perform an extensive experimental evaluation of our approach on multiple datasets and GNN architectures, including thorough empirical analysis.
9.3 Single-Node GNN Attack

In this section, we describe our Single-node INdirect Gradient adversariaL Evasion (SINGLE) attack. Our attack is the first attack that focuses on perturbing nodes (in contrast to edges [DLT+18]), which works with an arbitrary single attacker node (in contrast to multiple nodes [ZAG18]) that is not the node under attack (in contrast to “direct” attacks where the attacker perturbs the node under attack directly [ZAG18, LXC+20]).

9.3.1 Problem Definition

Given a graph \( G \), a trained model \( f_\theta \), a “victim” node \( v \) from the test set along with its classification by the model \( \hat{y}_v = f_\theta (G, v) \), we assume that an adversary controls another node \( a \) in the graph. The goal of the adversary is to modify its own feature vector \( x_a \) by adding a perturbation vector \( \eta \in \mathbb{R}^D \) of its choice, such that the model’s classification of \( v \) will change.

We denote by \( G_{x_a+\eta} \) the graph \( G \) where the row of \( X \) that corresponds to the node \( a \) was added with the vector \( \eta \). In a non-targeted attack, the goal of the attacker is to find a perturbation vector \( \eta \) that will change the classification to some other class, i.e., \( f_\theta (G_{x_a+\eta}, v) \neq f_\theta (G, v) \). In a targeted attack, the adversary chooses a specific label \( y_{adv} \in \mathcal{Y} \) and the adversary’s goal is to force \( f_\theta (G_{x_a+\eta}, v) = y_{adv} \).

Generally, the classification of a node \( v \) depends only on nodes whose distance to \( v \) in the graph is lower than or equal \( L \) – the number of GNN layers. Thus, a modification of the features of \( a \) will affect the classification of \( v \) only if the distance between \( a \) and \( v \) is lower than or equal \( L \). Otherwise, \( a \) will not be contained in the receptive field of \( v \), and the attack will result in “under-reaching” [AY20] – no perturbation of \( a \) will affect the prediction of \( v \) [BKM+20]. Therefore, we require that \( \text{distance}_G (a, v) \leq L \).

In this work, we focus on gradient-based attacks. These attacks assume that the attacker can access a similar model to the model under attack and compute gradients. As recently shown by [WSS20], this is reasonable assumption: an attacker can query the original model; using these queries, imitate the model under attack by training an imitation model; find adversarial examples using the imitation model; and transfer these adversarial examples back to the original model. Under this assumption, these attacks are general and are applicable to any GNN and dataset.

9.3.2 Unnoticeable Perturbations

Our main challenge is to find an adversarial example that will allow an imperceptible perturbation of the input. This objective is attainable in continuous domains such as images [SZS+13, GSS14] and audio [CW18] if we constrain \( l_\infty \)-norm of the perturbation vector \( \eta \). It is, however, unclear what imperceptibility means in graphs. In most GNN datasets, a node’s features are a bag-of-words representation of the words that are
associated with the node. For example, in Cora [MNRS00, SNB+08], every node is annotated by a many-hot feature vector of words that appear in the paper. We denote such datasets as discrete datasets, because the initial (given) feature vector of every node contains only discrete values. In contrast, in PubMed [NLGH12], node vectors are TF-IDF word frequencies; in Twitter [RCS+17], node features are averages of GloVe embeddings, which can be viewed as word frequency vectors multiplied by a (frozen) embedding matrix. We denote such datasets as continuous datasets, because the initial feature vector of every node contains continuous values (floating-point numbers).

An attack would be unnoticeable in an academic paper or in a set of Tweets if the frequency of some words is slightly modified. For example, a few word may be repeated a few times throughout the text. In continuous datasets, we thus constrain the perturbation vector \( \eta \) by requiring \( \| \eta \|_\infty \leq \epsilon_\infty \) – the absolute value of the elements in the perturbation vector is bounded by \( \epsilon_\infty \in \mathbb{R}^+ \). In discrete datasets, small, element-wise perturbations are invalid, because the perturbed vector \( x_a + \eta \) must be discrete as well – if every node is given as a many-hot vector \( x_a \), the perturbed vector \( x_a + \eta \) must remain many-hot as well. In discrete datasets, we thus constrain the perturbation vector by requiring \( \| \eta \|_0 / D \leq \epsilon_0 \) – the fraction of non-zero elements in the perturbation vector is bounded by \( \epsilon_0 \in \mathbb{R}^+ \). In this case, measuring the \( \ell_0 \) norm of \( \eta \) is equivalent to the \( \ell_1 \) norm: \( \| \eta \|_0 = \| \eta \|_1 \), and is proportional to the \( \ell_2 \) norm.

### 9.3.3 Finding the Perturbation Vector

To find the perturbation vector, our general approach is to iteratively differentiate the desired loss of \( v \) with respect to the perturbation vector \( \eta \), and update \( \eta \) according to the gradient, similarly to the general approach in images [GSS14]. In non-targeted attacks, we take the positive gradient of the loss of the undesired label to increase the loss; in targeted attacks, we take the negative gradient of the loss of the adversarial label \( y_{adv} \):

\[
\eta^{t+1} = \begin{cases} 
\eta^t + \gamma \nabla_\eta J \left( f_\theta \left( G_{x_a + \eta^t} ; v \right) , \hat{y}_v \right) & \text{non-targeted attack} \\
\eta^t - \gamma \nabla_\eta J \left( f_\theta \left( G_{x_a + \eta^t} ; v \right) , y_{adv} \right) & \text{targeted attack}
\end{cases}
\]

where \( \gamma \in \mathbb{R}^+ \) is a learning rate. We repeat this process for a predefined number of \( K \) iterations, or until the model predicts the desired label.

In continuous datasets, after each update, we clip the perturbation vector \( \eta^{t+1} \) according to the \( \epsilon_\infty \) constraint: \( \| \eta^{t+1} \|_\infty \leq \epsilon_\infty \).

In discrete datasets, where node features are many-hot vectors, the only possible perturbation to every vector attribute is “flipping” it from 0 to 1 or vice versa. In every update iteration, we thus “flip” the vector attribute with the largest gradient. We repeat this process as long as the \( \epsilon_0 \) ratio holds: \( \| \eta^{t+1} \|_0 / D \leq \epsilon_0 \), or until the model predict the desired label.

127
Differentiate by frequencies, not by embeddings. When taking the gradient $\nabla \eta$, there is a subtle, but crucial, difference between the way that node representations are provided in the dataset:

- indicative datasets provide initial node representations $X = [x_1, x_2, ...]$ that are word indicator vectors (many-hot) or frequencies such as (weighted) bag-of-words [SNB+08, SMBG18];

- contrarily, in encoded datasets, initial node representations are given encoded, e.g., as an average of word2vec vectors [HYL17, HFZ+20].

Indicative datasets can be converted to encoded by multiplying every vector by an embedding matrix; encoded datasets cannot be converted to indicative, without the authors releasing the textual data that was used to create the encoded dataset.

In indicative datasets, a perturbation of a node vector can be realized as a perturbation of the original text from which the indicative vector was derived. That is, adding or removing words in the text can result in the perturbed node vector. In contrast, a few-indices perturbation in encoded datasets might be an effective attack, but is not be realistic because there is no perturbation of the original text that results in that perturbation of the vector. That is, to obtain realistic adversarial examples, it is crucial to use indicative datasets, or convert encoded datasets to indicative representation (as we do in Section 9.4) using their original text.

9.3.4 Perturbing Nodes Rather than Edges

Previous work mostly focused on perturbing graph edges. [ZAG18] perturb both edges and node features, but conclude that “perturbations in the structure lead to a stronger change in the surrogate loss compared to feature attacks”; [WWT+19] also conclude that “perturbing edges is more effective than modifying the features”. In this thesis, we counter these conclusions and show that in the realistic scenario where the attacker controls only a single node, small node feature perturbations are stronger than edge perturbations:

- first, removing all the edges of a particular node is a special case of node feature perturbation. There exists a perturbation $\eta$ such that $W^1 (x_a + \eta) = 0$, i.e., the modified feature vector $x_a + \eta$ is in the null space of the first GNN layer.\(^1\) Such a feature perturbation is equivalent to removing all the edges of the node $a$.

- Second, we argue that perturbing the graph structure is not realistic, because a single attacker controls only its own edges, and cannot control the global graph structure as in previous work [DLT+18, BG19b, ZZ20].

\(^1\)This equation demonstrates GCN, but similar equations hold for other GNN types like GAT and GIN.
• Finally, when a successful attack is caused by removing edges, it is unclear whether the misclassification is caused by sensitivity to non-robust features in the data [IST+19], or simply due to smaller amount of information. Similarly, when a successful attack is caused by inserting edges, it is unclear whether this is simply due to incorrect or unrealistic added information.
9.4 Evaluation

We evaluate the effectiveness of our SINGLE attack. In Section 9.4.1, we show that SINGLE is more effective than alternatives such as single-edge attacks. In Section 9.4.2, we show that if we are allowed to choose the attacker node, SINGLE is significantly more effective.

Setup  We trained each GNN type with two layers ($L = 2$), using the Adam optimizer, early stopped according to the validation set, and applied a dropout of 0.5 between layers. We used up to $K = 20$ attack iterations. All experiments in this section were performed with GCN, except for Section 9.4.5, where additional GNN types (GraphSAGE [HYL17], GAT [VCC+18], and GIN [XHLJ19]) are shown. In the supplementary material, we show consistent results across all GNN types mentioned above as well as SGC [WSZ+19], and RobustGCN [ZG19].

Data  We used Cora and CiteSeer [SNB+08] which are discrete datasets, i.e., the given node vectors are many-hot vectors. In discrete datasets, we limit the fraction of perturbed attributes to 0.1 of the attributes, i.e., $\epsilon_0 = 0.1$. In practice, the attack usually uses fewer node attributes. We also used PubMed [SNB+08] and the Twitter-Hateful-Users [RCS+17] datasets, which are continuous, and node features represent frequencies of words. In continuous datasets, which allow finer control over the intensity of the perturbation, we set $\epsilon_\infty = 0.1$. An analysis of values of $\epsilon_0$ and $\epsilon_\infty$ is presented in Sections 9.4.5 and 9.4.6.

The Twitter-Hateful-Users dataset is originally provided as an encoded dataset, where every node is an average of GloVe vectors [PSM14]. We reconstructed this dataset using the original text from [RCS+17], to be able to compute gradients with respect to the weighted histogram of words rather than the embeddings. We took the most frequent 10,000 words as node features and used GloVe-Twitter embeddings to multiply by the node features. We thus converted this dataset to indicative rather than encoded. Statistics of all datasets are provided in the supplementary material.

Compared approaches. In SINGLE, the attacker node is selected randomly for each victim node, and the attack perturbs this node’s features according to $\epsilon_\infty$ and $\epsilon_0$. SINGLE-hops is a modification of SINGLE where the attacker node is sampled only among nodes that are not direct neighbors, i.e., the attacker and the victim are not directly connected ($(a, v) \not\in \mathcal{E}$). The idea in SINGLE-hops is to evaluate a variant of SINGLE that is even more unnoticeable in reality. We compare to additional approaches from the literature: EdgeGrad follows most previous work [XCL+19, LXC+20, ZG20]: EdgeGrad randomly samples an attacker node as in SINGLE, and either inserts or removes a single edge of the attacker node, according to the gradient.\footnote{This can be implemented easily using edge weights: training the GNN with weights of 1 for existing edges, adding all possible edges with weights of 0, and taking the gradient with respect to the vector of weights.} If both use a randomly
<table>
<thead>
<tr>
<th>Attack Method</th>
<th>Cora</th>
<th>CiteSeer</th>
<th>PubMed</th>
<th>Twitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clean (no attack)</td>
<td>80.5 ± 0.8</td>
<td>68.5 ± 0.7</td>
<td>78.5 ± 0.6</td>
<td>89.1 ± 0.2</td>
</tr>
<tr>
<td>EdgeGrad</td>
<td>70.5 ± 0.6</td>
<td>48.2 ± 0.9</td>
<td>59.7 ± 0.7</td>
<td>82.7 ± 0.0</td>
</tr>
<tr>
<td>SINGLE</td>
<td>54.1 ± 0.6</td>
<td>25.7 ± 1.4</td>
<td>45.5 ± 0.5</td>
<td>72.1 ± 7.2</td>
</tr>
<tr>
<td>SINGLE-hops</td>
<td>64.2 ± 0.7</td>
<td>39.7 ± 1.4</td>
<td>48.7 ± 0.9</td>
<td>74.5 ± 6.7</td>
</tr>
</tbody>
</table>

Table 9.1: Test accuracy (lower is better) under different types of attacks, when the attacker node is chosen randomly. Performed using GCN, $\epsilon_\infty = 1$ for the discrete datasets (Cora and CiteSeer), and $\epsilon_\infty = 0.1$ for the continuous datasets (PubMed and Twitter).

selected attacker node, EdgeGrad is strictly stronger than the GradArgmax attack of [DLT+18], which only removes edges. We ran each approach 5 times with different random seeds for each dataset, and report the mean and standard deviation.

9.4.1 Main Results

Table 9.1 shows our main results for non-targeted attacks: SINGLE is more effective than EdgeGrad across all datasets. SINGLE-hops, which is more unnoticeable than attacking with a neighbor node, performs almost as good as SINGLE, which attacks using either a neighboring or a non-neighboring node, and better than EdgeGrad. SINGLE-hops reduces test accuracy by an average of 22\% (absolute), and SINGLE reduces test accuracy by an average of 30\% (absolute). For example, in Twitter, SINGLE reduces the test accuracy significantly better than EdgeGrad: 72.1\% compared to 82.7\%.

Results for targeted attacks are shown in the supplementary material.

9.4.2 Attacker Choice

If the attacker could choose its node, e.g., by hijacking a specifically chosen existing account in a social network, could they increase the effectiveness of the attack? We examine the effectiveness of two approaches for choosing the attacker node.

Gradient Attacker Choice (GradChoice) chooses the attacker node according to the largest gradient with respect to the node representations (for a non-targeted attack): $a^* = \arg\max_{a \in V} \|\nabla_x J(\theta(G, v), \hat{y}_v)\|_\infty$. The chosen attacker node is never the victim node itself.

Topological Attacker Choice (Topology) chooses the attacker node according to topological properties of the graph. As an example, we choose the neighbor of the victim node $v$ with the smallest number of neighbors: $a^* = \arg\min_{a \in N_v} |N_a|$. The advantage of this approach is that the attacker choice is model-free: if the attacker cannot compute gradients, they can at least choose the most harmful attacker node, and then perform the perturbation itself using other non-gradient approaches such as [WMWR18] and [CRX+20].
<table>
<thead>
<tr>
<th>Method</th>
<th>Cora</th>
<th>CiteSeer</th>
<th>PubMed</th>
<th>Twitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>GlobalEdgeGrad</td>
<td>29.7 ± 2.4</td>
<td>11.9 ± 0.8</td>
<td>15.3 ± 0.4</td>
<td>82.7 ± 0.0</td>
</tr>
<tr>
<td>SINGLE+GradChoice</td>
<td>31.0 ± 1.9</td>
<td>31.0 ± 1.7</td>
<td>8.5 ± 1.2</td>
<td>7.0 ± 1.1</td>
</tr>
<tr>
<td>SINGLE+Topology</td>
<td>11.9 ± 0.7</td>
<td>7.5 ± 0.7</td>
<td>5.2 ± 0.1</td>
<td>6.6 ± 0.5</td>
</tr>
</tbody>
</table>

Table 9.2: Test accuracy when the adversary can *choose* the attacker node.

To perform a fair comparison, we compare these approaches with GlobalEdgeGrad, which is similar to EdgeGrad that can insert or remove an edge, with the difference that the chosen edge can be chosen from the entire graph.

Results. Results for these attacker choice approaches are shown in Table 9.2. The main results are that choosing the attacker node significantly increases the effectiveness of our SINGLE attack: for example, in Twitter, from 72.1% (Table 9.1) to 6.6% test accuracy (Table 9.2).

In all datasets, our node-based attack SINGLE+Topology outperforms the edge-based baseline GlobalEdgeGrad across all datasets, showing the superiority of node perturbation over edge perturbation in the global view. SINGLE+GradChoice significantly outperforms GlobalEdgeGrad in the continuous datasets (PubMed and Twitter), but performs worse in the discrete datasets (Cora and CiteSeer). We believe that the reason is the difficulty in the two-step optimization of discrete datasets: SINGLE+GradChoice needs to choose the node, and find the perturbation afterwards. Finding a perturbation for a discrete vector is more difficult than in continuous datasets, and the choice of the attacker node may not be optimal.

In the supplementary material, we show that allowing GlobalEdgeGrad to insert and remove multiple edges of the same attacker node – does not lead to a significant improvement.

Fraction of perturbed attributes. It is also interesting to note that SINGLE+Topology and SINGLE+GradChoice tend to perturb much fewer vector attributes than SINGLE. In Cora, SINGLE uses 7.9% of the attributes, while SINGLE+GradChoice uses 3.9%. In CiteSeer, SINGLE uses 5.3% of the attributes, while SINGLE+Topology uses 3.9%. In Twitter, SINGLE uses 8.9% of the attributes, while SINGLE+Topology and SINGLE+GradChoice use only 0.6%.

### 9.4.3 Scenario Ablation

The main scenario that we focus on in this thesis is a SINGLE approach that always perturbs a single node, which is not the victim node (a ≠ v). We now examine our SINGLE attack in other, easier but less realistic, scenarios: SINGLE-two attackers follows [ZAG18] and [ZXY20], randomly samples two attacker nodes and perturbs their features using the same approach as SINGLE. SINGLE-direct perturbs the victim
Table 9.3: Scenario ablation: test accuracy under different attacking scenarios.

<table>
<thead>
<tr>
<th></th>
<th>Cora</th>
<th>CiteSeer</th>
<th>PubMed</th>
<th>Twitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>SINGLE-two attackers</td>
<td>39.0 ± 1.1</td>
<td>22.5 ± 0.3</td>
<td>27.7 ± 0.2</td>
<td>44.3 ± 12.8</td>
</tr>
<tr>
<td>SINGLE-direct</td>
<td>0.9 ± 0.2</td>
<td>1.3 ± 0.1</td>
<td>0.3 ± 0.1</td>
<td>57.6 ± 8.7</td>
</tr>
<tr>
<td>SINGLE</td>
<td>54.1 ± 0.6</td>
<td>25.7 ± 1.4</td>
<td>45.5 ± 0.5</td>
<td>72.1 ± 7.2</td>
</tr>
</tbody>
</table>

Table 9.4: Test accuracy while attacking a model that was adversarially trained on PubMed.

<table>
<thead>
<tr>
<th></th>
<th>Standard training</th>
<th>Adversarial training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clean (no attack)</td>
<td>78.5 ± 0.6</td>
<td>76.9 ± 0.6</td>
</tr>
<tr>
<td>SINGLE</td>
<td>45.5 ± 0.5</td>
<td>58.5 ± 2.7</td>
</tr>
<tr>
<td>SINGLE-hops</td>
<td>48.7 ± 0.9</td>
<td>62.1 ± 2.5</td>
</tr>
<tr>
<td>SINGLE+GradChoice</td>
<td>8.5 ± 1.2</td>
<td>30.6 ± 6.8</td>
</tr>
<tr>
<td>SINGLE+Topology</td>
<td>5.2 ± 0.1</td>
<td>21.1 ± 2.1</td>
</tr>
<tr>
<td>SINGLE-two attackers</td>
<td>27.7 ± 0.2</td>
<td>40.7 ± 3.4</td>
</tr>
<tr>
<td>SINGLE-direct</td>
<td>0.3 ± 0.1</td>
<td>4.6 ± 1.1</td>
</tr>
</tbody>
</table>

Figure 9.2: Effectiveness of our SINGLE attack compared to the allowed $\epsilon_\infty$ (on PubMed).

Figure 9.3: Test accuracy under our SINGLE attack, compared to $\epsilon_0$, the maximal ratio of perturbed features.

As shown in Table 9.3, expectedly, perturbing two attacker nodes or perturbing the victim node itself is more effective, albeit being less realistic.

9.4.4 Effectiveness of the Attack Facing Defense

In the previous sections, we studied the effectiveness of our SINGLE attack. In this section, we investigate to what extent can defensive training approaches defend against node directly (i.e., $a = v$), an approach that was found to be the most efficient by [ZAG18]. Table 9.3 shows the test accuracy of these ablations. In the supplementary material, we additionally experiment with more than two attacker nodes.
SINGLE.

Adversarial Training. We experimented with attacking models that were adversarially trained [MMS+18]. For each training step and labeled training node, we perform $K_{\text{train}}$ adversarial steps to adversarially perturb another randomly sampled node, exactly as in SINGLE, but at training time. The model is then trained to minimize the original cross-entropy loss and the adversarial loss:

$$L(f_{\theta}, D) = \frac{1}{2N_D} \sum_{i=0}^{N_D} \left[ J(f_{\theta}(G, v_i), y_i) + J(f_{\theta}(G_{x_i} + \eta_i, v_i), y_i) \right]$$

In every training step, we randomly sample a new attacker for each victim node and compute new $\eta_i$ vectors. After the model is trained, we attack the model with $K_{\text{test}}$ SINGLE adversarial steps. This is similar to the approach of [FHTC19] and [DDZ19], except that they used adversarial training as a regularizer, to improve the accuracy of a model while not under attack. In contrast, we use adversarial training to defend a model against an attack at test time. We used $K_{\text{train}} = 5$, as we found it to be the maximal value for which the model’s accuracy is not significantly hurt while not under attack (“clean”), and $K_{\text{test}} = 20$ as before.

As shown in Table 9.4, adversarial training indeed improves the model’s robustness against the different SINGLE attacks. However, the main result of this section is that SINGLE, SINGLE+GradChoice and SINGLE+Topology are still very effective attacks, as they succeed in attacking the adversarially trained model, reducing its test accuracy to 58.5%, 30.6% and 21.1%, respectively.

Robust Training. We also experimented with attacking a RobustGCN model [ZG19], where the architecture and the training scheme are optimized for robustness. We used the publicly available code of [ZG19]. Surprisingly, we found that RobustGCN is as vulnerable to our SINGLE attack as a standard GCN, showing the effectiveness of our attack and showing that there is still much room for novel ideas and improvements to the robustness of current GNNs. Results are shown in the supplementary material.
9.4.5 Sensitivity to $\epsilon_\infty$

How does the intensity of the adversarial perturbation affect the performance of the attack? Intuitively, we say that the less we restrict the perturbation (i.e., larger values of $\epsilon_\infty$), the more powerful the attack. We examine whether this holds in practice.

In our experiments in Sections 9.4.1 to 9.4.4, we used $\epsilon_\infty = 0.1$ for the continuous datasets (PubMed and Twitter). In this section, we vary the value of $\epsilon_\infty$ across different GNN types and observe the effectiveness of the attack. Figure 9.2 shows the results on PubMed, and shows that smaller values of $\epsilon_\infty$ than 0.1 are effective as well. As we increase the value of $\epsilon_\infty$, GCN and GraphSage show a natural descent in test accuracy. Contrarily, GAT [VCC+18] and GIN [XHLJ19] are more robust to increased absolute values of perturbations, while GAT is also the most robust compared to the other GNN types.

9.4.6 Sensitivity to $\epsilon_0$

In Section 9.4.5, we analyzed the effect of $\epsilon_\infty$, the maximal allowed perturbation in each vector attribute, on the performance of the attack. In Cora and CiteSeer, the input features are discrete (i.e., the given input node vector is many-hot). In such datasets, the interesting analysis is of the value of $\epsilon_0$, the maximal fraction of allowed perturbed vector elements, on the performance of the attack.

We experimented with limiting the number of allowed perturbed vector elements and measuring the resulting test accuracy. The results are shown in Figure 9.3.

As shown, when $\epsilon_0 = 0$, no attack is allowed, and the test accuracy is equal to the “Clean” value of Table 9.1, and $\epsilon_0 = 0.1$ is the value that we used in the experiments of Sections 9.4.1 to 9.4.3. As we increase $\epsilon_0$, the test accuracy naturally decreases. Although we used $\epsilon_0 = 0.1$ in Sections 9.4.1 to 9.4.3, smaller values of $\epsilon_0$ can lead to even more unnoticeable perturbations, while still being effective. For example, in CiteSeer, even constraining $\epsilon_0 = 0.01$ is effective and decreases the test accuracy by 21% compared to “Clean”.

9.4.7 Distance Between Attacker and Victim

In Section 9.4.1, we found that SINGLE performs similarly to SINGLE-hops, although SINGLE-hops samples an attacker node $a$ whose distance from the victim node $v$ is at least 2. We further question whether the effectiveness of the attack depends on the distance between the attacker and the victim. We trained a new model for each dataset using $L = 8$ layers. Then, for each test victim node, we sampled attackers according to their distance to the test node.

As shown in Figure 9.4, the effectiveness of the attack increases as the distance between the attacker and the victim decreases. At distance of 5, the test accuracy saturates. A possible explanation is that apparently more than few layers (e.g., $L = 2$ in
Figure 9.4: Test accuracy compared to the distance between the attacker and the victim, when trained with $L = 8$ layers.

[17} are not needed in most datasets. Thus, the rest of the layers can theoretically learn not to pass much of their input, starting from the redundant layers, excluding adversarial signals as well.
9.5 Discussion

We demonstrate that GNNs are susceptible to the extremely limited scenario of a Single-node INdirect Gradient adversarial Evasion (SINGLE) attack. We confirm this hypothesis using a thorough evaluation across multiple datasets, GNN types and an extensive ablation study. We show that even adversarial training does not adequately defend against our SINGLE attack. Further, we show that for a single-layer GNN, SINGLE is stronger than an edge-based attack in expectation, if the norm of the perturbation is greater than the fraction of non-zero given features.

The practical consequences of these findings are that a single attacker in a network can force a GNN to classify any other target node as the attacker’s chosen label by slightly perturbing some of the attacker’s features. Furthermore, if the attacker can choose its attacker node – the effectiveness of the attack significantly increases.

We believe that this work will drive research in this field toward exploring novel defense approaches for GNNs. Such defenses can be crucial for real-world systems that are modeled using GNNs. We also believe that this work’s surprising results motivate a better theoretical understanding of the expressiveness and generalization of GNNs.
Chapter 10

Related work

In this section, we provide an overview of prior work of the methods proposed in previous chapters.

Quantization-aware training methods  Previous studies have investigated quantizing the network weights and the activations to as low as 1- or 2-bit representation [RORF16, HCS+17, ZWN+16, DNL+17]. Such extreme reduction of the range of parameter values greatly affects accuracy. Recent works proposed to use a wider network, i.e., one with more filters per layer, to mitigate the accuracy loss [ZHMD16, PPA18]. In some approaches, e.g., [ZHMD16] and [ZYG+17], a learnable linear scaling layer is added after each quantized layer to improve expressiveness.

A general approach to learning a quantized model adopted in recent papers [HCS+16, HCS+17, ZWN+16, RORF16, CHSV17] is to perform the forward pass using the quantized values, while keeping another set of full-precision values for the backward pass updates. In this case, the backward pass is still differentiable, while the forward pass is quantized. In the aforementioned papers, a deterministic or stochastic function is used at training for weight and activation quantization. Another approach introduced by [MM18] and [PPA18] is based on a teacher-student setup for knowledge distillation of a full precision (and usually larger) teacher model to a quantized student model. This method allows training highly accurate quantized models, but requires training of an additional, larger network.

Most previous studies have used uniform quantization (i.e., all quantization bins are equally-sized), which is attractive due to its simplicity. However, unless the values are uniformly distributed, uniform quantization is not optimal.

Unfortunately, neural network weights are not uniform but rather have bell-shaped distributions [HMD16, AB18].

Non-uniform quantization is utilized by [HMD16], where the authors replace the weight values with indexes pointing to a finite codebook of shared values. [UMW17] propose to use clustering of weights as a way of quantization. They fit a Gaussian prior model to the weights and use the cluster centroids as the quantization codebook.
[XWZ′18] also use such an approach but in addition embody incremental quantization both on network and layer levels. Other examples of methods that take data distribution into account are the Bayesian quantization [LUW17] and the value-aware quantization [PYV18]. In [LUW17], sparsity priors are imposed on the network weights, providing information as to the bits to be allocated per each layer. In [MAV17], variational dropout, which also relies on the Bayesian framework, is proposed to prune weights in the network. [PYV18] propose to leave a small part of weights and activations (with higher values) in full-precision, while using uniform quantization for the rest.

Another approach adopted by [ZYG′17] learns the quantizer thresholds by iteratively grouping close values and re-training the weights. [LMC′17] utilize a logarithmic scale quantization for an approximation of the $\ell_2$-optimal Lloyd quantizer [Llo82]. [CHSV17] proposed to optimize expectation of $\ell_2$ error of quantization function to reduce the quantization error. Normally distributed weights and half-normally distributed activations were assumed, that enables using a pre-calculated $k$-means quantizer. In [ZWW′17] balanced bins are used, so each bin has the same number of samples. In some sense, this idea is the closest to our approach presented in Chapter 2; yet, while [ZWW′17] force the values to have an approximately uniform distribution, we pose no such constraint.

**Post-training quantization** One of the first post-training schemes, proposed by [GSZ′18], used min-max ($\ell_\infty$ norm) as a threshold for 8-bit quantization. It produces a small performance degradation and can be deployed efficiently on the hardware. Afterward, better schemes involving the choosing of the clipping value, such as minimizing the Kullback-Leibler divergence [Mig17], assuming the known distribution of weights [BNHS18], or minimizing quantization MSE iteratively [CKYK19], were proposed.

Another efficient approach quantization is to change the distribution of the tensors, making it more suitable for quantization, such as equalizing the weight ranges [NBBW19] or splitting outlier channels [ZGB′19].

Recent works noted that the quantization process introduces a bias into distributions of the parameters and attempt to correct this bias. [MFAG19] addressed a problem of MobileNet quantization. They claimed that the source of degradation was shifting in the mean activation value caused by inherent bias in the quantization process and proposed a scheme for fixing this bias. Multiple alternative schemes for the correction of quantization bias were proposed, and those techniques are widely applied in state-of-the-art quantization approaches [BNHS18, MFAG19, NBBW19, ZYJ′19]. In the work presented in Chapter 3, we utilize the bias correction method by developed [BNHS18].

While the simplest form of quantization applies a single quantizer to the whole tensor (weights or activations), finer quantization allows reduction of performance degradation. Even though this approach usually boosts performance significantly
[BNHS18, FGCC19, CKYK19, LHC+18], it requires more parameters and special hardware support, which makes it unfavorable for real-life deployment.

One can achieve more powerful quantization by using more sophisticated ways to map values to a particular bin as clustering of the tensor entry values [NBBW19], aka non-uniform quantization [ZYJ+19]. By allowing a more general quantizer, these approaches provide better performance than uniform quantization, as a drawback requires complex hardware support. Similarly to fine-grained quantization, they are less suitable for deployment on consumer-grade hardware. In the work presented in chapter 3, we focus on the case of the symmetric tensor-wise quantizer, which is easy to implement in the hardware.

Influence of memory access on energy consumption [YCES17] studied the breakdown of energy consumption in CNN inference. For example, in GoogLeNet [SLJ+15] arithmetic operations consume only 10% of the total energy, while feature map transfers to and from an external RAM amount to about 68% of the energy footprint. However, due to the complexity of real memory systems, not every method that decreases the sheer memory bandwidth will necessarily yield significant improvement in power consumption. In particular, it depends on computational part optimization: while memory performance is mainly defined by the external memory chip, better optimization of computations will lead to higher relative energy consumption of the memory. For example, while [AO18] reported a 70% bandwidth reduction, the dynamic power consumption decreased by a mere 2%.

[XLL+17] proposed fusing convolutional layers to reduce the transfer of feature maps. In an extreme case, all layers are fused into a single group. A similar approach was adopted by [XLS+19], who demonstrated a hardware design that does not write any intermediate results into the off-chip memory. This approach achieved approximately 15% runtime improvement for ResNet and state-of-the-art throughput. However, the authors did not compare the energy footprint of the design with the baseline. [MHS+19] demonstrated that using on-chip cache cuts down the memory bandwidth and thus reduces power consumption by an order of magnitude. In addition, [JYP+17] noted that not only the power consumption but also the speed of DNN accelerators is memory-rather than compute-bound. This is confirmed by [WDCC19], who also demonstrated that increasing computation throughput without increasing memory bandwidths barely affects latency.

Network compression Lossless coding and, in particular, variable length coding (VLC), is a way to reduce the memory footprint without compromising performance. In particular, [HMD16] proposed using Huffman coding to compress network weights, alongside quantization and pruning. [WCMM19] proposed using the more computationally-demanding algorithm DEFLATE (LZ77 + Huffman) to further improve compression rates. [Cha18] used Huffman coding to compress feature maps and
thus reduce memory bandwidth. [GHR18] proposed passing only the lower-dimensional feature maps to the memory to reduce the bandwidth. [CB19] proposed using RLE-based algorithm to compress sparse network activations.

**Smoothing for adversarial robustness** Adversarial attacks were first proposed by [SZS+13], who noted that it was possible to use the gradients of a neural network to discover small perturbations of the input that drastically change its output. Moreover, it is usually possible to change the prediction to a particular class, i.e., perform a targeted attack. It is common to divide adversarial attacks into two classes: white-box attacks, which have access to the internals of the model (in particular, its gradients); and black-box attacks, which have access only to the output of the model for a given input.

Adversarial defenses. To improve the performance of the model under adversarial attack, [SZS+13] generated adversarial examples during training and used them as training samples, optimizing the loss

\[
\mathcal{L}_{\text{adv}}(b_x, y) = (1 - w) \cdot \mathcal{L}_{CE}(f(b_x), y) + w \cdot \mathcal{L}_{CE}(f(u_x), y),
\]

where \(\mathcal{L}_{CE}\) is the cross-entropy loss, \(f\) is the classifier, \(b_x\) is a training instance with the label \(y\), \(u_x\) is the corresponding adversarial example, and \(w\) is a hyperparameter usually set to \(w = 0.5\).

This method is particularly convenient if the generation of adversarial examples is fast [GSS14]. When combined with stronger attacks, it provides a powerful baseline for adversarial defenses [MMS+18], and is often utilized as part of defense procedures.

Many works introduced improvements over regular adversarial training by applying stronger attacks during the adversarial training phase, including using Voronoi cells instead of \(\epsilon\)-balls [KHM19], adding adversarial noise to all the activations [LLZ+19], using a learning-to-learn framework [JCS+18, XH20], or increasing the attack strength based on some heuristic [BGH19, WMB+19].

TRADES [ZYJ+19] uses a batch of randomly perturbed inputs to better cover the neighbourhood of the point. Together with replacing \(\mathcal{L}_{CE}(f(u_x), y)\) in Eq. (10.1) by \(\mathcal{L}_{CE}(f(u_x), f(b_x))\), this provided a significant improvement over standard adversarial training. MART [WZY+20] further improves the defense performance by treating the misclassified examples separately.

Randomization of the neural network can be a very powerful adversarial defense. [ZSLG16] improved robustness by reducing the distance between two samples differing by a normally distributed variable with a small variance. [ZL19] added normal noise to the input, which was shown to reduce the Kullback-Leibler divergence between the clean and the adversarially-perturbed inputs.

Another random-based defense based on adversarial training is parametric noise injection (PNI) by [HRF19]. PNI improves network robustness by injecting Gaus-
sian noise into parameters (weights or activations) with learned variance. [ZBN+20] extended the idea of PNI to colored noise injection (CNI) by injecting low-rank multivariate Gaussian noise instead of white noise:

\[ be \sim \mathcal{N}(b_0, b^* \Sigma) \]  

\[ b^* \Sigma = b^* \Lambda + b^* V b^* V^\top, \]

where \( b^* \Lambda \) is an \( N \times N \) non-negative diagonal matrix, and \( b^* V \) is an \( N \times M \) matrix. Both matrices are learned during the training using the reparametrization trick [KW13]. At inference time, a noise vector sampled from the learned distribution is added to activations, weights, or both. In our work, we focus on weight noise injection, which usually achieves higher adversarial robustness.

[ACW18] observed that many defenses do not improve robustness of the defended network but rather obfuscate gradients, making gradient-based optimization methods less effective. They identified common properties of obfuscated gradients and organized them in a checklist. In addition, they proposed techniques to overcome common instances of obfuscated gradients: in particular, approximating non-differentiable functions with a differentiable substitute and using averaging on the randomized ones.

Randomized smoothing [CRK19], as noted above, is a method for increasing the robustness of a classifier by averaging its outputs over some random distribution in the input space centered at the input sample. [CRK19] showed that randomized smoothing is useful for certification of the classifier [WK18] – that is, proving its performance under norm-bounded input perturbations. In particular, they demonstrated a tight bound on \( L_2 \) certification using randomized smoothing with normal distribution. Consequently, [SYL+19] used the smooth classifier to generate stronger “smooth” adversarial attacks, and utilized them for adversarial training. Such training allows the generation of a more accurate base classifier, and as a result, improves certified robustness properties. Certifying against \( \ell_\infty \) perturbations, however, is a significantly harder task [BDMZ20, YDH+20, ZWLX20, BV20, KLGF20]. In particular, the state-of-the-art certified accuracy is 30.5% for \( \epsilon = 8/255 \) [ZCX+20] and 62.6% for \( \epsilon = 3/255 \) [BDMZ20].

**Graph-based attacks** GNNs [SGT+09, Mic09] have recently shown sharply increasing popularity due to their generality and computation efficiency [DMI+15, LTBZ16, KW17, HYL17, VCC+18, XHLJ19]. Graph-structured data underlie a plethora of domains such as citation networks [SNB+08], social networks [LM12, RCS+17, RCS+18], knowledge graphs [TDWS17, WLLZ18, SKB+18], and product recommendations [SMBG18]. Therefore, GNNs are useful in a variety of real-world structured data.

Most work in this field has focused on designing new GNN variants and applying them to a growing number of domains. However, merely few past works have explored
the vulnerability of GNNs to realistic adversarial examples. Consider the following scenario: a malicious user or a bot joins a social network such as Twitter or Facebook. The malicious user mocks the behavior of a benign user, establishes connections with other users, and submits benign posts. After some time, the user submits a new adversarially crafted post, which might seem irregular but overall benign. If the social network uses a GNN-based model to detect malicious users, the new adversarial post changes the representation of the user as seen by the GNN. As a result, another specific benign user gets blocked from the network; alternatively, another malicious user submits an inciteful or racist post – but does not get blocked. This scenario is illustrated in Figure 9.1. In this work, we show the feasibility of such a troublesome scenario: a single attacker node can perturb its own representation, such that another node will be misclassified as a label of the attacker’s choice.

Most prior work that explored adversarial attacks on GNNs required the perturbation to span multiple nodes, which in reality required the cooperation of multiple attackers. For example, the pioneering work of [ZAG18] perturbs a set of attacker nodes; [BG19a] perturb edges, and perturbing these edges in reality requires control over the set of nodes that cover these edges. In contrast, in this thesis we show the surprising effectiveness of a single-node attack.

Additionally, most previous approaches perturb the adjacency of the graph, by adding and removing edges [DLT+18, LXC+20]. [ZAG18] and [WWT+19] perturb both edges and nodes, and concluded that perturbing edges is more effective than perturbing nodes. Nonetheless, in chapter 8, we showed empirically and analytically that perturbing nodes is more harmful than perturbing edges, when evaluated in realistic and comparable settings.
Bibliography


[BBL+18] Joseph Bethge, Marvin Bornstein, Adrian Loy, Haojin Yang, and


[FHTC19] Fuli Feng, Xiangnan He, Jie Tang, and Tat-Seng Chua. Graph adversarial training: Dynamically regularizing based on graph structure. IEEE Transactions on Knowledge and Data Engineering, 2019.


156


[WK18] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In Jennifer Dy and


звучרא אקריאית. עבודה זה כלל המתרים מתחדשים של השיטות המставленות, תרשימים מרפאות, הנוסחאות והדוגמאות, בנוסח ניסיוני חדש רבים ואת ביצועי השיטות המ耄עת של השיטות הקלאסיות על ממונח הורבק על תכניות מרפאות. בנוסח, מתארים מחקרים שונים לברטאות ואת עילוט היחסים של ניסיון בקצף אוון. כל פרק בسفر זה כולל בקצף הבה: מוניטוביציה לעובדה והבריקות, ואת нояוספיסוס של השיטות והניסיונות בתכנית, ולספק את כתובות המגנכים את הענישה לכל מתן פר RESP תובות למקרא המוסכום.
תקציר
בשנים 최근imas什么是を使用する学習, 深度学習による複雑な問題を解決するさまざまな分野で進歩があります。最然結果を提示する新しい手法と原理が提案されたが、それでも収縮前の深度学習による多くの問題がまだ残っています。この問題の主な原因は、環境の乱れに対する抵抗性が低く、また、エネルギー消費が多いことです。リスクを伴う深い学習を実現するには、エネルギー消費を減らすために、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。技術の進歩により、コンパクトでパワーサイズが小さいソーシャルシステムの開発が必要です。


תكنולוגיה רשתות למידהعمוקה לשיפור ריבונレストית

�名 מעחיר

לשם מולי חלקי של הדרישת להובלת התואר

דרקונר לפילוסופיה

חיבת בסקין

הוגש לתוכנית תכניון-Mその他 טכנולוגיה ליבוא

נוכך התשס"א א' חיפה מאי 2021
תבנונ ישנות למידה ע العسكري תלסקה עיילה
ורובסטיית

חירם בשקין