Exploring Binarization and Pruning of Convolutional Neural Networks

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science in Computer Science and Engineering by Research

by

Ameya Prabhu
201402004
ameya.prabhu@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
July 2019

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Exploring Binarization and Pruning of Convolutional Neural Networks" by Ameya Prabhu, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Adviser: Dr. Anoop Namboodiri

To understanding colourless green CNNs which dream furiously

Acknowledgments

I was very lucky to have Riddhiman Dasgupta as my mentor. He was the one who helped me take my first steps as a researcher back in 2015. Along with Koustav Ghosal, they were the instrumental duo in guiding me through "research 101". I would like to thank my research team "The Roaches": Sri Aurobindo Munagala, Rohit Gajawada and Vishal Batchu who made this journey memorable, the times we worked together, attacked challenging problems and won some battles. I am still proud of every paper we wrote. Exploring silly ideas since 2015 somehow ended up in this thesis.

Unlike the writing and procedural aspects of this thesis, which is usually an exercise in sustained suffering, the journey itself was one of my most memorable. I thank my best friend Shreedhar Manek, without whom I cannot imagine my college life. I've spent some of the best times with him: traveled mountains, explored jungles, become vegan. Along with my friends Shantanu, Dhruv, Aalekh, Anshul and other wingmates- the sole undivided wing. I would like to thank Auro, Anurag, Tushant, Arjun, Saujas, Alok and Shyam who awoke in me curiosity for philosophy, awareness of social and economic issues, awe of the great historical traditions, good writing and poetry among other things.

I love research. However, I largely attribute success in my research to luck. I thank the metaphorical God for the opportunity of making me the coin flip which won the lottery at every single step, so many times. This was augmented by some important causal factors. The largest of them are: Dr. Anoop Namboodiri provided me with the guidance, support, and tremendous amounts of freedom to explore. Dr. Maneesh Singh taught me how to think about a problem, a principled way of doing research: asking the right questions, identifying real issues, dissuade random exploration and post-facto reasoning. Dr. Manish Shrivastava taught me a respect for language (domain) while applying ML algorithms, and the resulting elegance when combined correctly. Actually, let's say they tried their best. I hope I learned. Yes, looking back they all displayed one quality: incredible patience in tolerating my youthful pride and ignorance. These four years of research, more than anything, have been a humbling experience.

I now see a glimpse of the sheer depth of vision, linguistics, statistics and machine learning literature. I realize how long a path lies ahead and have the determination to walk it with the goal of helping to alleviate some of the most pressing problems people face in the world.

Abstract

Deep learning models have evolved remarkably, and are pushing the state-of-the-art in various problems across domains. At the same time, the complexity and the amount of resources these DNNs consume has greatly increased. Today's DNNs are computationally intensive to train and run, especially Convolutional Neural Networks (CNNs) used for vision applications. They also occupy a large amount of memory and consume a large amount of power during training. This poses a major roadblock to the deployment of such networks, especially in real-time applications or on resource-limited devices. Two methods have shown promise in compressing CNNs: (i) Binarization and (ii) Pruning. We explore these two methods in this thesis.

The first method to achieve improvements in computational/spatial efficiency is to binarize (1-bit quantize) the weights and activations in a network. However, naive binarization results in accuracy drops for most tasks. In this work, we present a Distribution-Aware approach to Binarizing Networks (DABN) that allows us to retain the advantages of a binarized network, while improving accuracy over binary networks. We also develop efficient implementations of DABN across different architectures. We present a theoretical analysis of DABN to show the effective representational power of the resulting layers, and explore the forms of data they model best. Experiments on popular sketch datasets show that DABN offers better accuracies than naive binarization.

We further investigate the question of where to binarize inputs at layer-level granularity and show that selectively binarizing the inputs to specific layers in the network could lead to significant improvements in accuracy while preserving most of the advantages of binarization. We analyze the binarization trade-off using a metric that jointly models the input binarization error and computational cost. We introduce an efficient algorithm to select layers whose inputs are to be binarized. We discuss practical guidelines based on insights obtained from applying the algorithm to a variety of models. Experiments on the Imagenet dataset using AlexNet and ResNet-18 models show 3-4% improvement in accuracy over fully binarized networks with minimal impact on compression. The improvements are even more substantial on sketch datasets like TU-Berlin, where we match state-of-the-art accuracy, getting more than 8% increase in accuracies over binary networks. We further show that our approach can be applied in tandem with other forms of compression that deal with individual layers or overall model compression (e.g., SqueezeNets). In contrast to previous binarization approaches, we are able to binarize the weights in

the last layers of a network, which enables us to compress a large fraction of additional parameters.

The second method explored is pruning. We investigate pruning neural networks from a graph-theoretic perspective. Efficient CNN designs like ResNets and DenseNet were proposed to improve accuracy vs efficiency trade-offs. They essentially increased the connectivity, allowing efficient information flow across layers. Inspired by these techniques, we propose to model connections between filters of a CNN using graphs which are simultaneously sparse and well-connected. Sparsity results in efficiency while well-connectedness can preserve the expressive power of the CNNs. We use a well-studied class of graphs from theoretical computer science that satisfies these properties known as Expander graphs. Expander graphs are used to model connections between filters in CNNs to design networks called X-Nets. We present two guarantees on the connectivity of X-Nets: (i) Each node of a layer influences every node in a layer $O(logn)$ steps away, where n is the number of layers between the two layers (ii) The number of paths between two sets of nodes is proportional to the product of their sizes. We also propose efficient training and inference algorithms, making it possible to train deeper and wider X-Nets effectively. Expander based models give a 4% improvement in accuracy on MobileNet over grouped convolutions, a popular technique which has the same sparsity but worse connectivity. X-Nets give better performance trade-offs than the original ResNet and DenseNet-BC architectures. We achieve model sizes comparable to state-of-the-art pruning techniques using our simple architecture design, without any pruning.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

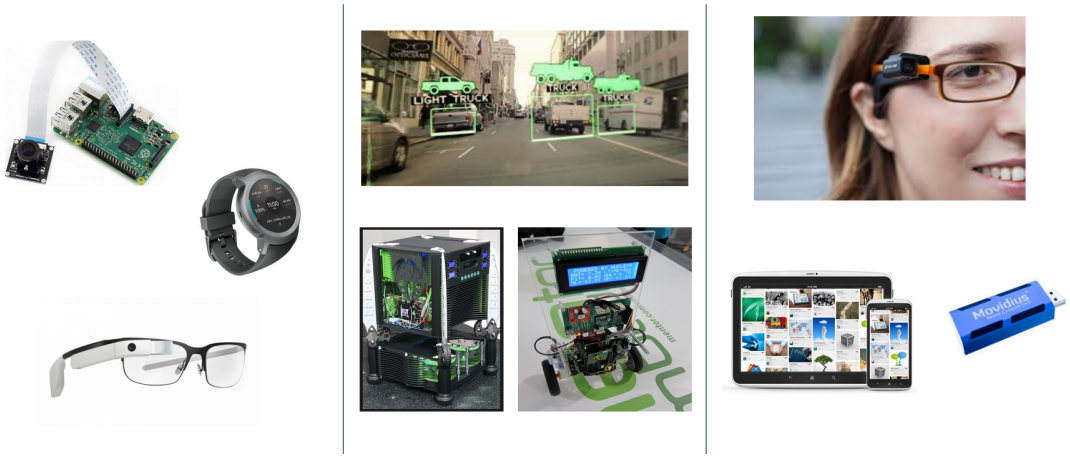


Figure 1.1: There are two significant roadblocks to efficient deployment of large CNN models: realtime computation, and in-memory compactness. Cloud is not an option due to these real-time operation constraints.

Convolutional Neural Networks (CNNs) have found applications in many vision-related domains ranging from generic image-understanding for self-driving cars [4] and automatic image captioning [37, 77] to recognition of specific image parts for scene-text recognition [56, 58] and face-based identification [71].

In the domain of image classification, AlexNet [42] and several architectural improvements such as VGG-Net [65] were proposed to push the limits of accuracy. These models were massive both in terms of memory usage and computational costs. AlexNet has around 60 million parameters in the network, while VGG has around 138 million, requiring 1.5 billion FLOPs and 19.6 billion FLOPs respectively for inference. The computational requirements make these architectures inappropriate for smaller portable systems such as mobiles and other embedded systems. These networks also consume large amounts of energy, creating a bottleneck for performance improvements. Full-precision multiply-accumulate (MAC) operations in floating point convolutional layers, for example, consumed 30x more power than

Method	Compression
Finetuned SVD 2 [76]	2.6x
Circulant CNN 2 [9]	3.6x
Adaptive Fastfood-16 [76]	3.7x
Collins *et al*. [12]	4x
Zhou *et al*. [84]	4.3x
ACDC [57]	6.3x
Network Pruning [22]	9.1x
Deep Compression [22]	9.1x
GreBdec [81]	10.2x
Srinivas *et al*. [68]	10.3x
Guo *et al*. [21]	**17.9x**
Binarization	**≈32x**

Table 1.1: Comparison of Binarization and other methods in terms of compression.

integer MAC operations (see Table 1.2). These issues are addressed in two ways: (i) developing new compressed architectures (ii) compressing a given architecture.

There have been many developments in the area of model compression in the last few years, with the aim of bringing down runtimes and storage requirements to alleviate these challenges. Compression strategies for convolutional neural networks include architectural improvements [25, 35], reparametrization of fully connected layers [57, 76], pruning techniques [22, 52], and quantization [14, 85]. Among these approaches, binarization and pruning provided the most compact models as shown in Table 1.1. In this chapter, we motivate our exploration of binarization and pruning of convolutional neural networks, provide a background of methods explored in the past, and state the concrete contributions made in this thesis.

1.1 Quantization

Quantized networks are networks where weights/activations are quantized into low-precision representations. These techniques achieve the best model compression and several promising directions have been explored in this domain [22]. They are classified into the categories based on the following questions:

1. How do they approximate the floating point values?

2. Which part of the network do they quantize?

Weights are mainly appproximated by two broad methods: deterministic rounding and stochastic rounding. In deterministic rounding, there is an one-to-one mapping between the quantized value and the real value, while in stochastic quantization, there is a sampling distribution which assigns the quantized value. We constrain ourselves to deterministic rounding in this work, primarily due to it being imple-

Operation	MUL	Power	ADD	Power
32-bit Float	3.7pJ	18.5x	0.9pJ	30x
16-bit Float	1.1pJ	5.5x	0.4pJ	13.3x
8-bit Integer	0.2pJ	1x	0.03pJ	1x

Table 1.2: As shown by Horowitz *et al*. [28], power consumption for various operations at 45nm 0.9V. Observe that 8-bit integers require significantly less energy than their equivalent 32-bit floating point operations.

mentable in hardware. There are two parts of a network which are targets of quantization: weights and activations. We take both steps, quantizing weights and activations to achieve compact models and lower computational cost.

Quantization has proven to be a powerful compression strategy, especially in its most extreme form: binarization. Binarization has enabled the use of XNOR-Popcount operations for vector dot products, which take much less time than full-precision Multiply-Accumulates (MACs), contributing to a huge speedup in convolutional layers [14, 60] on a general-purpose CPU. Moreover, as each binary weight requires only a single bit to represent, one can achieve drastic reductions in run-time memory requirements. Previous research [14, 60] shows that it is possible to perform weight and activation binarization on large networks with up to 58x speedups and approximately 32x compression ratios, albeit with significant drops in accuracy [60]. Later works moved away from binary representations of weights/inputs to multi-bit representations. The reason for this was mainly the large accuracy drops observed in binary networks.

This leads to the natural question of whether, in theory, binary-representations of neural networks can be used at all to effectively approximate a full-precision network. If shown to be sufficient, the search for an accurate binarization technique is worthwhile, due to the large gains in speedups (due to binary operations rather than full-precision MACs) and compression compared to multi-bit representations. We also offer intuitions about how this technique might be effective in problems involving data that is binary in nature, such as sketches.

We then explore the problem of hybrid binarization of a network. We propose a technique devised from our investigation into the question as to *where and which quantities of a network should one binarize*, with respect to inputs to a layer – to the best of our knowledge, this is the first work that explores this question. We observe that in a trained fully binarized model, binarization leads to minimal error in some layers and significant errors in others. Our proposed partition algorithm, when run on trained fully binarized models can design effective architectures. When these hybrid models are trained from scratch, they achieve a balance between compression, speedup, energy-efficiency, and accuracy. We

conduct extensive experiments applying our method to different model architectures on popular large-scale classification datasets over different domains. The resulting models achieve significant speedups and compression with significant accuracy improvements over a fully binarized network.

1.2 Compact layers (Pruning)

Network pruning is widely used technique for reducing the inference cost of deep models in low-resource settings. A typical pruning algorithm is a three stage pipeline, i.e., training (a large model), pruning and finetuning. During pruning, according to a certain criterion, redundant weights are pruned and important weights are kept to preserve the accuracy. All commonly used pruning techniques center around exploiting the sparsity in neural networks. A byproduct of pruning is that it removes redundancies during retraining, helping to prevent overfitting.

On the other hand, developing a compact layer for deep networks helps save memory and computational costs. Architectures such as ResNet [25], DenseNet [33] significantly reduced model size compared to VGG-Net by proposing a bottleneck structure to reduce the number of parameters while improving speed and accuracy. Additionally, they directed the focus of efficient designs of convolutional layers on increasing connectivity. Additional connectivity with residual connections to previous layers provided efficient information flow through the network, enabling them to achieve an order of magnitude reduction in storage and computational requirements. We take inspiration from these approaches, to focus on designing highly connected networks. We explore making networks efficient by designing sparse networks that preserve connectivity properties.

Recent architectures like MobileNet [30] improves the efficiency by an order of magnitude over a ResNet. However, in order to achieve this, they sparsify a network by removing several connections from a trained network, reducing their accuracies in the process. We ask a basic question: If we try to maximize the connectivity properties and information flow, can we achieve the same efficiency gains with minimal loss in accuracy? It is essential that the connections allow information to flow through the network easily. That is, each output node must at least have the capacity to be sensitive to features of previous layers. Traditional model compression techniques such as pruning can aggravate the problem, since they can prune the neuron connections of a layer, while being agnostic of global connectivity of the network. A necessary condition for having good representational power is efficient information flow through the network, which is particularly suited to be modeled by graphs.

We propose to make the connections between neurons (filters in the case of CNNs) according to specific graph constructions known as expander graphs. They have been widely studied in spectral graph theory [66] and pseudorandomness [73], and are known to be sparse but highly connected graphs. Expander

graphs have a long history in theoretical computer science, also being used in practice in computer networks, constructing error correcting codes, and in cryptography (for a survey, see [27]).

1.3 Contributions

Overall, in this thesis we make the following contributions:

1. Binarization

   (a) We show that binary representations are as expressive as full precision neural networks for polynomial functions, and offer theoretical insights into the same.

   (b) We present a generalized, distribution-aware representation for binary networks, and proceed to calculate the generalized parameter-values for any binary network. We offer an intuitive analysis and comparison of our representation vis-a-vis existing representations.

   (c) We provide a provably efficient implementation of networks trained using this representation. We demonstrate the effectiveness of our method by extensive experiments applying it to popular model architectures on large-scale sketch datasets and improving upon existing binarization approaches.

   (d) We propose a metric to jointly optimize binarization-errors of layers and the associated computational costs and a partitioning algorithm to find suitable layers for input binarization, based on the above metric, which generates hybrid model architectures which if trained from scratch, achieve a good balance between compression, speedup, energy-efficiency, and accuracy.

   (e) Insights into what the algorithm predicts, which can provide an intuitive framework for understanding why binarizing certain areas of networks demonstrably achieves significant compression with respect to other compression methods; with hybrid model architectures for AlexNet, ResNet-18, Sketch-A-Net and SqueezeNet with over 5-8% accuracy improvements on various datasets like Imagenet and TU-Berlin sketch recognition;

2. Pruning (Compact Layer Design)

   (a) We propose to represent neuronal connections in deep networks using expander graphs (see Section 5.1). We further prove that X-Nets have strong connectivity properties (see Theorem 11).

   (b) We provide memory-efficient implementations of Convolutional (X-Conv) layers using sparse matrices and propose a fast expander-specific algorithm and empirically compare X-Conv layers with grouped convolutions that have the same level of sparsity but worse connectivity. X-Conv layers obtain a 4% improvement in accuracy when both the techniques are applied to the MobileNet architecture trained on Imagenet (see Section 5.3.1).

(c) We demonstrate the robustness of our approach by obtaining better performance trade-offs when applied to some of the state of the art models like DenseNet-BC and ResNet with a simple design, additionally achieving comparable compression rates to even the state-of-the-art trained pruning techniques.

(d) Since we enforce the sparsity before the training phase itself, our models are inherently compact and faster to train compared to pruning techniques. We leverage this and showcase the performance of wider and deeper X-Nets.

We shall be diving into the details of each contribution in chronological order in this thesis. The next chapter shall explore existing literature and pose our contributions in context and how the proposed contributions are novel.

Chapter 2

Related Work

CNNs have a large number of redundant parameters, increasing memory and computational cost. Since these applications would be deployed on resource-constrained systems, CNN compression is an important emerging area for research on vision applications [14, 22, 25, 35, 52, 57, 76, 85]. In this chapter, we briefly point out the two areas which are similar to our work and mention some of the important contributions made in network compression literature. They are categorized as follows: (i) Network Pruning (ii) Network Quantization. We also describe hardware accelerators available for quantized networks to tie-in with some proposed algorithms. We detail approaches like efficient network design and using low-rank approximations to reduce the number of parameters of a network. A detailed overview of approaches to network compression can be found in surveys such as [8, 69].

2.1 Network Pruning

Some of the earliest work in network pruning were optimal brain damage [45] and optimal brain surgeon [24] used a function of the Hessian matrix of the loss function to prune a network by reducing the number of connections. With the recent sucess of deep learning, network pruning methods have risen to prominence. Weight-level pruning has the highest compression rate while filter-level pruning is easier to practically exploit , with compression rates not far behind the former. Hence, filter-level pruning is currently considered superior [52].

2.1.1 Pixel and Vector Pruning

Optimal brain damage and optimal brain surgeon both belonged to this category. They pruned individual parameters of a network. The problem of pruning weights during train-time have been extensively studied in the literature [69]. Deep compression [23] also used pruning to achieve compression by an order of magnitude in various standard neural networks. SSL [75] regularizes filter values and shapes, along with depth structures. It also performs structured pruning in a channel-wise and filter-wise manner. Dynamic network surgery [21] builds upon these methods to incorporate connection splicing into

pruning, implementing the pruning process in a dynamic way and achieving state-of-the-art compact networks.

2.1.2 Channel and Filter Pruning

Srinivas *et al*. [67] propose a systematic data-free method to remove neurons instead of removing individual connections as above. Li *et al*. [47] removed entire filters in the network along with their connecting feature maps, reducing the computation costs significantly. ThiNet [53] proposed to prune filters from a CNN to enable more efficient acceleration. He *et al*. [26] propose an iterative two-step algorithm to effectively prune each layer, by a LASSO regression-based channel selection procedure for weights and minimize feature map errors by least square reconstruction. Network slimming [52] proposed to zero out channels of inputs based on the scaling factor of the batch normalization layer as a measure of importance of each channel in the input.

Groupwise brain damage [44] prunes the convolutional kernel tensor in a group-wise fashion by adding group-level sparsity regularization to the standard training process. Group sparse regularization [64] proposed a sparse group lasso penalty in order to impose group-level sparsity on the networks connections. Group-level pruning has a practical speed-up which scales linearly with the sparsity level.

2.2 Quantization

There has been a major body of work that quantizes the networks at train-time to achieve efficiency, explored in the literature survey by Yunhui Guo [20].

2.2.1 Higher bit quantization

HashedNets [7] performed binning of network weights using hash functions. Deep compression [22] introduced quantization into the current literature by employing trained quantization and Huffman coding to reduce the non-runtime memory from Han *et al*. [23]. Zhou *et al*. [2] quantized networks to 4-bit weights, achieving 8x memory compression by using 4 bits to represent 16 different values and 1 bit to represent zeros. Trained ternary quantization [10] uses 2-bit weights and scaling factors to compress the model by 16x with little accuracy degradation. Ternary weight networks [46] optimize a threshold-based ternary function for approximation, with stronger expressive abilities than binary networks. The above works are limited in the sense that they cannot leverage most speedups obtained by quantization since general purpose hardware supports only 8-bit integers or 1-bit booleans. Binary networks, on the other hand gain speedups by `xnor-popcount` operations which could be performed on dedicated hardware. Similarly, they can be practically compressed by using the existing boolean datatype.

2.2.2 1-bit quantization

BinaryConnect [13] was one of the first works to use binary (+1, -1) values for network parameters, achieving significant compression. Binary neural networks [14] quantized weights and activations of neural networks and achieved near full-precision accuracies for datasets like MNIST, SVHN and CIFAR10. Quantized neural networks [34] use low-precision quantized weights and inputs and replace floating-point arithmetic operations with bit-wise ones. XNOR-Nets [60] followed the work of BNNs [34], binarizing both layer weights and inputs and multiplying them with scaling constants - bringing significant accuracy improvements. DoReFa-Net [85] used low bit-width gradients during backpropagation, and obtained train-time speedups. HWGQ-Net [6] introduces a better suited activation function substituting sign function for better training of quantized networks. Recent research has proposed a variety of additional methods: including novel activation functions [6], fixed-point bit-width allocations [48], etc. HTCBN [72] introduce helpful techniques such as replacing ReLU layers with PReLU layers and a scale layer to recover accuracy loss on binarizing the last layer. Hou *et al*. [29] use Hessian approximations to minimize loss w.r.t. binary weights during training. Anderson *et al*. [1] offer a theoretical analysis of the workings of binary networks in terms of high-dimensional geometry.

Merolla *et al*. [55] show that binary networks during testing exhibit a remarkable robustness to distortions beyond quantization, including additive and multiplicative noise, and a class of non-linear projections. Further works have extended this in various directions, including using local binary patterns [38] which proposed an LBC layer, comprising of a set of fixed sparse pre-defined binary convolutional filters that are not updated during the training process. Lookup-based compression methods [3] extended that by encoding convolutions by few lookups to a dictionary that is trained to cover the space of weights in CNNs - with training jointly learning a dictionary and a small set of linear combinations of the it.

2.3 Other network compression approaches

We cover several other aspects explored in the network compression literature, such as low-rank approximations, efficient network architecture design for image recognition and other tasks like sketch recognition.

2.3.1 Low-rank Approximations

Several methods since [62] have been introduced to compress pretrained networks as well as train-time compression by approximation weight matrices by their low-rak approximations [54, 59]. [36] approximate a learnt full rank filter bank as combinations of a rank-1 filter basis (wh into $w1$ and $1h$ filters). Denil *et al*. [17] use a low-rank approximation using SVD to reduce the network redundancy. [39] used Tucker decomposition method to get the low-rank approxmation. [43] developed a low-rank

CP-decomposition of the 4D convolution kernel tensor into a sum of a small number of rank-one tensors using non-linear least squares to replace the original layer.

2.3.2 Sketch Recognition

Works before [80] did not lead to good results by applying image classification architectures, since they are better suited to images than sketches [80]. Sketches require specialized, fine-tuned networks since they have significantly different characteristics as compared to images. Sketch-a-Net from Yu *et al*. [79, 80] took these factors into account, and proposed a carefully designed network structure that suited sketch representations. Their model showed tremendous increments over the then state-of-the-art, and managed to beat the average human performance using late and early fusion of ensembles. This model has been adopted by a number of later works such as Bui *et al*. [5], Yu *et al*. [78], Wang *et al*. [74].

2.3.3 Efficient Architecture Design

Currently there is extensive interest in developing novel convolutional layers/blocks and effectively leveraging them to improve architectures like [30, 31, 35]. In contrast, approaches like [70] try to design the macro-architectures by connecting pre-existing blocks. Recent concurrent work has been on performing architecture searches effectively [50, 51, 83, 86]. Our work is complementary to architecture search techniques as we can leverage their optimized macro-architectures.

Another line of efficient architecture design is grouped convolutions, first proposed in AlexNet [42] and popularized by MobileNets [30] and XCeption [11] architectures . [32, 63, 82] are the latest works in this area.

It is interesting to note that recent breakthroughs in designing accurate deep networks [25, 33] were made by introducing additional connectivity to enable the efficient flow of information through deep networks. This enables the training of compact, accurate deep networks. These approaches, along with grouped convolutions are closely related to our approach.

2.4 Novelty

We ask the question: Do CNNs need the representational power of 32-bit floating point operations? Is it possible to cut down memory costs and make output computations significantly less expensive? Previous literature performed binarization independent of the distribution weights. In Chapter 3, we introduce distribution-aware binarization method which depends on the distribution of weights to calculate an optimal binarization.

Unlike previous work in this area, we look at binarizing specific parts of a network, instead of simply binarizing the inputs to all the layers end-to-end. We see in Chapter 4, binarizing the right areas in the network contributes significantly to the overall accuracy of the network without compromising on speed-ups.

To the best of our knowledge, the method we propose in Chapter 5 is the first attempt at constraining neural network connections by graph-theoretic approaches to improve deep network architecture designs. We differ from previous work by not pruning weights during training, and still achieve comparable performance.

Chapter 3

Distribution Aware Binarization of Networks

In this chapter, we will explore the first question: How to form a distribution aware method of compressing binary neural networks. As illustrated in Figure 3.1 that changes in the binarization function can influence it's ability to detect portions of an image more accurately.

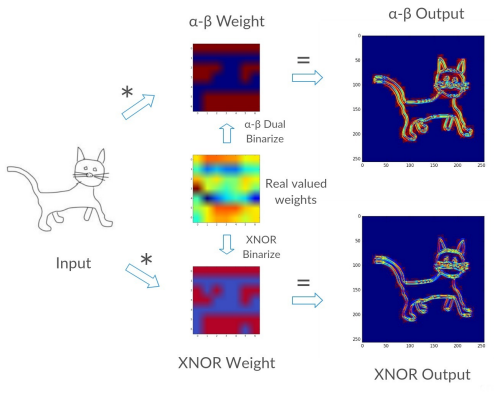


Figure 3.1: An example sketch passing through a convolutional layer filter, with the real-valued filter shown alongside corresponding α-β and XNOR-Net filters. Orange signifies the highest response areas. We can see that DAB-Net has significantly better responses when compared to XNOR-Net

3.1 Representational Power of Binary Networks

Many recent works in network compression involve higher bit weight quantization using two or more bits [2, 46, 46] instead of binarization, arguing that binary representations would not be able to approximate full-precision networks. In light of this, we explore whether the representational power that binary networks can offer is theoretically sufficient to get similar representational power as full-precision networks.

Rolnick *et al*. [49,61] have done extensive work in characterizing the expressiveness of neural networks. They claim that due to the nature of functions - that they depend on real-world physics, in addition to mathematics - the seemingly huge set of possible functions could be approximated by deep learning models. From the Universal Approximation Theorem [15], it is seen that any arbitrary function can be well-approximated by an Artificial Neural Network; but *cheap learning*, or models with far fewer parameters than generic ones, are often sufficient to approximate multivariate monomials - which are a class of functions with practical interest, occurring in most real-world problems.

We can define a binary neural network having k layers with activation function $\sigma(x)$ and consider how many neurons are required to compute a multivariate monomial $p(x)$ of degree d. The network takes an n dimensional input $\mathbf{x}$, producing a one dimensional output $p(x)$. We define $B_k(p, \sigma)$ to be the minimum number of binary neurons (excluding input and output) required to approximate p, where the error of approximation is of degree at least $d + 1$ in the input variables. For instance, $B_1(p, \sigma)$ is the minimal integer m such that:

$$\sum_{j=1}^{m} w_j \sigma \left(\sum_{i=1}^{n} a_{ij} x_i \right) = p(x) + \mathcal{O}(x_1^{d+1} + \ldots + x_n^{d+1}).$$

Any polynomial can be approximated to high precision as long as input variables are small enough [49]. Let $B(p, \sigma) = \min_{k \geq 0} B_k(p, \sigma)$.

Theorem 1 *For $p(\mathbf{x})$ equal to the product $x_1 x_2 \cdots x_n$, and for any σ with all nonzero Taylor coefficients, we have one construction of a binary neural network which meets the condition*

$$B_k(p, \sigma) = \mathcal{O}\left(n^{(k-1)/k} \cdot 2^{n^{1/k}} \right). \tag{3.1}$$

Proof of the above can be found in the appendix A.

Conjecture III.2. of Rolnick *et al*. [61] says that this bound is approximately optimal. If this conjecture is true, weight-binarized networks would have the same representational power as full-precision networks, but with finite (1-bit) precision. Another case would be 2 layer network, where we can say a binary network requires the same number of neurons as a full-precision network, 2^n.

Theorem 2 *For any desired accuracy $\epsilon > 0$, there exists a binary neural network $B_1(p, \sigma)$ that can compute the function $p(x)$ using a single hidden layer of 2^n binary-weighted neurons.*

Proof of the above can be found in the appendix A.

The above theorem shows that any neural network that can be represented as a multivariate polynomial function is considered as a simplified model with ELU-like activations, using continuously differentiable layers - so pooling layers are excluded as well. While there can exist a deep binary-weight network

that can possibly approximate polynomials similar to full precision networks, it does say that such a representation would be efficiently obtainable through Stochastic Gradient Descent. Also, this theorem assumes only weights are binarized, not the activations. Activation binarization typically loses a lot of information and might not be a good thing to do frequently. However, this insight motivates the fact that more investigation is needed into approximating networks through binary network structures.

3.2 Distribution-Aware Binarization

We have so far established that binary representations are possibly sufficient to approximate a polynomial with similar numbers of neurons as a full-precision neural network. We now investigate the question - What is the most general form of binary representation possible? In this section, we derive a generalized distribution-aware formulation of binary weights, and provide an efficient implementation of the same. We consider models binarized with our approach as DAB-Nets (Distribution Aware Binarized Networks).

We model the loss function layer-wise for the network. We assume that inputs to the convolutional layers are binary - i.e. belong to $\{+1, -1\}$, and find constants α and β (elaborated below) as a general binary form for layer weights. These constants are calculated from the distribution of real-valued weights in a layer - thus making our approach *distribution-aware*.

3.2.1 Derivation

Without loss of generality, we assume that $\mathbf{W}$ is a vector in R^n , where $n = c \cdot w \cdot h$. We attempt to binarize the weight vector $\mathbf{W}$ to $\widetilde{\mathbf{W}}$ which takes a form similar to this example - $[\alpha\alpha...\beta\alpha\beta]$. Simply put, $\widetilde{\mathbf{W}}$ is a vector consisting of scalars α and β, the two values forming the binary vector. We represent this as $\widetilde{\mathbf{W}} = \alpha\mathbf{e} + \beta(\mathbf{1} - \mathbf{e})$ where $\mathbf{e}$ is a vector such that $\mathbf{e} \in \{0, 1\}^n \ni \mathbf{e} \neq 0$ and $\mathbf{e} \neq 1$. We define K as $\mathbf{e}^T\mathbf{e}$ which represents the number of ones in the $\mathbf{e}$ vector. Our objective is to find the best possible binary approximation for $\mathbf{W}$. We set up the optimization problem as:

$$\widetilde{\mathbf{W}}^* = \underset{\widetilde{W}}{argmin} \mid\mid \mathbf{W} - \widetilde{\mathbf{W}} \mid\mid^2$$

We formally state this as the following:

The optimal binary weight vector $\widetilde{\mathbf{W}}^*$ *for any weight vector* $\mathbf{W}$ *which minimizes the approximate-error function* $\mathbf{J} = \mid\mid \mathbf{W} - \widetilde{\mathbf{W}} \mid\mid^2$ *can be represented as:*

$$\widetilde{\mathbf{W}}^* = \alpha\mathbf{e} + \beta(\mathbf{1} - \mathbf{e}) \; where$$

$$\alpha = \frac{\mathbf{W}^T\mathbf{e}}{K} \; , \; \beta = \frac{\mathbf{W}^T(\mathbf{1} - \mathbf{e})}{n - K}$$

for a given K. That is, given a K, the optimal selection of $\mathbf{e}$ would correspond to either the K smallest weights of $\mathbf{W}$ or the K largest weights of $\mathbf{W}$.

The best suited K, we calculate the value of the following expression for every value of K, giving us an $\mathbf{e}$, and maximize the expression:

$$\mathbf{e}^* = \underset{e}{argmax}(\frac{|| \mathbf{W}^T\mathbf{e} ||^2}{K} + \frac{|| \mathbf{W}^T(\mathbf{1} - \mathbf{e}) ||^2}{n - K})$$

A detailed proof of the above can be found in the appendix A.

The above representation shows the values obtained for $\mathbf{e}$, α and β are the optimal approximate representations of the weight vector $\mathbf{W}$. The vector $\mathbf{e}$, which controls the number and distribution of occurrences of α and β, acts as a mask of the top/bottom K values of $\mathbf{W}$. We assign α to capture the greater of the two values in magnitude. Note that the scaling values derived in the XNOR formulation, α and $-\alpha$, are a special case of the above, and hence our approximation error is at most that of the XNOR error. We explore what this function represents and how this relates to previous binarization techniques in the next subsection.

3.2.2 Intuitions about DAB-Net

In this section, we investigate intuitions about the derived representation. We can visualize that $\mathbf{e}$ and $(\mathbf{1} - \mathbf{e})$ are orthogonal vectors. Hence, if normalized, $\mathbf{e}$ and $(\mathbf{1} - \mathbf{e})$ form a basis for a subspace R^2. Theorem 2 says the best α and β can be found by essentially projecting the weight matrix $\mathbf{W}$ into this subspace, finding the vector in the subspace which is *closest* to $\mathbf{e}$ and $(\mathbf{1} - \mathbf{e})$ respectively.

$$\alpha = \frac{\langle \mathbf{W}, \mathbf{e} \rangle}{\langle \mathbf{e}, \mathbf{e} \rangle} \cdot \mathbf{e} \ , \ \beta = \frac{\langle \mathbf{W}, (\mathbf{1} - \mathbf{e}) \rangle}{\langle (\mathbf{1} - \mathbf{e}), (\mathbf{1} - \mathbf{e}) \rangle} \cdot (\mathbf{1} - \mathbf{e})$$

We also show that our derived representation is different from the previous binary representations since we cannot derive them by assuming a special case of our formulation. XNOR-Net [60] or BNN [14]-like representations cannot be obtained from our formulation.

3.2.3 Implementation

The representation that we earlier derived requires to be efficiently computable, in order to ensure that our algorithm runs fast enough to be able to train binary networks. In this section, we investigate the implementation, by breaking it into two parts: 1) Computing the parameter K efficiently for every iteration. 2) Training the entire network using that value of K for a given iteration. We show that it is possible to get an efficiently trainable network at minimal extra cost. We provide an efficient algorithm using Dynamic Programming which computes the optimal value for K quickly at every iteration.

Algorithm 1 Finding an optimal K value.

```
Initialization
W = 1D weight vector
T = Sum of all the elements of W
Sort(W)
D = [00...0]  // Empty array of same size as W
optK_1 = 0  // Optimal value for K
maxD_1 = 0  // Value of D for optimal K value

for I= 1 to D.size do
    P_i = P_{i-1} + W_i
    D_i = P_i^2/i + (T-P_i)^2/(n-i)
    if D_i ≥ maxD_1 then
        maxD_1 = D_i
        optK_1 = i

Sort(W, reverse=true) and Repeat steps 4-13 with optK_2 and maxD_2

optK_final = optK_1
if maxD_2 > maxD_1 then
    optK_final = optK_2

return optK_final
```

Parallel Prefix-Sums to Obtain K

Theorem 3 *The optimal K^* which minimizes the value* $\mathbf{e}$ *can be computed in $O(n \cdot logn)$ complexity.*

Considering one weight filter at a time for each convolution layer, we flatten the weights into a 1-dimensional weight vector $\mathbf{W}$. We then sort the vector in ascending order and then compute the prefix-sum array P of $\mathbf{W}$. For a selected value of K, the term to be maximized would be $(\frac{||\mathbf{W}^T\mathbf{e}||^2}{K} + \frac{||\mathbf{W}^T(\mathbf{1}-\mathbf{e})||^2}{n-K})$, which is equal to $(\frac{P_i^2}{i} + \frac{(T-P_i)^2}{n-i})$ since the top K values in $\mathbf{W}$ sum up to P_i where T is the sum of all weights in $\mathbf{W}$. We also perform the same computation with a descending order of $\mathbf{W}$'s weights since K can correspond to either the smallest K weights or the largest K weights as we mentioned earlier. In order to speed this up, we perform these operations on all the weight filters at the same time considering them as a 2D weight vector instead. Our algorithm runs in $O(n \cdot logn)$ time complexity, and is specified in Algorithm 3. This algorithm is integrated into our code, and will be provided alongside.

Forward and Backward Pass

Now that we know how to calculate K, $\mathbf{e}$, α, and β for each filter in each layer optimally, we can compute $\widetilde{\mathbf{W}}$ which approximates $\mathbf{W}$ well. Here, $topk(\mathbf{W}, K)$ represents the top K values of $\mathbf{W}$ which remain as is whereas the rest are converted to zeros. Let $\mathbf{T_k} = topk(\mathbf{W}, K)$.

Corollary 1 (Weight Binarization) *The optimal binary weight $\widetilde{\mathbf{W}}$ can be represented as,*

$$\widetilde{\mathbf{W}} = \alpha.sgn(\mathbf{T_k}) + \beta.(1 - sgn(\mathbf{T_k}))$$

where,

$$\alpha = \frac{\mathbf{T_k}}{K} \; and \; \beta = \frac{(\mathbf{W} - \mathbf{T_k})}{n - K}$$

Once we have $\widetilde{\mathbf{W}}$, we can perform convolution as $\mathbf{I} \circledast \widetilde{\mathbf{W}}$ during the forward pass of the network. Similarly, the optimal gradient $\widetilde{\mathbf{G}}$ can be computed as follows, which is back-propagated throughout the network in order to update the weights:

Theorem 4 (Backward Pass) *The optimal gradient value $\widetilde{G}$ can be represented as,*

$$\widetilde{\mathbf{G}} = \widetilde{\mathbf{G_1}} + \widetilde{\mathbf{G_2}} \tag{3.2}$$

where,

$$\widetilde{\mathbf{G_1}} = \frac{sgn(\mathbf{T_k})}{K} \circ sgn(\mathbf{T_k}) + \frac{||\mathbf{T_k}||_{l1}}{K}.STE(\mathbf{T_k}) \tag{3.3}$$

$$\widetilde{\mathbf{G_2}} = \frac{sgn(\mathbf{W} - \mathbf{T_k})}{n - K} \circ (1 - sgn(\mathbf{T_k})) + \frac{||\mathbf{W} - \mathbf{T_k}||_{l1}}{n - K}.STE(\mathbf{W} - \mathbf{T_k}) \tag{3.4}$$

$$STE(\mathbf{T_k})^i = \begin{cases} {\mathbf{T_k}}^i, \textit{where } |\mathbf{W}|^i <= 1 \\ 0, \textit{ elsewhere} \end{cases} \tag{3.5}$$

The gradient vector, as seen above, can be intuitively understood if seen as the sum of two independent gradients $\widetilde{\mathbf{G_1}}$ and $\widetilde{\mathbf{G_2}}$, each corresponding to the vectors $\mathbf{e}$ and $(\mathbf{1} - \mathbf{e})$ respectively. Further details regarding the derivation of this gradient would be provided in the supplementary material.

3.2.4 Training Procedure

Putting all the components mentioned above together, we have outlined our training procedure in Algorithm 2. During the forward pass of the network, we first mean center and clamp the current weights of the network. We then store a copy of these weights as $\mathbf{W}_{real}$. We compute the binary forward pass of the network, and then apply the backward pass using the weights $\widetilde{\mathbf{W}}$, computing gradients for each of the weights. We then apply these gradients on the original set of weights $\mathbf{W}^t$ in order to obtain $\mathbf{W}^{t+1}$. In essence, binarized weights are used to compute the gradients, but they are applied to the original stored weights to perform the update. This requires us to store the full precision weights during training, but once the network is trained, we store only the binarized weights for inference.

Algorithm 2 Training an L-layers CNN with binary weights:

A minibatch of inputs and targets $(\mathbf{I}, \mathbf{Y})$, cost function $C(\mathbf{Y}, \hat{\mathbf{Y}})$, current weight $\mathbf{W}^t$ and current learning rate η^t.
updated weight $\mathbf{W}^{t+1}$ and updated learning rate η^{t+1}.
Binarizing weight filters:
$\mathbf{W}^t$ = MeanCenter($\mathbf{W}^t$)
$\mathbf{W}^t$ = Clamp($\mathbf{W}^t$, -1, 1)
$\mathbf{W}_{real}$ = $\mathbf{W}^t$
for $l = 1$ to L **do**
    for j^{th} filter in l^{th} layer **do**
        Find K_{lj} using Algorithm 3
        $\alpha_{lj} = \frac{topk(\mathbf{W}_{lj}, K_{lj})}{K_{lj}}$
        $\beta_{lj} = -\frac{(\mathbf{W}_{lj} - topk(\mathbf{W}_{lj}, K_{lj}))}{n - K_{lj}}$
        $\widetilde{\mathbf{W}}_{lj} = \alpha.sgn(topk(\mathbf{W}_{lj}, K_{lj}))$
            $+ \beta.(1 - sgn(topk(\mathbf{W}_{lj}, K_{lj})))$

$\hat{\mathbf{Y}} =$ **BinaryForward**$(\mathbf{I}, \widetilde{\mathbf{W}})$

$\frac{\partial C}{\partial \widetilde{\mathbf{W}}} =$ **BinaryBackward**$(\frac{\partial C}{\hat{\mathbf{Y}}}, \widetilde{\mathbf{W}})$ // Standard backward propagation except that gradients are computed using $\widetilde{\mathbf{W}}$ instead of $\mathbf{W}^t$ as mentioned in Theorem. 4

We then copy back the real weights in order to apply the gradients computed. $\mathbf{W}^t$ = $\mathbf{W}_{real}$

$\mathbf{W}^{t+1}$ = **UpdateParameters**$(\mathbf{W}^t, \frac{\partial C}{\partial \widetilde{\mathbf{W}}}, \eta^t)$
$\eta^{t+1} =$ **UpdateLearningrate**(η^t, t)

3.3 Experiments

We empirically demonstrate the effectiveness of our optimal distribution-aware binarization algorithm (DAB-Net) on the TU-Berlin and Sketchy datasets. We compare DAB-Net with BNN and XNOR-Net [60] on various architectures, on two popular large-scale sketch recognition datasets as sketches are sparse and binary. Also, they are easier to train with than standard images, for which we believe the algorithm needs to be stabilized - in essence, the K value must be restricted to change by only slight amounts. We show that our approach is superior to existing binarization algorithms, and can generalize to different kinds of CNN architectures on sketches.

3.3.1 Experimental Setup

In our experiments, we define the network having only the convolutional layer weights binarized as WBin, the network having both inputs and weights binarized as FBin and the original full-precision network as FPrec. Binary Networks have achieved accuracies comparable to full-precision networks

on limited domain/simplified datasets like CIFAR-10, MNIST, SVHN, but show considerable losses on larger datasets. Binary networks are well suited for sketch data due to its binary and sparse nature of the data.

TU-Berlin: The TU-Berlin [18] dataset is the most popular large-scale free-hand sketch dataset containing sketches of 250 categories, with a human sketch-recognition accuracy of 73.1% on an average.

Sketchy: A recent large-scale free-hand sketch dataset containing 75,471 hand-drawn sketches spanning 125 categories. This dataset was primarily used to cross-validate results obtained on the TU-Berlin dataset, to ensure the robustness of our approach with respect to the method of data collection.

For all the datasets, we first resized the input images to 256 x 256. A 224 x 224 (225 x 225 for Sketch-A-Net) sized crop was then randomly taken from an image with standard augmentations such as rotation and horizontal flipping, for TU-Berlin and Sketchy. In the TU-Berlin dataset, we use three-fold cross validation which gives us a 2:1 train-test split ensuring that our results are comparable with all previous methods. For Sketchy, we use the training images for retrieval as the training images for classification, and validation images for retrieval as the validation images for classification. We report ten-crop accuracies on both the datasets.

We used the PyTorch framework to train our networks. We used the Sketch-A-Net [80], ResNet-18 [25] and GoogleNet [70] architectures. Weights of all layers except the first were binarized throughout our experiments, except in Sketch-A-Net for which all layers except first and last layers were binarized. All networks were trained from scratch. We used the Adam optimizer for all experiments. Note that we do not use a bias term or weight decay for binarized Conv layers. We used a batch size of 256 for all Sketch-A-Net models and a batch size of 128 for ResNet-18 and GoogleNet models, the maximum size that fits in a 1080Ti GPU. Additional experimental details are available in the supplementary material.

3.3.2 Results

We compare the accuracies of our distribution aware binarization algorithm for WBin and FBin models on the TU-Berlin and Sketchy datasets. Note that higher accuracies are an improvement, hence stated in green in Table 4.3.

On the TU-Berlin and Sketchy datasets in Table 4.3, we observe that FBin DAB-Net models consistently perform better over their XNOR-Net counterparts. They improve upon XNOR-Net accuracies by 0.8%, 2.5%, and 1.5% in Sketch-A-Net, ResNet-18, and GoogleNet respectively on the TU-Berlin dataset. Similarly, they improve by 2.0%, 1.4%, and 0.6% respectively on the Sketchy dataset. We also compare them with state-of-the-art sketch classification models in Table 4.4. We find that our compressed models

Models	Method	Accuracies	
		TU-Berlin	Sketchy
Sketch-A-Net	FPrec	72.9%	85.9%
	WBin (BWN)	73.0%	85.6%
	FBin (XNOR-Net)	59.6%	68.6%
	WBin DAB-Net	72.4%	84.0%
	FBin DAB-Net	**60.4%**	**70.6%**
Improvement	XNOR-Net vs DAB-Net	+0.8%	+2.0%
ResNet-18	FPrec	74.1%	88.7%
	WBin (BWN)	73.4%	89.3%
	FBin (XNOR-Net)	68.8%	82.8%
	WBin DAB-Net	73.5%	88.8%
	FBin DAB-Net	**71.3%**	**84.2%**
Improvement	XNOR-Net vs DAB-Net	+2.5%	+1.4%
GoogleNet	FPrec	75.0%	90.0%
	WBin (BWN)	74.8%	89.8%
	FBin (XNOR-Net)	72.2%	86.8%
	WBin DAB-Net	75.7%	90.1%
	FBin DAB-Net	**73.7%**	**87.4%**
Improvement	XNOR-Net vs DAB-Net	+1.5%	+0.6%

Table 3.1: Our DAB-Net models compared to FBin, WBin and FPrec models on TU-Berlin and Sketchy in terms of accuracy.

perform significantly better than the original sketch models and offer compression, runtime and energy savings additionally.

Our DAB-Net WBin models attain accuracies similar to BWN WBin models and do not offer major improvements mainly because WBin models achieve FPrec accuracies already, hence do not have much scope for improvement unlike FBin models. Thus, we conclude that our DAB-Net FBin models are able to attain significant accuracy improvements over their XNOR-Net counterparts when everything apart from the binarization method is kept constant.

3.3.3 XNOR-Net vs DAB-Net

We measure how K, α, and β vary across various layers over time during training, and these variations are observed to be quite different from their corresponding values in XNOR-Net. These observations show that binarization can approximate a network much better when it is distribution-aware (like in our technique) versus when it is distribution-agnostic (like XNOR-Nets).

[1]It is the sketch-a-net SC model trained with additional imagenet data, additional data augmentation strategies and considering an ensemble, hence would not be a direct comparison

Models	Accuracy
AlexNet-SVM	67.1%
AlexNet-Sketch	68.6%
Sketch-A-Net SC	72.2%
Humans	73.1%
Sketch-A-Net-2[1] [79]	**77.0%**
Sketch-A-Net WBin DAB-Net	72.4%
ResNet-18 WBin DAB-Net	73.5%
GoogleNet WBin DAB-Net	**75.7%**
Sketch-A-Net FBin DAB-Net	60.4%
ResNet-18 FBin DAB-Net	71.3%
GoogleNet FBin DAB-Net	**73.7%**

Table 3.2: A comparison between state-of-the-art single model accuracies of recognition systems on the TU-Berlin dataset.

Variation of α and β across Time

We plot the distribution of weights of a randomly selected filter belonging to a layer and observe that α and β of DAB-Net start out to be similar to α and $-\alpha$ of XNOR-Nets, since the distributions are randomly initialized. However, as training progresses, we observe as we go from Subfigure (1) to (4) in Figure 3.3, the distribution eventually becomes non-symmetric and complex, hence our values significantly diverge from their XNOR-Net counterparts. This divergence signifies a better approximation of the underlying distribution of weights in our method, giving additional evidence to our claim that the proposed DAB-Net technique gives a better representation of layer weights, significantly different from that of XNOR-Nets.

Variation of K across Time and Layers

We define *normalized* K as the $\frac{K}{n}$ for a layer filter. For XNOR-Nets, K would be the number of values below zero in a given weight filter - which has minimal variation, and does not take into consideration the distribution of weights in the filter - as K in this case is simply the number of weights below a certain fixed global threshold, zero. However, we observe that the K computed in DAB-Net varies significantly across epochs initially, but slowly converges to an optimal value for the specific layer as shown in Figure 3.3.

We also plot the variation of *normalized* K values for a few randomly chosen filters indexes across layers and observe that it varies across layers, trying to match the distribution of weights at each layer. Each filter has its own set of weights, accounting for the differences in variation of K in each case, as shown in Figure 3.3.

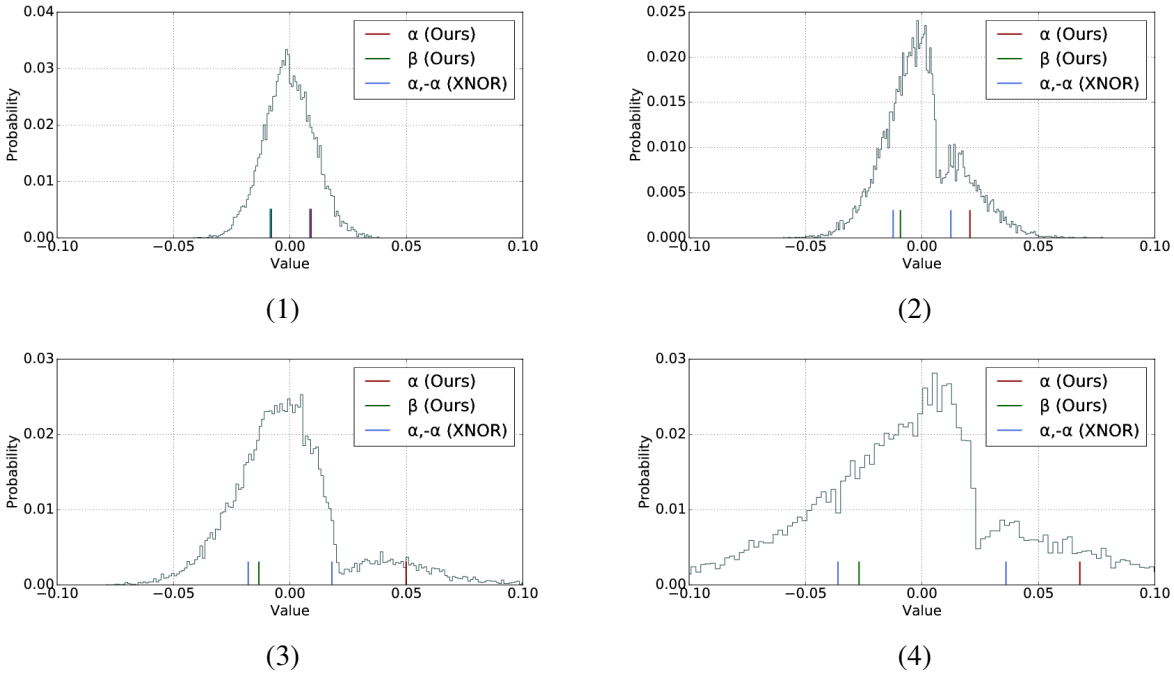
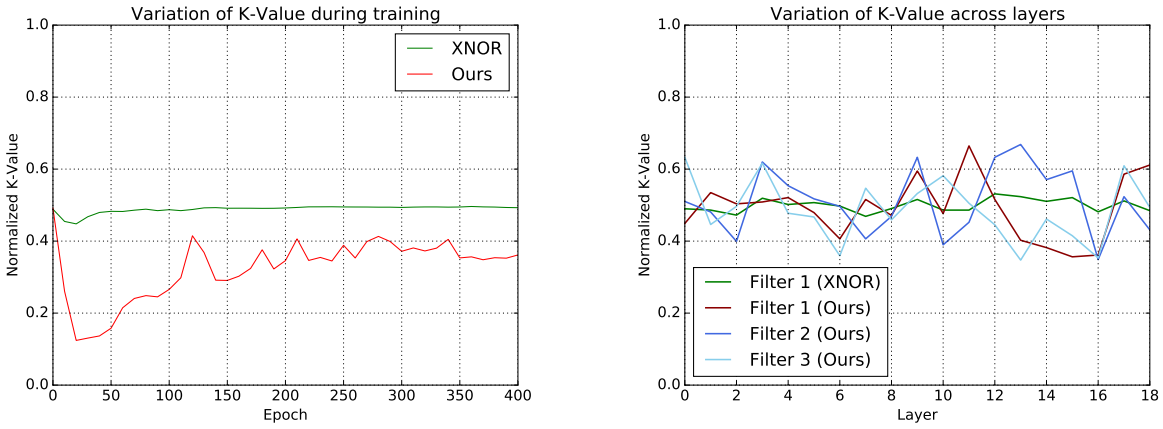


Figure 3.2: Sub-figures (1) to (4) show the train-time variation of α and β for a layer filter. Initially, α and β have nearly equal magnitudes, similar to the XNOR-Net formulation, but as we progress to (4), we see that α and β have widely different magnitudes.Having just one scaling constant (XNOR-Net) would be a comparatively poor approximator.

3.4 Summary

We have proposed an optimal binary representation for network layer-weights that takes into account the distribution of weights, unlike previous distribution-agnostic approaches. We showed how this representation could be computed efficiently in $n.logn$ time using dynamic programming, thus enabling efficient training on larger datasets. We applied our technique on various datasets and noted significant accuracy improvements over other full-binarization approaches.



(1) (2)

Figure 3.3: (1) shows the variation of the normalized K-value over time during training. It falls initially but converges eventually to 0.35. The normalized K-value for XNOR-Net remains almost at 0.5 till the end. (2) shows the variation of normalized K values on random filters across layers. The K-value corresponding to DAB-Net varies across layers based on the distribution of weights of the specific layer, which is not captured by XNOR-Net.

Chapter 4

Hybrid Binary Networks

As Figure 4.1 shows, some layers' activations are binarizable (produce lower error when binarized) while other layers give high binarization errors. In this chapter, we will explore the second question: Which layers are more important than other layers in a network?

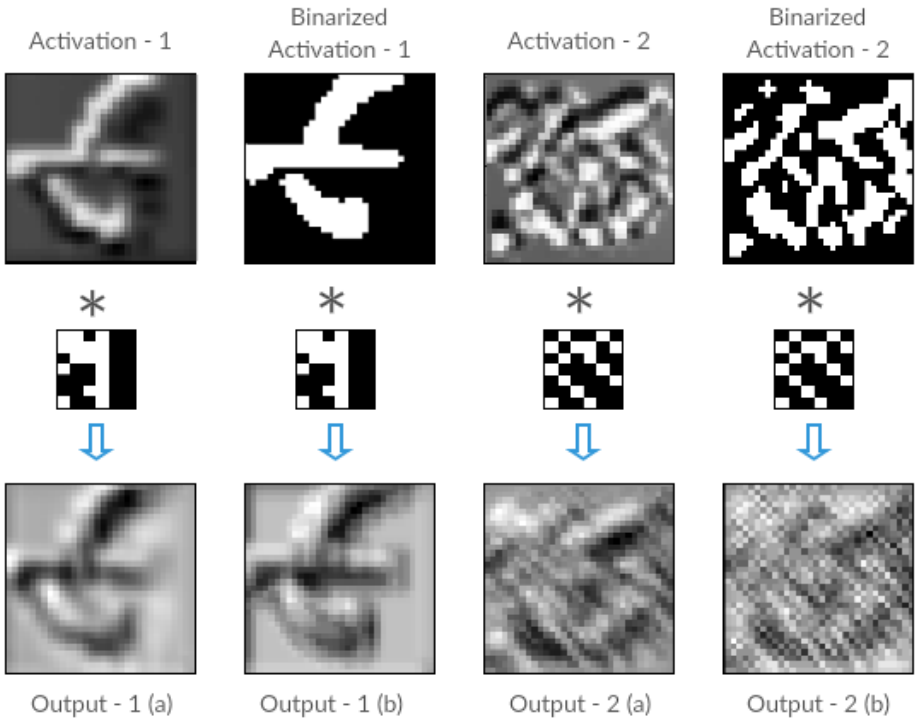


Figure 4.1: Convolution of binary and non-binary activations of two different layers. Note that the error introduced due to binarization is minimal in the first pair compared to the second. Hence, efficiently deciding *which* layers to binarize could contribute significantly to the overall accuracy of the network and not damage the speed-ups.

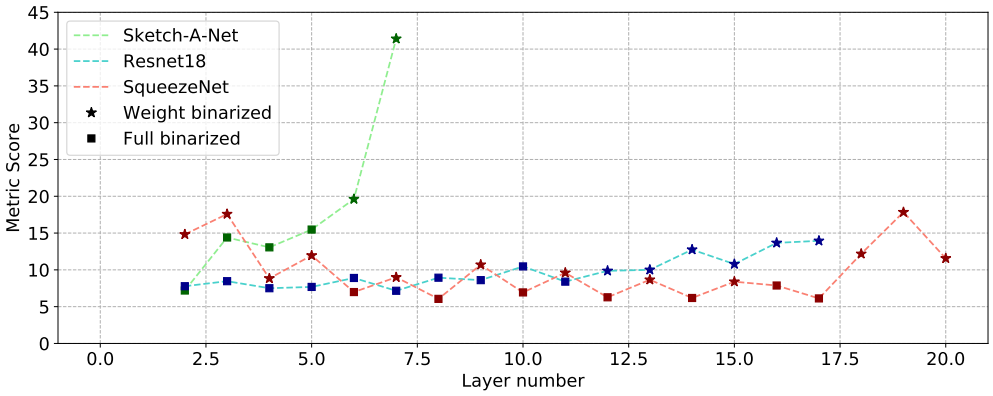


Figure 4.2: Binarization-error metric across layers for Sketch-A-Net, ResNet-18, and SqueezeNet. Stars indicate that the layer was replaced with a WeightBinConv layer, while squares indicate the FullBinConv layer was retained in the FBin model. We see that the algorithm selects the last layers in the case of Sketch-A-Net and ResNet, while in the case of SqueezeNet, it selects the first four, last three and some alternate intermediate layers to be replaced by WeightBinConv layers, retaining the rest as FullBinConv layers.

4.1 Hybrid Binarization

We define certain conventions to be used throughout the paper. We define a WBin CNN to be a CNN having the weights of convolutional layers binarized (referred to as WeightBinConv layers), FBin CNN to be a CNN having both inputs and weights of convolutional layers binarized (referred to as FullBinConv layers) and FPrec CNN to be the original full-precision network having both weights and inputs of convolutional layers in full-precision (referred to as Conv layers). We compare the FBin and WBin networks with FPrec networks at specific layers.

Table 4.2 and Table 4.3 in the Experiments section show test accuracies for WBin, FBin and FPrec networks of different models. Observe that there is very little loss in accuracy from FPrec to WBin networks with significant memory compression and fewer FLOPs. However, as we go from WBin to FBin networks, there is a significant drop in accuracy along with the trade-off of significantly lower FLOPs in FBin over WBin networks. Hence, we focus on improving the accuracies of FBin networks along with preserving the lower FLOPs as far as possible by investigating which activations to binarize.

4.1.1 Error Metric: Optimizing Speed & Accuracy

Full-precision inputs $\mathbf{I} \in \mathbb{R}^n$, are approximated by binary matrix $\mathbf{I_B} \in \{-1, +1\}^n$. The optimal binary representation $\mathbf{I_B}$ is calculated by

$$\mathbf{I_B}^* = argmin(\| \mathbf{I} - \mathbf{I_B} \|^2) \tag{4.1}$$

XNOR-Net [60] minimized the error function:

$$\mathbf{E} = \frac{\| \mathbf{I} - \mathbf{I_B} \|^2}{n} \tag{4.2}$$

In order to do that, they maximized $\mathbf{I}^\top \mathbf{I_B}$ and proposed the binary activation $\mathbf{I_B}$ to be

$$\mathbf{I_B}^* = \underset{I_B}{argmax}(\mathbf{I}^\top \mathbf{I_B}), \mathbf{I_B} \in \{-1, +1\}^n, \mathbf{I} \in \mathbb{R}^n \tag{4.3}$$

, obtaining the optimal $\mathbf{I_B}^*$ can be shown to be $sgn(\mathbf{I})$.

We need to investigate *where* to replace FullBinConv with WeightBinConv layers. In order to optimize for accuracy, we need to measure the efficacy of the binary approximation for inputs to any given layer. A good metric of this is the average error function calculated over a subset of training images $\mathbf{E}$ (defined in Eq. 2) used to calculate the optimal I_B itself, which is explicitly being minimized in the process. Hence, we use that error function to capture the binarization error.

Similarly to optimize speed, we need to convert layers with low number of FLOPs to WeightBinConv and layers having high number of FLOPs should be kept in FullBinConv. Since we need to jointly optimize both, we propose a metric that tries to achieve a good tradeoff between the two quantities. A simple but effective metric is the linear combination

$$\mathbf{M} = \mathbf{E} + \gamma \cdot \frac{1}{\mathbf{NF}} \tag{4.4}$$

where γ is the tradeoff ratio, $\mathbf{NF}$ is the number of flops in the layer and $\mathbf{E}$ is the binarization error per neuron. The trade-off ratio γ is a hyperparameter which ensures that both the terms are of comparable magnitude. Figure 4.2, captures the layer-wise variation of the error metric across multiple models.

4.1.2 Partitioning Algorithm

We aim to partition the layers of a network into two parts, one set of layers to keep FullBinConv and the other set which are replaced with WeightBinConv layers. A naive but intuitive partitioning algorithm would be to sort the list of metric errors M and replace FullBinConv layers which have highest error values M_i one-by-one with WeightBinConv layers, train new hybrid models and stop when the accuracies in the retrained models stop improving i.e when the maxima in accuracy v/s flops tradeoff is reached. However, we need a partitioning algorithm which gives informed guesses on where are the effective places to partition the set. This would avoid the long retraining times and large resources required to try every possible option for a hybrid model. We propose a layer selection algorithm that gives informed partitions from a trained FBin model, helping us to determine which layers are to be converted to WeightBinConv and which layers are to be converted to FullBinConv without having to train all possible hybrid models from scratch.

Algorithm 3 Partition Algorithm

Marks layers for binarization and creates a hybrid network.

Inputs ⇒Layer-wise Binarization Errors

`Initialization`

P = Total convolutional layers

R = Hybridization Ratio

ToConvert = List()

`Mark binary layers`

for N = 2 to P **do**

    Compute KMeans with N means

    K = Number of layers in highest-error cluster

    if $K/P \leq R$ **then**

        for Q in high-error clusters **do**

            ToConvert.add(Q) ▷ Add layer Q

        Break

`Create Hybrid Network`

HybridNet = ()

HybridNet.Add(Conv)

for N = 2 to P **do**

    if N in ToConvert **then**

        HybridNet.Add(WeightBinConv)

    else

        HybridNet.Add(FullBinConv)

Output ⇒ HybridNet

Our algorithm starts by taking a trained FBin model. We pass in a subset of the training images and calculate the average error metric for all layers over them. Then we perform K-Means Clustering on the metric values with each point being the metric error of layers as shown in Figure 4.2. We perform the K-Means Clustering for different values of the number of clusters. We find a suitable number of clusters such that the ratio of layers in the highest-error cluster (K) to the total number of convolutional layers (P) is less than a hyperparameter, which we define as the Hybridization Ratio R. Layers with terms falling in the highest mean cluster are converted to WeightBinConv, while the ones in all other clusters are left as FullBinConv. A flow of the algorithm is illustrated in Figure 4.3 and is explained step-by-step in Algorithm 3. We show metric scores of various layers for different networks in Figure 4.2 and indicate which layers are replaced with WeightBinConv/FullBinConv layers. This algorithm guides in forming the architecture of the hybrid model, which is then trained from scratch obtaining the accuracies given in the tables presented in the Experiment section. Note that this algorithm does not

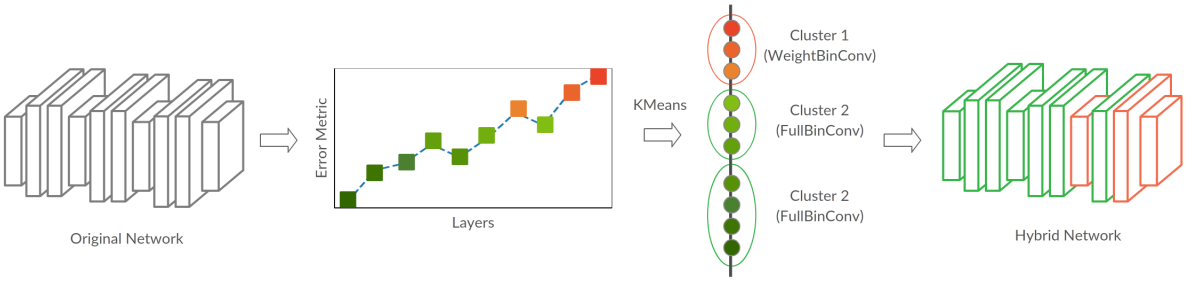


Figure 4.3: The Procedure: Error metrics from binarization of inputs to the network layers are partitioned into clusters using K-means. The highest error cluster indicates the inputs that are not binarized to generate the hybrid version.

change the configuration of the model; it only converts certain layers to their binarized versions.

To give an intuition of what the Hybridization ratio R means, a low R would indicate we need the number of WeightBinConv layers to be low, ensuring a high asymmetry between errors in WeightBinConv and FullBinConv layers, prioritizing saving computational cost. Conversely, a higher R would prioritize accuracy over computational cost. R was set to be 0.4 for AlexNet and ResNet-18, and 0.6 for Squeezenet. Variation with different values of R is further discussed in the experiments section.

4.1.3 Impact on Speed and Energy Use

Computational Speedups: Convolutional operations are computationally expensive. For each convolution operation between an image $\mathbf{I} \in \mathbb{R}^{c_{in} \times h_I \times w_I}$ and weight $\mathbf{W} \in \mathbb{R}^{c_{out} \times h \times w}$, the number of MAC operations required N are $\approx C_{in}C_{out}N_W N_I$ where $N_W = wh$ and $N_I = w_I h_I$. According to benchmarks done in XNOR-Net, the current speedup obtained in these operations is 58x after including the overhead induced by computing α. Accordingly, in later sections, we take one FLOP through a layer as equivalent to 58 binary operations when weights and inputs are binarized.

Exploiting filter repetitions: The number of unique convolutional binary filters is bounded by the size of the filter [14]. As most of our intermediate convolutional layers have 3×3 filters which only have 2^9 unique filters, we find that the percentage of unique filters decreases as we go deeper into the network. We can exploit this fact to simply prune filters and use that in calculating speedups for binary networks. More details regarding how the speedup was computed is included in the supplementary material.

4.2 Experiments and Results

We report and compare accuracies, speedups and compression between the FPrec model, different kinds of binarization models (WBin and FBin), and their generated hybrid versions of the same. We also

present a detailed comparison of our method with several different compression techniques applied on AlexNet [42], ResNet-18 [25], Sketch-A-Net [18] and SqueezeNet [35].

We empirically demonstrate the effectiveness of hybrid binarization on several benchmark image and sketch datasets. We show that our approach is robust and can generalize to different types of CNN architectures across domains.

4.2.1 Datasets and Models

Binary Networks have achieved accuracies comparable to full-precision networks on limited domain/simplified datasets like CIFAR-10, MNIST, SVHN, but show drastic accuracy losses on larger-scale datasets. To compare with state-of-the-art vision, we evaluate our method on ImageNet [16]. To show the robustness of our approach, we test it on sketch datasets, where models fine-tuned with ImageNet are demonstrably not suitable as shown in [80]. Binary networks might be better suited for sketch data due to its binary nature and sparsity of information in the data.

ImageNet: The benchmark dataset for evaluating image recognition tasks, with over a million training images and 50,000 validation images. We report the single-center-crop validation errors of the final models.

TU-Berlin: The TU-Berlin [18] sketch dataset is the most popular large-scale free-hand sketch dataset containing sketches of 250 categories, with a human sketch-recognition accuracy of 73.1% on average.

Sketchy: It is a recent large-scale free-hand sketch dataset containing 75,471 hand-drawn sketches from across 125 categories. This dataset was primarily used to cross-validate results obtained on the TU-Berlin dataset and ensure that our approach is robust to the variation in collection of data.

We use the standard splits with commonly used hyper-parameters to train our models. Each FullBinConv block was structured as in XNOR-Net (Batchnorm-Activ-Conv-ReLU). Each WeightBinConv and Conv block has the standard convolutional block structure (Conv-Batchnorm-ReLU). Weights of all layers except the first were binarized throughout our experiments unless specified otherwise. Note that FLOPs are stated in millions in all diagrams and sections. All networks are trained from scratch independently. The architecture of the hybrid network once designed does not change during training. Additional details about the datasets, model selection and layer-wise description of each of the hybrid models along with experimental details can be found in the supplementary material.

Technique	Acc-Top1	Acc-Top5	W/I	Mem	FLOPs
AlexNet					
BNN	39.5%	63.6%	1/1	32x	121 (1x)
XNOR	43.3%	68.4%	1/1	10.4x	**121 (1x)**
Hybrid-1	48.6%	72.1%	1/1	10.4x	174 (1.4x)
Hybrid-2	**48.2%**	**71.9%**	1/1	**31.6x**	174 (1.4x)
HTCBN	46.6%	71.1%	1/2	31.6x	780 (6.4x)
DoReFa-Net	47.7%	-	1/2	10.4x	780 (6.4x)
Res-Net 18					
BNN	42.1%	67.1%	1/1	32x	134 (1x)
XNOR	51.2%	73.2%	1/1	13.4x	**134 (1x)**
Hybrid-1	54.9%	77.9%	1/1	13.4x	359 (2.7x)
Hybrid-2	**54.8%**	**77.7%**	1/1	**31.2x**	359 (2.7x)
HTCBN	53.6%	-	1/2	31.2x	1030 (7.7x)

Table 4.1: A detailed comparison of accuracy, memory use, FLOPs with popular benchmark compression techniques on ImageNet. Our hybrid models outperform other 1-bit activation models and perform on par with 2-bit models while having a significantly higher speedup. Hybrid-2 models have the last layer binarized.

4.2.2 Results

We compare FBin, WBin, Hybrid and FPrec recognition accuracies across models on ImageNet, TU-Berlin and Sketchy datasets. Note that higher accuracies are an improvement, hence stated in green in the table, while higher FLOPs mean more computational expense, hence are stated in red. W/I indicates the number of bits used for weights and inputs to the layer respectively. Note that in the table, the compression obtained is only due to the weight binarization, while the decrease in effective FLOPs are due to activation binarization.

On the ImageNet dataset in Table 4.2, hybrid versions of AlexNet and ResNet-18 models outperform their FBin counterparts in top-1 accuracy by 4.1% and 3.6% respectively, and around 20x compression for both. We also compare with the results of other compression techniques in Table 4.1.

On the TU-Berlin and Sketchy datasets in Table 4.3, we find that Sketch-A-Net and ResNet-18 have significantly higher accuracies in the hybrid models compared to their FBin counterparts, a 13.5% gain for Sketch-A-Net and 5.0% for ResNet-18.

These hybrid models also achieve over 29x compression over FPrec models and with a reasonable increase in the number of FLOPs - a mere 7M increase in Sketch-A-Net and a decent 225M increase in ResNet-18. We also compare them with state-of-the-art sketch classification models in Table 4.4. Our hybrid Sketch-A-Net and ResNet-18 models achieve similar accuracies to state-of-the-art, while also

Model	Method	Accuracy		Mem	FLOPs
		Top-1	Top-5		
AlexNet	FPrec	57.1%	80.2%	1x	1135 (9.4x)
	WBin (BWN)	56.8%	79.4%	10.4x	780 (6.4x)
	FBin (XNOR)	43.3%	68.4%	10.4x	**121 (1x)**
	Hybrid-1	48.6%	72.1%	10.4x	174 (1.4x)
	Hybrid-2	**48.2%**	**71.9%**	**31.6x**	174 (1.4x)
Increase	Hybrid vs FBin	+4.9%	+3.5%	+21.2x	+53 (+0.4x)
ResNet-18	FPrec	69.3%	89.2%	1x	1814 (13.5x)
	WBin (BWN)	60.8%	83.0%	13.4x	1030 (7.7x)
	FBin (XNOR)	51.2%	73.2%	13.4x	**134 (1x)**
	Hybrid-1	54.9%	77.9%	13.4x	359 (2.7x)
	Hybrid-2	**54.8%**	**77.7%**	**31.2x**	359 (2.7x)
Increase	Hybrid vs FBin	+3.6%	+4.5%	+17.8x	+225 (+1.7x)

Table 4.2: Our hybrid models compared to FBin, WBin and NoBin models on Imagenet in terms of accuracy, memory and computations expense.

highly compressing the models upto 233x compared to the AlexNet FPrec model.

Thus, we find that our hybrid binarization technique finds a balance between sacrificing accuracy and gaining speedups and compression for various models on various datasets.

4.2.3 Algorithmic Insights

We gained some insights into where to binarize from our investigation. We provide them as a set of practical guidelines to enable rapid prototyping of hybrid models, which gives meaningful insights into which layers were partitioned.

Convert layers towards the end to WeightBinConv: It is observed that later layers typically have high error rates, more filter repetitions, and lower computational cost. Hence, the algorithm tends to start converting models to Hybrid from the last layers.

Convert the smaller of the layer placed parallely to WeightBinConv: It is a good idea to convert the smaller of the parallely placed layers in the architecture like Residual layers in the ResNet architecture to WeightBinConv, since converting them to WeightBinConv would not damage the computational speedup obtained by the parallel FullBinConv layers.

Pick a low Hybridization Ratio: Try to pick low values of the Hybridization Ratio R, ensuring a low proportion of number of layers the highest-error cluster.

Model	Method	Accuracy		Mem	FLOPs
		TU-Berlin	Sketchy		
Sketch-A-Net	FPrec	72.9%	85.9%	1x	608 (7.8x)
	WBin (BWN)	73%	85.6%	29.2x	406 (5.2x)
	FBin (XNOR)	59.6%	68.6%	19.7x	**78 (1x)**
	Hybrid	**73.1%**	**83.6%**	**29.2x**	**85 (1.1x)**
Increase	Hybrid vs FBin	+13.5%	+15.0%	+9.5x	+7 (+0.1x)
ResNet-18	FPrec	74.1%	88.7%	1x	1814 (13.5x)
	WBin (BWN)	73.4%	89.3%	31.2x	1030 (7.7x)
	FBin (XNOR)	68.8%	82.8%	31.2x	**134 (1x)**
	Hybrid	**73.8%**	**87.9%**	**31.2x**	359 (2.7x)
Increase	Hybrid vs FBin	+5.0%	+5.1%	-	+225 (+1.7x)

Table 4.3: Our hybrid models compared to FBin, WBin and full prec models on TU-Berlin and Sketchy datasets in terms of accuracy, memory and speed tradeoff.

Model	Acc	Mem	FLOPs
AlexNet-SVM	67.1%	1x	1135 (13.4x)
AlexNet-Sketch	68.6%	1x	1135 (13.4x)
Sketch-A-Net SC	72.2%	8x	608 (7.2x)
Sketch-A-Net-Hybrid	**73.1%**	**233x**	**85 (1x)**
ResNet18-Hybrid	**73.8%**	-	**359**
Humans	73.1%	-	-
Sketch-A-Net-2 [1] [79]	**77.0%**	8x	608 (7.2x)

Table 4.4: A comparison between state-of-the-art single model accuracies of recognition systems on the TU-Berlin dataset.

Relax the Hybridization Ratio for compact models: Having a higher Hybridization Ratio for compact models which inherently have fewer flops leaves more layer inputs un-binarized and retains accuracy.

4.2.4 Why are layer-wise errors independent?

Can binarization noise introduced in a layer propagate further into the network and influence other layers? Courbariaux *et al*. [14] provide some insights for the same. Let $\mathbf{W}$ be the weight and $\mathbf{I}$ be the input to the convolutional layer. The output of the convolution between the binary weights and inputs can be represented by

$$\mathbf{O_B} = \alpha \cdot (sgn(\mathbf{W}^\intercal) \odot sgn(\mathbf{I})) \tag{4.5}$$

The desired output $\mathbf{O}$ is modelled by $\mathbf{O_B}$ along with the binarization noise $\mathbf{N}$ introduced due to the function $sgn(.)$.

$$\mathbf{O} = \mathbf{W} * \mathbf{I} = \sum_i \mathbf{O_{B}}_i + \mathbf{N_i} \tag{4.6}$$

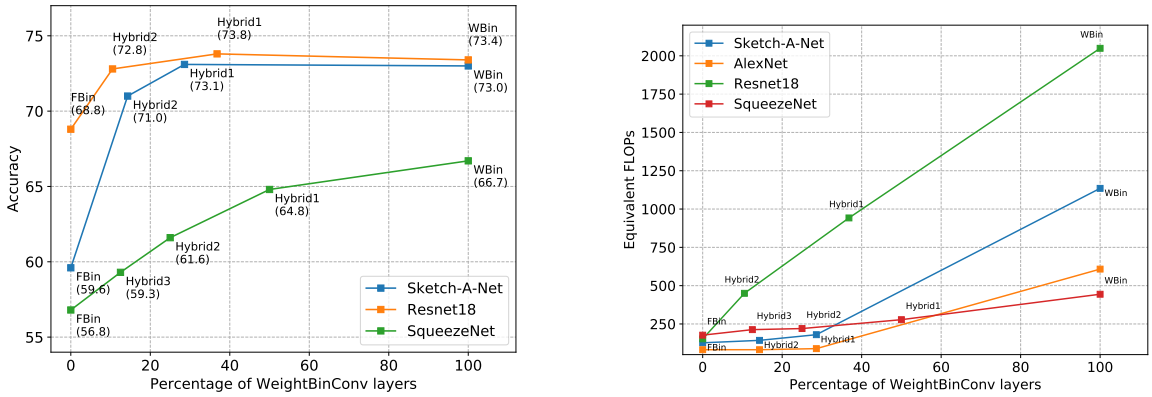


Figure 4.4: Trade-off between WeightBinConv layers and accuracy on the TU-Berlin dataset is shown in the left figure, while the trade-off between weight binarized layers and speedup is shown in the right figure. Early on, we observe that a small increase in the percentage of WeightBinConv layers leads to a large increase in accuracy and a marginal decrease in speed. We achieve accuracies comparable to the WBin model with much fewer WeightBinConv layers.

When the layer is wide, we expect the deterministic term $\mathbf{O_B}$ to dominate, because the noise term $\mathbf{N}$ is a summation over many independent binarizations from all the neurons in the layer. Thus, we argue that the binarization noise $\mathbf{N}$ should have minimal propagation and do little to influence the further inputs. Hence, it is a reasonable approximation to consider the error across each layer independently of the other layers.

4.2.5 Variation with the Hybridization Ratio (R)

To observe the trade-off between accuracy and speedup on different degrees of binarization, we chose different values of the Hybridization Ratio (R) to create multiple hybrid versions of the AlexNet, ResNet-18 and SqueezeNet models. Picking a larger R would result in a higher number of WeightBinConv layers. We compare these hybrid networks to their corresponding FBin and WBin versions.

In Figure 4.4, we show model accuracies of AlexNet, ResNet-18 and SqueezeNet on the ImageNet dataset plotted against the number of WeightBinConv layers, starting from only FBin versions on the left, to only WBin versions on the right. We observe that in the case of AlexNet and ResNet-18, which are large models, we recover WBin accuracies quickly, at around the 35% mark (Roughly a third of the network containing WeightBinConv layers), with low computational trade-off. We also observe that on sketch data, hybrid models tend to perform significantly better and perform on par with their WBin counterparts.

Model	BinType	Last Bin?	Acc	Mem
Sketch-A-Net	FBin (XNOR)	No	59.6%	19.7x
		Yes	48.3%	**29.2x**
Sketch-A-Net	Hybrid	No	73.1%	19.7x
		Yes	72.0%	**29.2x**
Resnet-18	FBin (XNOR)	No	69.9%	13.4x
		Yes	68.8%	**31.2x**
Resnet-18	Hybrid	No	73.9%	13.4x
		Yes	73.8%	**31.2x**

Table 4.5: Effects of last layer weight-binarization on TU-Berlin dataset, for Sketch-A-Net and ResNet-1. Observe that our hybrid models do not face drastic accuracy drop when the last layer is weight-binarized.

We also notice that the smaller a model, the more trade-off must be made to achieve WBin accuracy, i.e a larger Hybridization Ratio must be used. AlexNet, the largest model crosses WBin accuracy at around 32%, while ResNet-18, being smaller, saturates at around 40%. SqueezeNet, a much more compact model, reaches its WBin accuracy at 60%.

4.2.6 Optimizing Memory

We measured accuracies for FBin and Hybrid variants of Sketch-A-Net and ResNet-18 models on TU-Berlin and Sketchy Datasets with weights of the last layer binarized as well as non-binarized and the results are presented in Table 4.5. For AlexNet-styled architectures (Sketch-A-Net), we observe a drastic drop in accuracies (From 59.1% to 48.3%) on binarizing the last layer, similar to observations made in previous binarization works [72, 85].

Many efforts were made to quantize the last layer and avoid this drop. DoReFaNet and XNOR-Net did not binarize the last layer choosing to incur a degradation in model compression instead while [72] proposed an additional scale layer to mitigate this effect. However, our hybrid versions are able to achieve similar accuracies (a 1% drop for hybrid Sketch-A-Net and no drop for ResNet-18 or AlexNet) since the last layer is weight binarized instead. Hence, our method preserves the overall speedup even though we only weight-binarize the last layer, owing to the comparatively smaller number of computations that occur in this layer.

Note that the first layer is always a full-precision Conv layer. The reasons behind this are the insights obtained from [1]. They state that the first layer of the network functions are fundamentally different than the computations being done in the rest of the network because the high variance principal components are not randomly oriented relative to the binarization. Also, since it contains fewer parameters and low computational cost, it does not affect our experiments.

Model	Method	Accuracy		Mem	FLOPs
		TU-Berlin	Sketchy		
Sketch-A-Net	FPrec	72.9%	85.9%	1x	1135 (12.3x)
Squeezenet	FPrec	71.2%	86.5%	8x	610 (6.6x)
Squeezenet	WBin	66.7%	81.1%	23.7x	412 (4.5x)
Squeezenet	FBin	56.8%	66.0%	23.7x	**92 (1x)**
Squeezenet	Hybrid	**64.8%**	**79.6%**	**23.7x**	164 (1.8x)
Improvement	Hybrid vs FBin	+8.0%	+13.6%	-	+72 (+0.8x)

Table 4.6: Our performance on SqueezeNet, an explicitly compressed model architecture. Although SqueezeNet is an inherently compressed model, our method still achieves further compression on it.

4.2.7 Compressing Compact Models

Whether compact models can be compressed further, or *need* all of the representational power afforded through dense floating-point values is an open question asked originally by [35].

We show that our hybrid-binarization technique can work in tandem with other compression techniques, which do not involve quantization of weights/activations and that hybrid binarization is possible even on compact models. We apply hybrid binarization to SqueezeNet [35] a recent model that employed various architectural design strategies to achieve compactness. SqueezeNet achieves an 8x compression on the compact architecture of Sketch-A-Net. On applying hybrid binarization we achieve a further 32x compression, an overall 256x compression with merely 6% decrease in accuracy. This is due to the high rate of compression inherent and further compression is difficult due to the small number of parameters. After showing that efficacy of hybrid binarization in the previous section, we show that hybrid binarization can work in combination with other compression techniques here.

Results for SqueezeNet are shown in Table 4.6 for the TU-Berlin and Sketchy datasets, and we see that accuracy is only slightly lower compared to the hybridized versions of ResNet-18 and Sketch-A-Net on the same. Hybrid SqueezeNet achieves a total compression of 256x. Similarly, this technique can be combined with many techniques such as HWGQ-Net [6] which proposes an alternative layer to ReLU and repeated binarization as illustrated in [72] among others. Since our primary goal is to investigate the viability of hybrid binarization, these investigations- albeit interesting, are out of the scope of our current work.

4.3 Summary

We proposed a novel algorithm for selective binarization of CNNs, which strikes a balance between performance, memory-savings and accuracy. The accuracies of our hybrid models were on par with their corresponding full-precision networks on TU-Berlin and Sketchy datasets, while providing the benefits

of network binarization in terms of speedups, compression and energy efficiency. We successfully weight-binarized the last layers without significant accuracy drops, a problem faced by previous works in this area. We also showed that we can successfully combine the advantages of our approach with other architectural compression strategies, to obtain highly efficient models with negligible accuracy penalties.

Chapter 5

Deep Expander Networks

How do we prune before training? In this chapter, we explore an principled way of pruning on a layer-level before training by imposing some necessary constraints on the architecture.

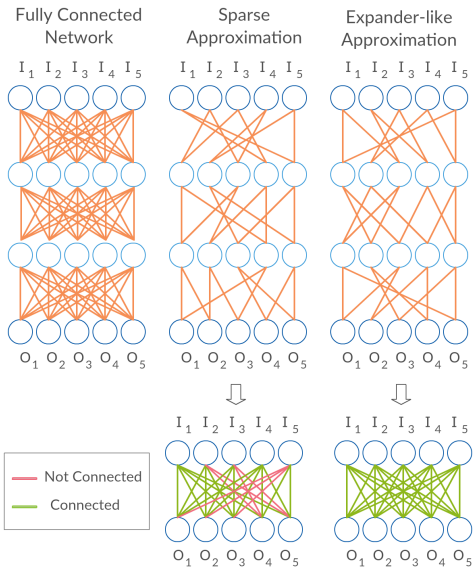


Figure 5.1: Popular sparse approximations are agnostic to the global information flow in a network, possibly creating disconnected components. In contrast, expander graph-based models produce sparse yet highly connected networks.

5.1 Approach

Recent breakthroughs in CNN architectures like ResNet [25] and DenseNet-BC [33] are ideas based on increasing connectivity, which resulted in better performance trade-offs. These works suggest that connectivity is an important property for improving the performance of deep CNNs. In that vein, we investigate ways of preserving connectivity between neurons while significantly sparsifying the connections

between them, shown in 5.1. Such networks are expected to preserve accuracy (due to connectivity) while being runtime efficient (due to the sparsity). We empirically demonstrate this in the later sections.

5.1.1 Graphs and Deep CNNs

We model the connections between neurons as graphs. This enables us to leverage well-studied concepts from Graph Theory like Expander Graphs. Now, we proceed to formally describe the connection between graphs and Deep CNNs.

Linear Layer defined by a Graph: Given a bipartite graph G with vertices U, V, the Linear layer defined by G, is a layer with $|U|$ input neurons, $|V|$ output neurons and each output neuron $v \in V$ is only connected to the neighbors given by G. Let the graph G be sparse, having only M edges. Then this layer has only M parameters as compared to $|V| \times |U|$, which is the size of typical linear layers.
Convolutional Layer defined by a Graph: Let a Convolutional layer be defined as a bipartite graph G with vertices U, V and a window size of $c \times c$. This layer takes a 3D input with $|U|$ channels and produces a 3D output with $|V|$ channels. The output channel corresponding to a vertex $v \in V$ is computed only using the input channels corresponding to the neighbors of v. Let G be sparse, having only M edges. Hence the kernel of this convolutional layer has $M \times c \times c$ parameters as compared to $|V| \times |U| \times c \times c$, which is the number of parameters in a vanilla CNN layer.

5.1.2 Sparse Random Graphs

We want to constrain our convolutional layers to form a sparse graph G. Without any prior knowledge of the data distribution, we take inspiration from randomized algorithms and propose choosing the neighbours of every output neuron/channel uniformly and independently at random from the set of all its input channels. It is known that a graph G obtained in this way belongs to a well-studied category of graphs called Expander Graphs, known to be sparse but well connected.

Expander Graph: A bipartite expander with degree D and spectral gap γ, is a bipartite graph $G = (U, V, E)$ (E is the set of edges, $E \subseteq U \times V$) in which:

1.) Sparsity: Every vertex in V has only D neighbors in U. We will be using constructions with $D << |U|$. Hence the number of edges is only $D \times |V|$ as compared to $|U| \times |V|$ in a dense graph.

2.) Spectral Gap: The eigenvalue with the second largest absolute value λ of the adjacency matrix is bounded away from D (the largest eigenvalue). Formally $1 - \lambda/D \geq \gamma$.

Random expanders: A random bipartite expander of degree D on the two vertex sets U, V, is a graph in which for every vertex $v \in V$, the D neighbors are chosen independently and uniformly from U.

It is a well-known result in graph theory that such graphs have a large spectral gap ([73]). Similar to random expanders, there exist several explicit expander constructions. More details about explicit expanders can be found in the supplementary section. We now proceed to give constructions of deep networks that have connections defined by an expander graph.

Expander Linear Layer (X-Linear): The Expander Linear (X-Linear) layer is a layer defined by a random bipartite expander G with degree D. The expander graphs that we use have values of $D << |U|$, while having an expansion factor of $K \approx D$, which ensures that the layer still has good expressive power.

Expander Convolutional Layer (X-Conv): The Expander Convolutional (X-Conv) layer is a convolutional layer defined by a random bipartite expander graph G with degree D, where $D << |U|$.

Deep Expander Networks (X-Nets): Given expander graphs

$$G_1 = (V_0, V_1, E_1), G_2 = (V_1, V_2, E_2), \cdots, G_t = (V_{t-1}, V_t, E_t)$$

, we define the Deep Expander Convolutional Network (Convolutional X-Net or simply X-Net) as a t layer deep network in which the convolutional layers are replaced by X-Conv layers and linear layers are replaced by X-Linear layers defined by the corresponding graphs.

5.1.3 Measures of Connectivity

In this subsection, we describe some connectivity properties of Expander graphs (see [73], for the proofs). These will be used to prove the properties of sensitivity and mixing of random walks in X-Nets.

Expansion: For every subset $S \subseteq V$ of size $\leq \alpha|V|$ ($\alpha \in (0, 1)$ depends on the construction), let $N(S)$ be the set of neighbors. Then $|N(S)| \geq K|S|$ for $K \approx D$. That is, the neighbors of the vertices in S are almost distinct. It is known that random expanders have expansion factor $K \approx D$ (see Theorem 4.4 in [73]).

Small Diameter: The diameter of a graph is the length of the longest path among all shortest paths. If $G(U, V, E)$ is a D-regular expander with expansion factor $K > 1$ and diameter d, then $d \leq O(\log n)$. This bound on the diameter implies that for any pair of vertices, there is a path of length $O(\log n)$ in the graph.

Mixing of Random Walks: Random walks in the graph quickly converge to the uniform distribution over nodes of the graph. If we start from any vertex and keep moving to a random neighbor, in $O(\log n)$ steps the distribution will be close to uniform over the set of vertices.

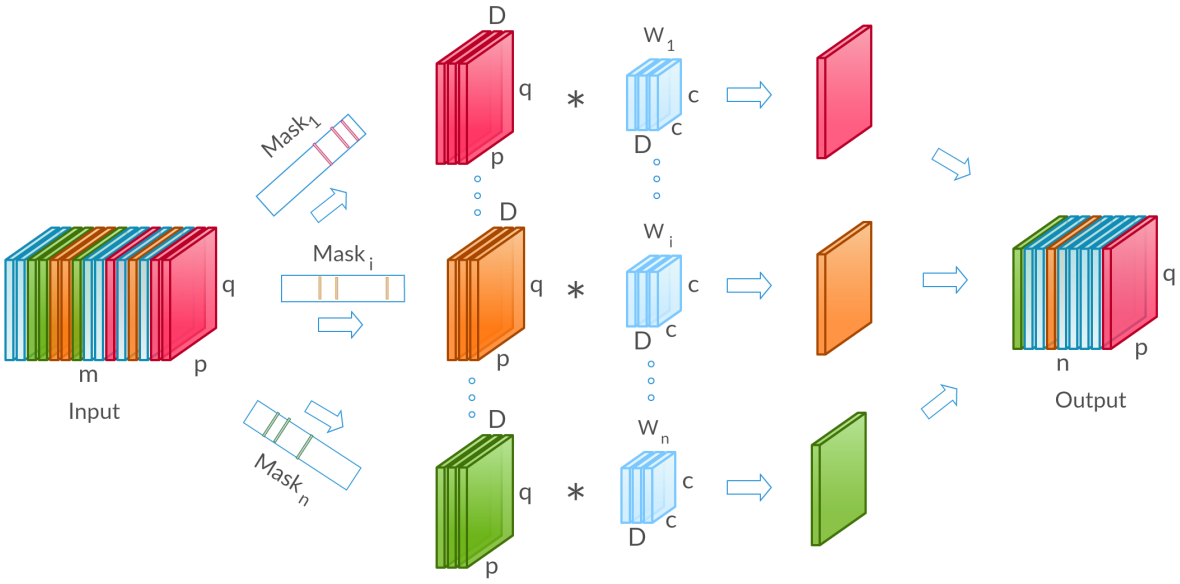


Figure 5.2: The proposed fast convolution algorithm for X-Conv layer. We represent all the non-zero filters in the weight matrix of the X-Conv layer as a compressed dense matrix of D channels. The algorithm starts by selecting D channels from input (with replacement) using a mask created while initializing the model. The output is computed by convolving these selected channels with the compressed weight matrices.

5.1.4 Sensitivity of X-Nets

X-Nets have multiple layers, each of which have connections derived from an expander graph. We can guarantee that the output nodes in such a network are sensitive to all the input nodes.

Theorem 5 (Sensitivity of X-Nets) *Let n be the number of input as well as output nodes in the network and $G_1, G_2, \cdots, G_t$ be D regular bipartite expander graphs with n nodes on both sides. Then every output neuron is sensitive to every input in a Deep X-Net defined by G_i's with depth $t = O(\log n)$.*

Proof: For every pair of input and output (u, v), we show that there is a path in the X-Net. The proof is essentially related to the the fact that expander graphs have diameter $O(\log n)$. A detailed proof can be found in the Appendix B.

Next, we show a much stronger connectivity property known as mixing for the X-Nets. The theorem essentially says that the number of edges between subsets of input and output nodes is proportional to the product of their sizes. This result implies that the connectivity properties are uniform and rich across all nodes as well as subsets of nodes of the same size. Simply put, all nodes tend to have equally rich representational power.

Algorithm 1: Fast Algorithm for Convolutions in X-Conv Layer

For every vertex $v \in \{1, \cdots, n\}$, let $N(v, i)$ denote the ith neighbor of v ($i \in \{1, \cdots, D\}$).
Let K_v be the $c \times c \times D \times 1$ sized kernel associated with the vth output channel.
Let $O_v[x, y]$ be the output value of the vth channel at the position x, y.
for v= 1 to n **do**
    $O_v[x, y] = K_v * \mathit{Mask}_{N(v,1),\cdots N(v,D)}(I)[x, y]$.

Theorem 6 (Mixing in X-Nets) *Let n be the number of input as well as output nodes in the network and G be D regular bipartite expander graph with n nodes on both sides. Let S, T be subsets of input and output nodes in the X-Net layer defined by G. The number of edges between S and T is $\approx D|S||T|/n$.*

Proof: A detailed proof is provided in the Appendix B.

5.2 Efficient Algorithms

In this section, we present efficient algorithms of X-Conv layers. Our algorithms achieve speedups and save memory in the training as well as the inference phase. This enables one to experiment with significantly wider and deeper networks given memory and runtime constraints. We exploit the structured sparsity of expander graphs to design fast algorithms. We propose two methods of training X-Nets, both requiring substantially less memory and computational cost than their vanilla counterparts:
1) Using Sparse Representations
2) Expander-Specific Fast Algorithms.

5.2.1 Using Sparse Representation

The adjacency matrices of expander graphs are highly sparse for $D << n$. Hence, we can initialize a sparse matrix with non-zero entries corresponding to the edges of the expander graphs. Unlike most pruning techniques, the sparse connections are determined before training phase, and stay fixed. Dense-Sparse convolutions are easy to implement, and are supported by most deep learning libraries. CNN libraries like Cuda-convnet [40] support such random sparse convolution algorithms.

5.2.2 X-Net based Fast Dense Convolution

Next, we present fast algorithms that exploit the sparsity of expander graphs.

X-Conv: In an X-Conv layer, every output channel is only sensitive to out rom input channels. We propose to use a mask to select D channels of the input, and then convolve with a $c \times c \times D \times 1$ kernel, obtaining a single channel per filter in the output. The mask is obtained by choosing D samples

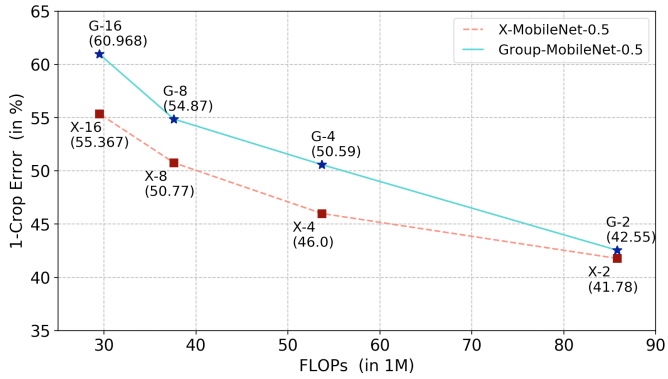


Figure 5.3: Comparison between Grouped convolutions and X-Conv using MobileNet architecture trained on ImageNet. X-d or G-d represents the 1x1 conv layers are compressed by d times using X-Conv or Groups. We observe X-MobileNets beat Group-MobileNet by 4% in accuracy on increasing sparsity.

uniformly (without replacement) from the set $\{1, \cdots N\}$, where N is the number of input channels. The mask value is 1 for each of the selected D channels and 0 for others (see Algorithm **??**). This is illustrated in Figure 5.2. There has been recent work about fast CUDA implementations called Block-Sparse GPU Kernels [19], which can implement this algorithm efficiently.

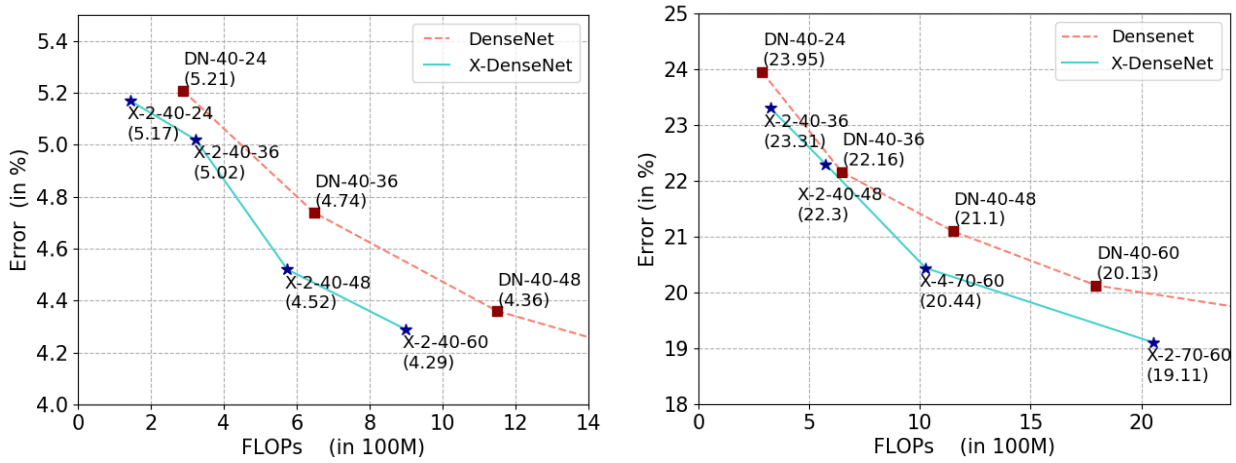
5.3 Experiments and Results

In this section, we benchmark and empirically demonstrate the effectiveness of X-Nets on a variety of CNN architectures. Our code is available at: `https://github.com/DrImpossible/Deep-Expander-Networks`.

5.3.1 Comparison with Grouped Convolution

First, we compare our Expander Convolutions (X-Conv) against Grouped Convolutions (G-Conv). We choose G-Conv as it is a popular approach, on which a lot of concurrent works [82] have developed their ideas. G-Conv networks have the same sparsity as X-Conv networks but lack only the connectivity property. This will test whether increasing connectivity increases accuracy, i.e does a graph without good connectivity properties provides worse accuracy? We choose MobileNet as the base model for this experiment, since it is the state-of-the-art in efficient CNN architectures.

We compare X-Conv against grouped convolutions using MobileNet-0.5 on the ImageNet classification task. We replace the 1×1 convolutional layers in MobileNet-0.5 with X-Conv layers forming X-MobileNet-0.5. Similarly, we replace them with G-Conv layers to form Group-MobileNet-0.5. Note that we perform this only in layers with most number of parameters (after the 8th layer as given in Table 1 of [30]). We present our results in Figure 5.3. The reference original MobileNet-0.5 has an error of 36.6% with a cost of 150M FLOPs. Additional implementation details are given in the supplementary



(a) CIFAR10 (b) CIFAR100

Figure 5.4: We show the error as a function of #FLOPs during test-time (below) for DenseNet-BC with X-DenseNet-BCs on CIFAR10 and CIFAR100 datasets. We observe X-DenseNet-BCs achieve better performance tradeoffs over DenseNet-BC models. For each datapoint, we mention the X-C-D-G notation (see Section 5.3.2) along with the accuracy.

material.

We can observe that X-MobileNets beat Group-MobileNets by over 4% in terms of accuracy when we increase sparsity. This also demonstrates that X-Conv can be used to further improve the efficiency of even the most efficient architectures like MobileNet.

5.3.2 Comparison with Efficient CNN Architectures

In this section, we test whether Expander Graphs can improve the performance trade-offs even in state-of-the-art architectures such as DenseNet-BCs [33] and ResNets [25] on the ImageNet [16] dataset. We additionally train DenseNet-BCs on CIFAR-10 and CIFAR-100 [41] datasets to demonstrate the robustness of our approach across datasets.

Our X-ResNet-C-D is a D layered ResNet that has every layer except the first and last replaced by an X-Conv layer that compresses connections between it and the previous layer by a factor of C. We compare across various models like ResNets-34,50,101. Similarly, our X-DenseNet-BC-C-D-G architecture has depth D, and growth rate G. We use DenseNet-BC-121-32,169-32,161-48,201-32 as base models. These networks have every layer except the first and last replaced by an X-Conv layer that compresses connections between it and the previous layer by a factor of C. More details are provided in the supplementary material.

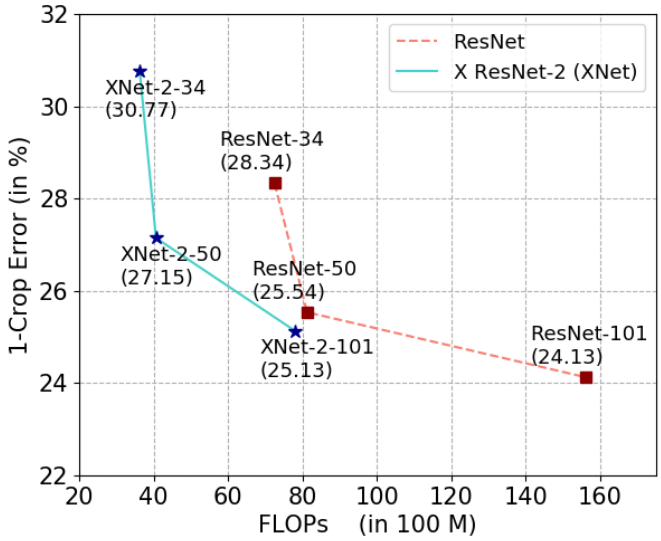


Figure 5.5: We show the error as a function of #FLOPs to compare between ResNet and X-ResNet on the ImageNet dataset. We observe X-ResNets achieve better performance tradeoffs over original ResNet models.

Model	**Accuracy**	**#FLOPs**
ResNet		**(in 100M)**
X-ResNet-2-34	69.23%	35
X-ResNet-2-50	**72.85%**	**40**
ResNet-34	71.66%	70
X-ResNet-2-101	**74.87%**	**80**
ResNet-50	74.46%	80
ResNet-101	75.87%	160
DenseNet-BC		
X-DenseNet-BC-2-121	70.5%	28
X-DenseNet-BC-2-169	71.7%	33
X-DenseNet-BC-2-201	72.5%	43
X-DenseNet-BC-2-161	**74.3%**	**55**
DenseNet-BC-121	73.3%	55
DenseNet-BC-169	74.8%	65
DenseNet-BC-201	75.6%	85
DenseNet-BC-161	76.3%	110

Table 5.1: Results obtained by ResNet and DenseNet-BC models on ImageNet dataset, ordered by #FLOPs. or each datapoint, we use the X-C-D-G notation (see Section 5.3.2) along with the accuracy.

5.3.3 Comparison with Pruning Techniques

We plot the performance tradeoff of X-ResNets against ResNets in Figure A.2 . We achieve significantly better performance tradeoffs compared to the original model. More specifically, we can reduce the #FLOPs in ResNets by half while incurring only 1-1.5% decrease in accuracy. Also, we can compare models with similar #FLOPs or accuracy with the help of Table B.4. We observe that X-ResNet-2-50 has 43% fewer FLOPs than ResNet-34, but achieves a 1% improvement in accuracy against it. Similarly, X-DenseNet-BC-2-161 has similar #FLOPs as DenseNet-BC-121, but achieves a 1% improvement in accuracy.

To further prove the robustness of our approach on DenseNet-BC, we test the same on CIFAR10 and CIFAR100, and plot the tradeoff curve in Figure 5.4. We observe that we can achieve upto 33% compression keeping accuracy constant on CIFAR-10 and CIFAR-100 datasets.

We compare our approach with methods which prune the weights during or after training. Our method can be thought of as constraining the weight matrices with a well studied sparse connectivity pattern even before the training starts. This results in fast training for the compact X-Conv models, while the trained pruning techniques face the following challenges:

1) Slow initial training due to full dense model.
2) Several additional phases of pruning and retraining.

Hence they achieve the compactness and runtime efficiency only in test time. Nevertheless we show similar sparsity can be achieved by our approach without explicitly pruning. We benchmark on VGG16 and AlexNet architectures since most previous results in the pruning literature have been reported on these architectures. In Table 5.2, we compare two X-VGG-16 models against existing pruning techniques. We achieve comparable accuracies to the previous state-of-the-art model with 50% fewer parameters and #FLOPs. Similarly, in Table 5.3 we compare X-AlexNet with trained pruning techniques on the Imagenet dataset. Despite having poor connectivity due to parameters being concentrated only in the last three fully connected layers, we achieve similar accuracy to AlexNet model using only 7.6M-9.7M parameters out of 61M, comparable to the state-of-the-art pruning techniques which have upto 3.4M-5.9M parameters. Additionally, it is possible to improve compression by applying pruning methods on our compact architectures, but pruning X-Nets is out of the scope of our current work.

5.3.4 Stability of Models

We give empirical evidence as well as a theoretical argument regarding the stability of our method. For the vanilla DNN training, the weights are randomly initialized, and randomized techniques like

Method	Accuracy	#Params
Li et al. [47]	93.4%	5.4M
Liu et al. [52]	93.8%	2.3M
X-VGG16-1	93.4%	1.65M (9x)
X-VGG16-2	93.0%	1.15M (13x)
VGG16-Orig	94.0%	15.0M

Table 5.2: Comparison with other methods on CIFAR-10 dataset using VGG16 as the base model. We significantly outperform popular compression techniques, achieving similar accuracies with upto 13x compression rate.

Method	Accuracy	#Params
Network Pruning		
Collins et al. [12]	55.1%	15.2M
Zhou et al. [84]	54.4%	14.1M
Han et al. [22]	57.2%	6.7M
Han et al. [22]	57.2%	6.7M
Srinivas et al. [68]	56.9%	5.9M
Guo et al. [21]	56.9%	3.4M
X-AlexNet-1	55.2%	7.6M
X-AlexNet-2	56.2%	9.7M
AlexNet-Orig	57.2%	61M

Table 5.3: Comparison with other methods on ImageNet-2012 using AlexNet as the base model. We are able to achieve comparable accuracies using only 9.7M parameters.

dropouts, augmentation are used. Hence there is some randomness present and is well accepted in DNN literature prior to our method. We repeat experiments on different datasets (Imagenet and CIFAR10) and architectures (VGG, DenseNet and MobileNet0.5) to empirically show that the accuracy of expander based models has variance similar to vanilla DNN training over multiple runs.

We repeated the experiments with independent sampling of random expanders on the VGG and DenseNet baselines on the CIFAR10 dataset. The results can be seen in Table B.3. It is noted that the accuracy values changes only by less than 0.3% across runs and the standard deviation of expander method is also comparable to the vanilla DNN training.

We also repeated experiments of our main result, which is the comparison with grouped convolutions on ImageNet dataset. We rerun the experiment with MobileNet0.5 feature extractor twice with Groups and the expander method. As can be seen from Table 5.5, the accuracy variations are comparable between the two models, and it is less than 1%.

A theoretical argument also concludes that choosing random graphs doesn't degrade stability. It is a well known result (See Theorem 4.4 in [73]) in random graph theory, that graphs chosen randomly are well

Model	Accuracy%	Max%	Min%
VGG	93.96±0.12	94.17	93.67
X-VGG-1	93.31±0.18	93.66	93.06
X-VGG-2	92.91±0.19	93.26	92.69
XDNetBC-40-24	94.41±0.19	94.63	94.18
XDNetBC-40-36	94.98±0.14	95.21	94.84
XDNetBC-40-48	95.49±0.15	95.65	95.28
XDNetBC-40-60	95.75±0.07	95.81	95.68

Table 5.4: The accuracies (mean ± stddev) of various models over 10 training runs on CIFAR-10 dataset.

MobileNet Variant	Mean Accuracy	Range (Max-Min)
Base	63.39%	0.11%
G2	57.45 %	0.06 %
X2	58.22 %	0.14 %
G4	49.41 %	0.55 %
X4	54.00 %	0.53 %
G8	45.13 %	0.03 %
X8	49.23 %	0.60 %
G16	39.03 %	0.64 %
X16	44.63 %	0.18 %

Table 5.5: The mean accuracy and range of variation over 2 runs of MobileNet0.5 variants on ImageNet dataset.

connected with overwhelmingly high probability (with only inverse exponentially small error, due to the Chernoff's Tail bounds) and satisfies the Expander properties. Hence the chance that for a specific run, the accuracy gets affected due to the selection of a particularly badly connected graph is insignificant.

5.3.5 Training Wider and Deeper networks

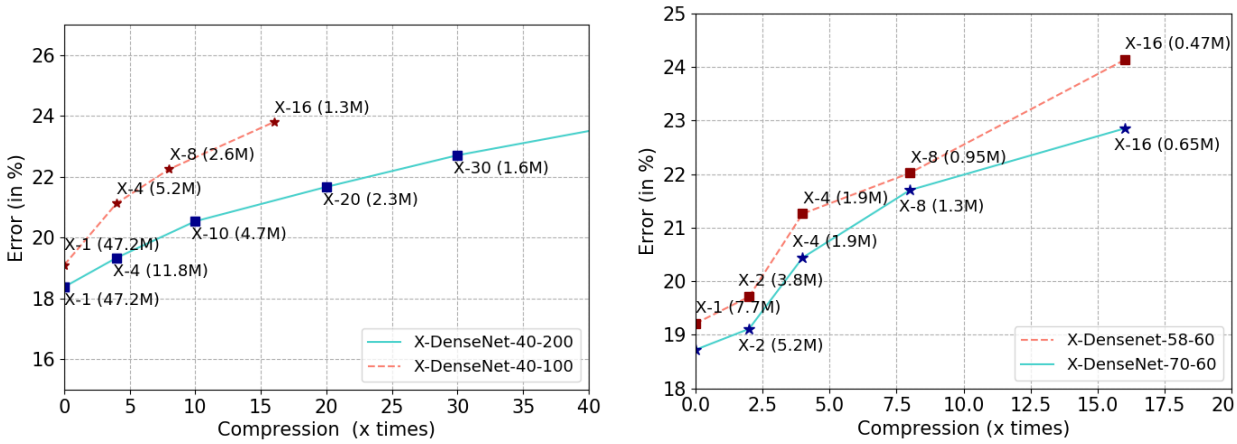
Since X-Nets involve constraining the weight matrices to sparse connectivity patterns before training, the fast algorithms can make it possible to utilize memory and runtime efficiently in training phase. This makes it possible to train significantly deeper and wider networks. Note the contrast with pruning techniques, where it is necessary to train the full, bulky model, inherently limiting the range of models that can be compressed.

Wide-DenseNets[1] offered a better accuracy-memory-time trade-off. We increase the width and depth of these networks to train significantly wider and deeper networks. The aim is to study whether leveraging the effectiveness of X-Nets in this fashion can lead to better accuracies.

We widen and deepen the DenseNet-BC-40-60 architecture, increasing the growth rate from 60 to 100 and 200 respectively and compare the effect of increasing width on these new models. Similarly, we increase the depth from 40 to 58 and 70 to obtain deeper networks. We benchmark these approaches using CIFAR-100 dataset and present the results in Figure 5.6.

We have two interesting observations. First, the deeper X-DenseNet-BC-70-60 significantly outperforms X-DenseNet-BC-58-60 and wider X-DenseNet-40-200 outperforms X-DenseNet-BC-40-100 with fewer parameters for a wide range of C values (Expander degree).

[1] https://github.com/liuzhuang13/DenseNet#wide-densenet-for-better-timeaccuracy-and-memoryaccuracy-tradeoff



(a) Effect of Width

(b) Effect of Depth

Figure 5.6: We show the performance tradeoff obtained on training significantly wider and deeper networks on CIFAR-100 dataset. Every datapoint is X-C specified along with the number of parameters, C being the compression factor. We show that training wider or deeper networks along with more compression using X-Nets achieve better accuracies with upto two-thirds of the total parameter and FLOPs on CIFAR-100 dataset.

The second interesting observation is the decreasing slope of the curves. This indicates that expander graph modeling seems to be effective on wider and deeper X-Nets i.e X-DenseNet-BC models suffer lesser penalty with increasing depth and width compression. This enables X-Nets to work at high compression rates of 30x, compressing DenseNet-BC-40-200 model from 19.9B FLOPs to 0.6B FLOPs with only 4.3% drop in accuracy. We hope this preliminary investigation holds significant value in alleviating the constraint of GPU memory and resources.

5.4 Summary

We proposed a new network layer architecture for deep networks using expander graphs that give strong theoretical guarantees on connectivity. The resulting architecture (X-Net) is shown to be highly efficient in terms of both computational requirements and model size. In addition to being compact and computationally efficient, the connectivity properties of the network allow us to achieve significant improvements over the state-of-the-art architectures in performance on a parameter or run-time budget. In short, we show that the use of principled approaches that sparsify a model while maintaining global information flows can help in developing efficient deep networks. To the best of our knowledge, this is the first attempt at using theoretical results from graph theory in modeling connectivity to improve deep network architectures.

Chapter 6

Conclusions

Convolutional Neural Networks (CNNs) have found applications in many vision-related domains ranging such as generic image-understanding to self-driving cars [4]. However, deep neural networks are increasingly computationally and memory intensive, and need to work with embedded hardware which have limited energy and memory availability. To address this problem, we presented algorithms for improving the efficiency and accuracy of compressed deep networks. In this thesis, we developed three improvements over existing network compression literature. Distribution Aware Binary Networks, which offers a binary representation specific to the input weight distribution. Hybrid Binary Networks, where we demonstrated that binarizing the right areas in the network contributes significantly to the overall accuracy of the network and does not damage its speed-ups in Hybrid Binary Networks. Deep Expander Networks provided a principled method to prune networks before considering the training data. These methods share a common principle: utilizing over-parameterization in deep neural networks for developing fast, accurate and compact deep networks.

We first showed that binary networks have similar representation power as infinite-precision networks. We proposed a distribution-aware binary representation for layer-weights. Using dynamic programming, we came up with an algorithm for computing this representation efficiently. We got significant improvements on large-scale datasets, showing the efficacy of our algorithm. We provide inuitions and reductions to previous binarization techniques.

Secondly, we presented the idea that avoiding binarization of few select layers can restore the accuracy boost, while nearly maintaining the compression rates. We proposed a heuristic for selecting the layers to be binarized. We successfully weight-binarized the last layers without significant accuracy drops, a problem faced by previous works in this area. We also showed that we can successfully combine the advantages of our approach with other architectural compression strategies, to obtain highly efficient models with negligible accuracy penalties. Overall, we asked the question: Where to binarize to augment our previously proposed binarization strategy.

There can be several possible directions to extend this work. We see promise in binary networks and hope to further develop binary representations which are competitive with floating point representations in accuracy.

Finally, we attack pruning in a novel fashion. We provide a principled approach for pruning before training by utilizing concepts from graph theory: particularly using expander graphs that give strong theoretical guarantees on connectivity. The resulting architecture (X-Net) is shown to be highly efficient in terms of both computational requirements and model size, achieve significant improvements over the state-of-the-art architectures in performance on a parameter or run-time budget. The principle used is: global information flows can be maintained even while sparsifying the model significantly, producing efficient deep networks. We hope that this work motivates other approaches to utilize results from graph theory to develop efficient network architectures. Further interesting directions include using random graphs for architecture design, combine pruning and binarization to provide extremely compact networks, etc.

Related Publications

- Ameya Prabhu, Vishal Batchu, Rohit Gajawada, Sri Aurobindo Munagala, Anoop Namboodiri **Hybrid Binary Networks: Optimizing for Accuracy, Efficiency and Memory**. *IEEE Winter conference on Applications of Computer Vision (WACV 2018)*, Lake tahoe, USA. (Oral)
- Ameya Prabhu, Vishal Batchu, Sri Aurobindo Munagala, Rohit Gajawada, Anoop Namboodiri. **Distribution-Aware Binarization of Neural Networks for Sketch Recognition**. *IEEE Winter conference on Applications of Computer Vision (WACV 2018)*, Lake tahoe, USA. (Oral)
- Ameya Prabhu*, Girish Varma*, Anoop Namboodiri. **Deep Expander Networks: Efficient Deep Networks from Graph Theory**. *European Conference on Computer Vision (ECCV 2018)*, Munich, Germany. (Oral)

Appendix A

Binary Networks: Appendix

This appendix consists of additional details regarding Chapters 3 and 4.

A.1 Optimal representation of $\widetilde{\mathbf{W}}$

Theorem 2. 1 *The optimal binary weight* $\widetilde{\mathbf{W}}$ *which minimizes the error function* $\mathbf{J}$ *is*

$$\widetilde{\mathbf{W}} = \alpha\mathbf{e} + \beta(\mathbf{1} - \mathbf{e})$$

where $\alpha = \frac{\mathbf{W}^T\mathbf{e}}{K}$, $\beta = \frac{\mathbf{W}^T(\mathbf{1}-\mathbf{e})}{n-K}$ *and the optimal* $\mathbf{e}^*$ *is* $\mathbf{e}^* = \underset{e,K}{argmax}(\frac{\|\mathbf{W}^T\mathbf{e}\|^2}{K} + \frac{\|\mathbf{W}^T(\mathbf{1}-\mathbf{e})\|^2}{n-K})$

Proof: The approximated weight vector $\widetilde{\mathbf{W}} = [\alpha\alpha \dots \beta\alpha \dots \beta\beta]$ can be decomposed as:

$$[\alpha\alpha \dots \beta\alpha \dots \beta\beta] = \alpha \cdot [11 \dots 01 \dots 00] + \beta \cdot [00 \dots 10 \dots 11]$$

where without loss of generality, $\mathbf{e} \in \{0,1\}^n, \mathbf{e}^T\mathbf{e} > 0, \mathbf{e} \in \{0,1\}^n, (\mathbf{1}-\mathbf{e})^T(\mathbf{1}-\mathbf{e}) > 0$ and $\alpha, \beta \in \mathbb{R}$. This is because the trivial case where $\mathbf{e} = \mathbf{0}$ or $\mathbf{e} = \mathbf{1}$ is covered by substituting $\alpha = \beta$ instead and the equation is independent of $\mathbf{e}$. We have to find the values of α and β which would be the best approximation of this vector.
Let us define the error function $\mathbf{J} = \mathbf{W} - (\alpha \cdot \mathbf{e} + \beta \cdot (\mathbf{1} - \mathbf{e}))$. We have to minimize $\| \mathbf{J} \|^2 = \mathbf{E}$, where:

$$\mathbf{E} = (\mathbf{W} - (\alpha \cdot \mathbf{e} + \beta \cdot (\mathbf{1} - \mathbf{e})))^T(\mathbf{W} - (\alpha \cdot \mathbf{e} + \beta \cdot (\mathbf{1} - \mathbf{e}))) \tag{A.1}$$

$$\begin{aligned}\mathbf{E} = \mathbf{W}^T\mathbf{W} + \alpha^2 \cdot \mathbf{e}^T\mathbf{e} + \beta^2(\mathbf{1}-\mathbf{e})^T(\mathbf{1}-\mathbf{e}) \\ - 2\alpha \cdot \mathbf{W}^T\mathbf{e} - 2\beta \cdot \mathbf{W}^T(\mathbf{1}-\mathbf{e}) + 2\alpha\beta\mathbf{e}^T(\mathbf{1}-\mathbf{e})\end{aligned} \tag{A.2}$$

where $\mathbf{e}^T\mathbf{e} = K$, then $(\mathbf{1}-\mathbf{e})^T(\mathbf{1}-\mathbf{e}) = n - K$ and $\mathbf{e}^T(\mathbf{1}-\mathbf{e}) = 0$. Substituting these in, we get

$$\mathbf{E} = \mathbf{W}^T\mathbf{W} + \alpha^2K + \beta^2(n-K) - 2\alpha \cdot \mathbf{W}^T\mathbf{e} - 2\beta \cdot \mathbf{W}^T(\mathbf{1}-\mathbf{e}) \tag{A.3}$$

We minimize this equation with respect to α and β giving us:

$$\frac{\partial \mathbf{E}}{\partial \alpha} = 0, \frac{\partial \mathbf{E}}{\partial \beta} = 0 \tag{A.4}$$

Solving the above, we get the equations:

$$\frac{\partial \mathbf{E}}{\partial \alpha} = 2\alpha K - 2 \cdot \mathbf{W}^T \mathbf{e} = 0$$

$$\frac{\partial \mathbf{E}}{\partial \beta} = 2\beta(n-K) - 2 \cdot \mathbf{W}^T(\mathbf{1} - \mathbf{e}) = 0$$

We can get the values of α and β from the above equations.

$$\alpha = \frac{\mathbf{W}^T \mathbf{e}}{K}, \beta = \frac{\mathbf{W}^T(\mathbf{1} - \mathbf{e})}{(n-K)}$$

Then substituting the values of α and β in equation A.3, we get

$$\mathbf{E} = \| \mathbf{W}||^2 + \frac{\| \mathbf{W}^T \mathbf{e} \|^2}{K} + \frac{\| \mathbf{W}^T(\mathbf{1} - \mathbf{e}) \|^2}{n-K} - 2\frac{\| \mathbf{W}^T \mathbf{e} \|^2}{K} - 2\frac{\| \mathbf{W}^T(\mathbf{1} - \mathbf{e}) \|^2}{n-K} \tag{A.5}$$

$$\mathbf{E} = \| \mathbf{W}||^2 - (\frac{\| \mathbf{W}^T \mathbf{e} \|^2}{K} + \frac{\| \mathbf{W}^T(\mathbf{1} - \mathbf{e}) \|^2}{n-K}) \tag{A.6}$$

In the above equation, we want to minimize $\mathbf{E}$. Since $\mathbf{W}$ is a given value, we need maximize the second term to minimize the expression. For a given K, $\mathbf{e_K} = sgn(\mathbf{T_k})$ where $\mathbf{T_k} = topk(\mathbf{W}, K)$. Here, $topk(\mathbf{W}, K)$ represents the top K values of $\mathbf{W}$ corresponding to either the largest positive K values or the largest negative K values, which remain as is whereas the rest are converted to zeros.

$$\mathbf{e}^\star = \underset{e,K}{argmax}(\frac{\| \mathbf{W}^T \mathbf{e} \|^2}{K} + \frac{\| \mathbf{W}^T(\mathbf{1} - \mathbf{e}) \|^2}{n-K})$$

Selecting the $topk(\mathbf{W}, K)$ would be optimal since $||\mathbf{W}^T\mathbf{e}||$ and $||\mathbf{W}^T(\mathbf{1} - \mathbf{e})||$ are both maximized on selecting either the largest K positive values or the largest K negative values. Hence, this allows us to select the optimal $\mathbf{e}$ given a K. With this, we obtain the optimal $\mathbf{e}$.

A.2 Gradient derivation

$$W \approx \widetilde{\mathbf{W}} = \alpha \mathbf{e} + \beta(\mathbf{1} - \mathbf{e})$$

$$where\ \alpha = \frac{\mathbf{W}^T \mathbf{e}}{K}\ and\ \beta = \frac{\mathbf{W}^T(\mathbf{1} - \mathbf{e})}{n-K}$$

Let $\mathbf{T_k} = topk(\mathbf{W}, K)$, and $\widetilde{\mathbf{W}_1} = \alpha\mathbf{e}$, and $\widetilde{\mathbf{W}_2} = \beta(\mathbf{1} - \mathbf{e})$.
Considering α, on substituting $e = sgn(T_k)$.

$$\alpha = \frac{\mathbf{W}^T\mathbf{e}}{K}$$
$$\therefore \alpha = \frac{\mathbf{W}^T sgn(\mathbf{T_k})}{K}$$

Hence, we have $\alpha = \frac{\mathbf{W}^T sgn(\mathbf{T_k})}{K}$ and similarly $\beta = \frac{\mathbf{W}^T(1-sgn(\mathbf{T_k}))}{n-K}$. Putting these back in $\widetilde{\mathbf{W}}$, we have,

$$\therefore \widetilde{\mathbf{W}} = \frac{\mathbf{W}^T sgn(\mathbf{T_k})}{K} \circ sgn(\mathbf{T_k}) + \frac{\mathbf{W}^T(1 - sgn(\mathbf{T_k}))}{n-K} \circ (1 - sgn(\mathbf{T_k})) \tag{A.7}$$

Now, we compute the derivatives of α and β with respect to $\mathbf{W}$,

$$\frac{d\alpha}{d\mathbf{W}} = \frac{d(\mathbf{W}^T sgn(\mathbf{T_k}))}{d\mathbf{W}} . \frac{1}{K}$$
$$\frac{d\alpha}{d\mathbf{W}} = \frac{d(\mathbf{T_k}^T sgn(\mathbf{T_k}))}{d\mathbf{W}} . \frac{1}{K}$$

$$\begin{aligned} \frac{d\alpha}{d\mathbf{W}} &= \frac{d(||\mathbf{T_k}||_{l1})}{d\mathbf{W}} . \frac{1}{K} \\ &= \frac{sgn(\mathbf{T_k})}{K} \end{aligned} \tag{A.8}$$

Similarly,

$$\begin{aligned} \frac{d\beta}{d\mathbf{W}} &= \frac{d(||\mathbf{W} - \mathbf{T_k}||_{l1})}{d\mathbf{W}} . \frac{1}{n-K} \\ &= \frac{sgn(\mathbf{W} - \mathbf{T_k})}{n-K} \end{aligned} \tag{A.9}$$

Now, $\widetilde{\mathbf{W}_1} = \alpha\mathbf{e}$ therefore,

$$\frac{d\widetilde{\mathbf{W}_1}}{d\mathbf{W}} = e\frac{d\alpha}{d\mathbf{W}} + \alpha\frac{d\mathbf{e}}{d\mathbf{W}}$$
$$\therefore \frac{d\widetilde{\mathbf{W}_1}}{d\mathbf{W}} = \frac{sgn(\mathbf{T_k})}{K} \circ sgn(\mathbf{T_k}) + \alpha.STE(\mathbf{T_k})$$

With this, we end up at the final equation for $\widetilde{\mathbf{G_1}} = \frac{d\widetilde{\mathbf{W}_1}}{d\mathbf{W}}$ as mentioned in the paper,

$$\therefore \widetilde{\mathbf{G_1}} = \frac{sgn(\mathbf{T_k})}{K} \circ sgn(\mathbf{T_k}) + \frac{||\mathbf{T_k}||_{l1}}{K} STE(\mathbf{T_k}) \tag{A.10}$$

Considering the second term $\widetilde{\mathbf{W}_2}$, we have,

$$\frac{d\widetilde{\mathbf{W}_2}}{d\mathbf{W}} = (\mathbf{1} - \mathbf{e})\frac{d\beta}{d\mathbf{W}} + \beta\frac{d(\mathbf{1} - \mathbf{e})}{d\mathbf{W}}$$

$$\therefore \frac{d\widetilde{\mathbf{W}_2}}{d\mathbf{W}} = \frac{sgn(\mathbf{W}-\mathbf{T_k})}{n-K} \circ (1 - sgn(\mathbf{T_k})) + \beta.STE(\mathbf{W}-\mathbf{T_k})$$

This provides us $\widetilde{\mathbf{G_2}} = \frac{d\widetilde{\mathbf{W}_2}}{d\mathbf{W}}$ as mentioned in the paper,

$$\widetilde{\mathbf{G_2}} = \frac{sgn(\mathbf{W}-\mathbf{T_k})}{n-K} \circ (1 - sgn(\mathbf{T_k})) + \frac{||\mathbf{W}-\mathbf{T_k}||_{l1}}{n-K}.STE(\mathbf{W}-\mathbf{T_k}) \tag{A.11}$$

Together, we arrive at our final gradient $\widetilde{\mathbf{G}} = \frac{d\widetilde{\mathbf{W}}}{d\mathbf{W}}$,

$$\widetilde{\mathbf{G}} = \widetilde{\mathbf{G_1}} + \widetilde{\mathbf{G_2}} \tag{A.12}$$

A.3 Binary Networks as Approximators

We define m_k as the number of neurons required to approximate a polynomial of n terms, given the network has a depth of k. We show that this number is bounded in terms of n and k.

Theorem 7 *For $p(x)$ equal to the product $x_1x_2\cdots x_n$, and for any s with all nonzero Taylor coefficients, we have:*

$$m_k(p,s) = \mathcal{O}\left(n^{(k-1)/k} \cdot 2^{n^{1/k}}\right). \tag{A.13}$$

Proof: We construct a binary network in which groups of the n inputs are recursively multiplied. The n inputs are first divided into groups of size b_1, and each group is multiplied in the first hidden layer using 2^{b_1} binary neurons (as described in [49]). Thus, the first hidden layer includes a total of $2^{b_1}n/b_1$ binary neurons. This gives us n/b_1 values to multiply, which are in turn divided into groups of size b_2. Each group is multiplied in the second hidden layer using 2^{b_2} neurons. Thus, the second hidden layer includes a total of $2^{b_2}n/(b_1b_2)$ binary neurons.

We continue in this fashion for $b_1, b_2, \ldots, b_k$ such that $b_1b_2\cdots b_k = n$, giving us one neuron which is the product of all of our inputs. By considering the total number of binary neurons used, we conclude

$$m_k(p,s) \leq \sum_{i=1}^{k} \frac{n}{\prod_{j=1}^{i} b_j} 2^{b_i} = \sum_{i=1}^{k} \left(\prod_{j=i+1}^{k} b_j\right) 2^{b_i}. \tag{A.14}$$

Setting $b_i = n^{1/k}$, for each i, gives us the desired bound (A.13).

A.4 Expressibility of Binary Networks

A binary neural network (a network with weights having only two possible values, such as $+1$ and -1) with a single hidden layer of m binary-valued neurons that approximates a product gate for n inputs can

be formally written as a choice of constants a_{ij} and w_j satisfying

$$\sum_{j=1}^{m} w_j \sigma \left(\sum_{i=1}^{n} a_{ij} x_i \right) \approx \prod_{i=1}^{n} x_i. \tag{A.15}$$

[49] shows that 2^n neurons are sufficient to approximate a product gate with n inputs - each of these weights are assigned, in the proof, a value of $+1$ or -1 before normalization, and all coefficients a_{ij} also have $+1/-1$ values. This essentially makes it a binary network. Weight normalization introduces a scaling constant of sorts, $\frac{1}{2^n n! \sigma_n}$, which would translate to α in our representation, with its negative denoting β.
The above shows how binary networks are expressive enough to approximate real-valued networks, without the need for higher bit quantization.

Note: The above proofs for expressibility power have been borrowed from [49].

A.5 Experimental details

We used the Adam optimizer for all the models with a maximum learning rate of 0.002 and a minimum learning rate of 0.00005 with a decay factor of 2. All networks are trained from scratch. Weights of all layers except the first were binarized throughout our experiments. Our FBin layer is structured the same as the XNOR-Net. We performed our experiments using a cluster of GeForce GTX 1080 Tis using PyTorch v0.2.

A.6 Experimental details

Here we present additional experimental details for experiments in Chapter 3 and 4.

A.6.1 Data processing

For all the datasets, we resized the images to 256×256. A 224×224 (225×225 for Sketchanet) sized crop was randomly taken from an image with standard augmentations such as rotation and horizontal flipping for TU-Berlin and Sketchy. In the TU-Berlin dataset, we use three-fold cross-validation which gives us a 2:1 train-test split to make our results comparable to all previous methods. For Sketchy, we use the training images for retrieval as the training images for classification and validation images for retrieval as the validation images for classification. We train models using the standard training and validation data for ImageNet-12. We report ten-crop accuracies for TU-Berlin and Sketchy, and only single-crop accuracies for ImageNet.

A.6.2 Hyper-parameters

We use the PyTorch framework to train our networks. We used TU-Berlin and Sketchy datasets to evaluate Sketch-A-Net, ResNet-18, and SqueezeNet (v1.0) architectures, and ImageNet data on AlexNet and ResNet-18 architectures. Each FullBinConv block was structured as in XNOR-Net (Batchnorm-Activ-Conv-ReLU). Each WeightBinConv and Conv block has the standard convolutional block structure (Conv-Batchnorm-ReLU). Weights of all layers excepting the first were binarized throughout our experiments, unless specified otherwise. We used a dropout of 0.2 before the last two convolutional layers in Sketch-A-Net and AlexNet, and a dropout of 0.2 before the last layer in SqueezeNet except after an FBin layer as followed in the XNOR-Net paper. All networks were trained from scratch. We used the Adam optimizer for all the models with a maximum learning rate of 0.002 and a minimum of 0.00005 with a decay factor of 2. We do not use a bias term or weight decay for FullBinConv and WeightBinConv layers. We used a batch size of 256 for all Sketch-A-Net models and a batch size of 128 for ResNet-18 and SqueezeNet models, the maximum size which fits in a 1080Ti GPU.

A.7 FLOPs, Exploiting Filter Repetition and Computational Cost Calculation

Layers of AlexNet and ResNet-18 models are shown in following Tables, along with the number of parameters and the corresponding FLOPs. The number of parameters of each layer for FPrec and Binary versions of the model shown in multiples of 0.1 million for the sake of clarity. The number of parameters in the Binary version of a given layer (#BinParams) is calculated as follows:

FLOPs through each layer are given for FPrec, WBin, FBin and Hybrid versions, in multiples of 10 million. Except for the first layers, where weights are not binarized, the other layers have 58 times lesser FBin FLOP values due to enabling of XNOR/popcount operations. The number of repeated parameter values are also indicated and equivalent number of parameters in corresponding binarized layers are calculated as

$$\text{\#WeightBinConvFLOPs} = \text{\#FLOPs} \times (1 - \text{Repeated})$$

Then, we calculate the FLOPs in the FBin model by

$$\text{\#FullBinConvFLOPs} = \frac{\text{\#WbinFLOPs}}{58}$$

Further on, we select the parameters and FLOPs for each layer of the Hybrid models by selecting whether the layer is WeightBinConv or FullBinConv and pick the corresponding FLOPs.
The total number of parameters and FLOPs are calculated as the sum of individual layer FLOPs.

A.8 Models used

AlexNet: A deep CNN that paved the way for most state-of-the-art CNN based models that exist today, and serves as a benchmark for comparison of network compression techniques.
ResNet-18: A residual, implicitly compact model which is substantially deeper than previous standard models, that can be trained easily.
Sketch-A-Net: A benchmark AlexNet-based model that was specifically engineered for sketches that beats human accuracy at sketch recognition, beating standard models fine-tuned on ImageNet which were unsuitable for sketch data.
SqueezeNet: An explicitly compact CNN architecture achieving AlexNet accuracy with 50x fewer parameters.

A.9 Model Architectures

Figure A.1 is a comparison of architectures of FPrec, WBin, FBin, and two Hybrid versions of the Sketch-A-Net model. The hybrids replace almost all convolutional layers with FullBinConv layers, except the ones towards the end, which are replaced with WeightBinConv layers. Figures A.2 and A.3 show a similar architectural comparison for ResNet-18 and SqueezeNet models.

A.10 Binarization-errors across layers

In Figure 2 in the main paper, we use the proposed metric to measure binarization-error across layers in Sketch-A-Net, ResNet-18, and SqueezeNet models. We use stars to indicate where our algorithm replaces a layer with a WeightBinConv layer, and squares when our algorithm replaces a layer with a FullBinConv layer. Our metric is a function of approximation-error and the inverse of the number of FLOPs through the layer. Observe that our algorithm replaces layers with high error scores with FullBinConv layers, which occur towards the end, and weight-binarizes the rest.
The partitioning algorithm clustered these layers to mark them for full-binarization and weight-binarization. For example, in ResNet-18, the algorithm gave two major partitions - layers 2 to 11 in the first, and 12 to 16 in the next, with a significant difference in the cluster means. Hence, layers 12 to 16 were binarized for the hybrid model.

Layers	Parameters in 0.1M		FLOPs				
	FPrec	Bin	FPrec	Repeats	WBin	FBin	Hybrid
AlexNet							
Conv1	0.23	0.232	10.54	0.00	10.54	10.54	10.54
Conv2	3.07	0.096	44.79	0	44.79	0.77	0.77
Conv3	6.64	0.207	14.95	0.71	4.34	0.07	0.07
Conv4	8.85	0.276	22.43	0.71	6.50	0.11	0.11
Conv5	5.90	0.184	14.95	0.60	5.98	0.10	0.10
Conv-FC1	377.49	11.796	3.77	0.00	3.77	0.07	3.77
Conv-FC2	167.77	5.243	1.68	0.00	1.68	0.03	1.68
Conv-FC3	40.96	1.280	0.41	0.00	0.41	0.41	0.41
Total	610.90	19.316	113.53		78.01	12.11	17.47

Table A.1: Layers of the AlexNet model, with the number of parameters and FLOPs for versions (WBin, Fbin, Hybrid, FPrec) of each. Also, the amount of unique parameters (a high number indicating high compressibility) is shown for each layer.

Layers	Parameters in 0.1M		FLOPs				
	FPrec	Bin	FPrec	Repeats	WBin	FBin	Hybrid
Sketch-A-Net							
Conv1	0.23	0.232	7.26	0.00	7.26	7.26	7.26
Conv2	3.07	0.096	19.68	0.00	19.68	0.34	0.34
Conv3	6.64	0.207	6.64	0.58	2.80	0.05	0.05
Conv4	8.85	0.276	13.27	0.62	5.02	0.09	0.09
Conv5	5.90	0.184	13.27	0.61	5.19	0.09	0.09
Conv-FC1	47.19	1.475	0.64	0.00	0.64	0.01	0.64
Conv-FC2	2.62	0.082	0.03	0.00	0.03	0.00	0.03
Conv-FC3	1.28	0.040	0.05	0.00	0.05	0.05	0.05
Total	75.77	2.593	60.84		40.68	7.89	8.54

Table A.2: Layer descriptions of the Sketch-A-Net model.

Layers	Parameters in 0.1M		FLOPs				
	FPrec	Bin	FPrec	Repeats	WBin	FBin	Hybrid
ResNet-18							
Conv1	0.09	0.094	11.80	0.00	11.80	11.80	11.80
Conv2	0.37	0.012	11.56	0.23	8.86	0.15	0.15
Conv3	0.37	0.012	11.56	0.31	7.99	0.14	0.14
Conv4	0.37	0.012	11.56	0.23	8.90	0.15	0.15
Conv5	0.37	0.012	11.56	0.23	8.86	0.15	0.15
Conv6	0.74	0.023	5.78	0.49	2.92	0.05	0.05
Conv7	1.47	0.046	11.56	0.39	7.02	0.12	0.12
Conv8	0.08	0.003	0.64	0.00	0.64	0.01	0.01
Conv9	1.47	0.046	11.56	0.38	7.19	0.12	0.12
Conv2d	1.47	0.046	11.56	0.39	7.08	0.12	0.12
Conv2d	2.95	0.092	5.78	0.57	2.46	0.04	0.04
Conv2d	5.90	0.184	11.56	0.50	5.74	0.10	0.10
Conv2d	0.33	0.010	0.64	0.00	0.64	0.01	0.01
Conv2d	5.90	0.184	11.56	0.52	5.51	0.10	5.51
Conv2d	5.90	0.184	11.56	0.57	5.00	0.09	5.00
Conv2d	11.80	0.369	5.78	0.70	1.76	0.03	1.76
Conv2d	23.59	0.737	11.56	0.67	3.83	0.07	3.83
Conv2d	1.31	0.041	0.64	0.00	0.64	0.01	0.64
Conv2d	23.59	0.737	11.56	0.71	3.38	0.06	3.38
Conv2d	23.59	0.737	11.56	0.76	2.73	0.05	2.73
Linear	5.12	0.160	0.05	0.00	0.05	0.05	0.05
Total	116.79	3.741	181.41		103.02	13.42	35.89

Table A.3: Layers descriptions of the ResNet-18 model.

Layers	Parameters in 0.1M		FLOPs				
	FPrec	Bin	FPrec	Repeats	WBin	FBin	Hybrid
Squeezenet							
Conv2d	0.14	0.141	16.77	0.00	16.77	16.77	16.77
Conv2d	0.02	0.000	0.45	0.00	0.45	0.01	0.45
Conv2d	0.01	0.000	0.30	0.00	0.30	0.01	0.30
Conv2d	0.09	0.003	2.69	0.25	2.03	0.03	2.03
Conv2d	0.02	0.001	0.60	0.00	0.60	0.01	0.60
Conv2d	0.01	0.000	0.30	0.00	0.30	0.01	0.01
Conv2d	0.09	0.003	2.69	0.20	2.15	0.04	0.04
Conv2d	0.04	0.001	1.19	0.00	1.19	0.02	1.19
Conv2d	0.04	0.001	1.19	0.00	1.19	0.02	0.02
Conv2d	0.37	0.012	10.75	0.33	7.15	0.12	0.12
Conv2d	0.08	0.003	0.60	0.00	0.60	0.01	0.60
Conv2d	0.04	0.001	0.30	0.00	0.30	0.01	0.01
Conv2d	0.37	0.012	2.69	0.36	1.73	0.03	0.03
Conv2d	0.12	0.004	0.90	0.00	0.90	0.02	0.90
Conv2d	0.09	0.003	0.67	0.00	0.67	0.01	0.01
Conv2d	0.83	0.026	6.05	0.48	3.15	0.05	0.05
Conv2d	0.18	0.006	1.34	0.00	1.34	0.02	1.34
Conv2d	0.09	0.003	0.67	0.00	0.67	0.01	0.01
Conv2d	0.83	0.026	6.05	0.54	2.77	0.05	0.05
Conv2d	0.25	0.008	1.79	0.00	1.79	0.03	1.79
Conv2d	0.16	0.005	1.19	0.00	1.19	0.02	0.02
Conv2d	1.47	0.046	10.75	0.63	4.02	0.07	0.07
Conv2d	0.33	0.010	0.55	0.00	0.55	0.01	0.55
Conv2d	0.16	0.005	0.28	0.00	0.28	0.00	0.28
Conv2d	1.47	0.046	2.49	0.74	0.66	0.01	0.66
Conv2d	5.12	0.160	8.65	0.00	8.65	8.65	8.65
Total	12.44	0.526	61.09		41.26	9.22	16.40

Table A.4: Layers descriptions of the SqueezeNet model.

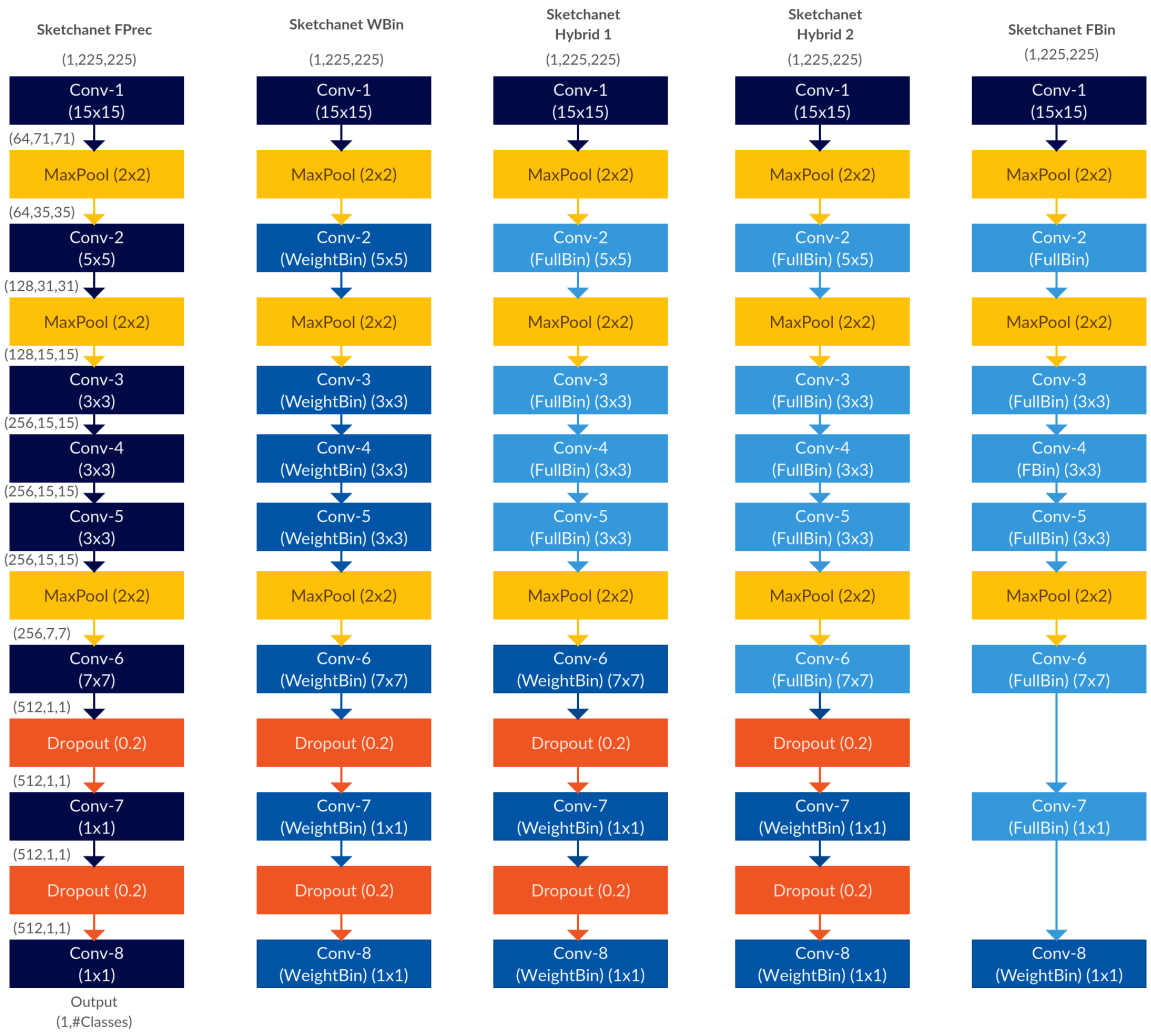


Figure A.1: Comparing architectures of FPrec, Wbin, Fbin, and two Hybrid versions of Sketch-A-Net. Our hybrid versions replace most conv layers with FullBinConv layers, but replace layers towards the end with WeightBinConv layers, following the algorithm.

Figure A.2: Comparing architectures of FPrec, Wbin, Fbin, and two Hybrid versions of ResNet-18.

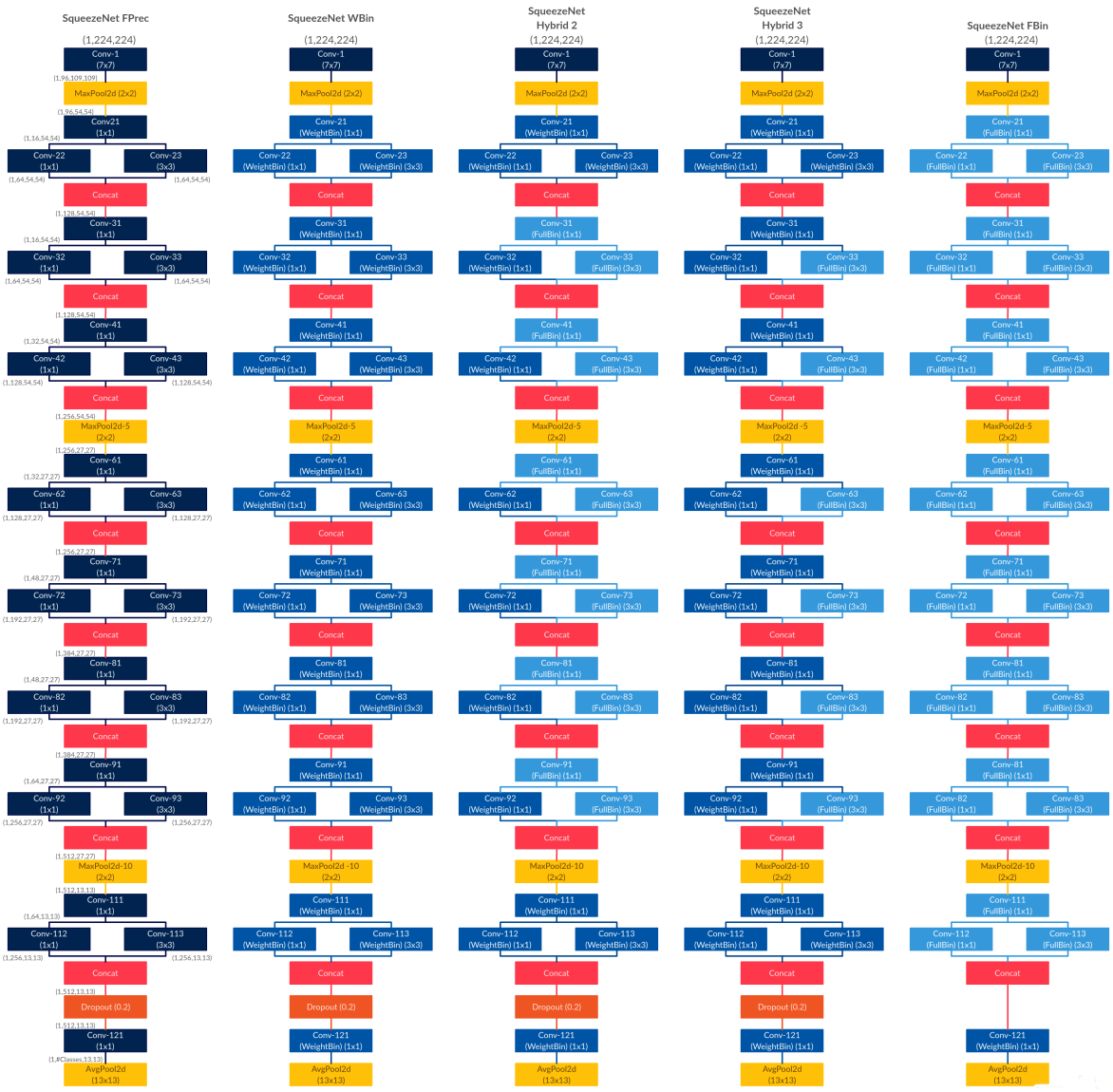


Figure A.3: Comparing architectures of FPrec, Wbin, Fbin, and two Hybrid versions of SqueezeNet.

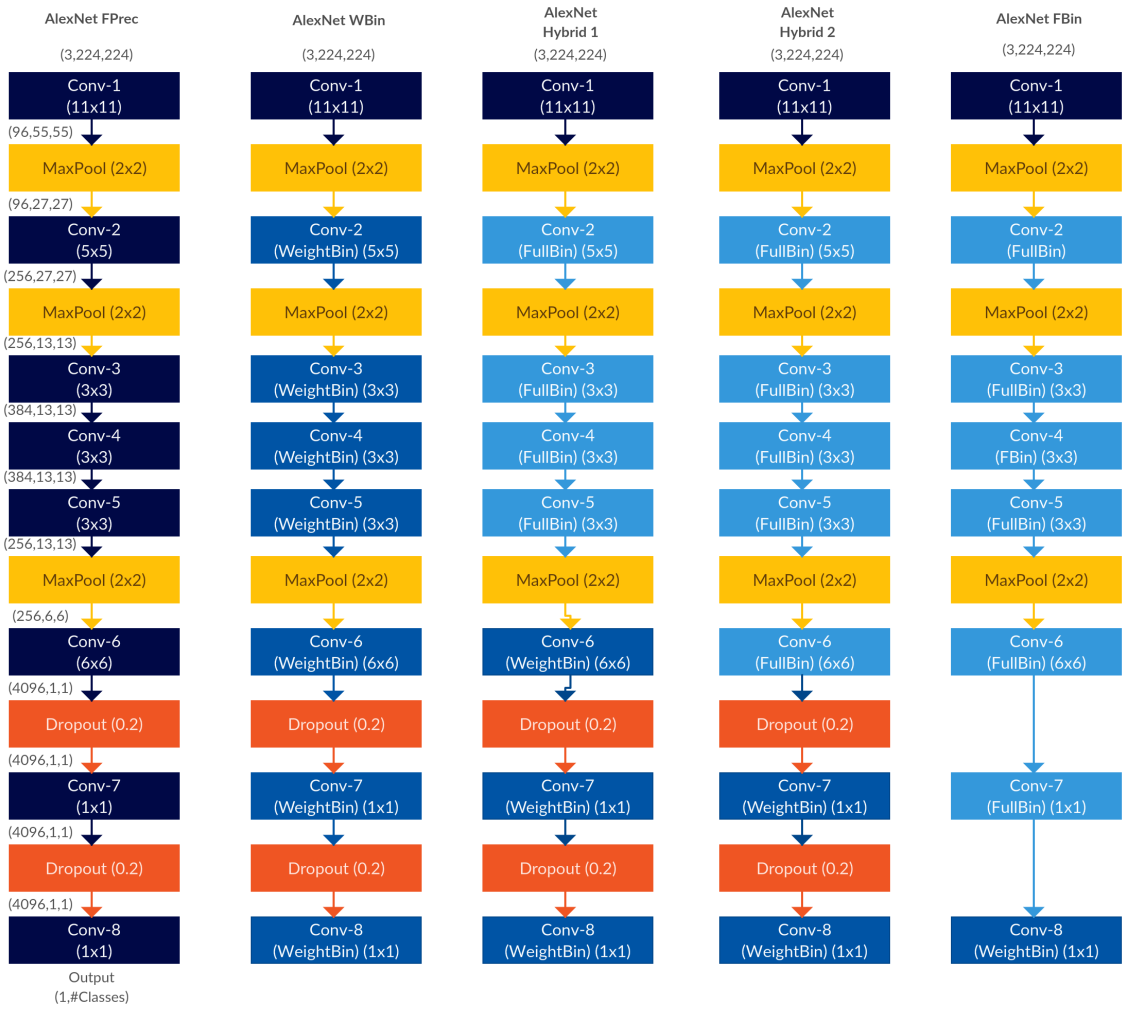


Figure A.4: Comparing architectures of FPrec, Wbin, Fbin, and two Hybrid versions of AlexNet.

Appendix B

Deep Expander Networks: Appendix

This part of the appendix provides formal proofs for the theorems stated in Chapter 5 of the thesis. We also include tabulation of results of Figure 3 and 4, alongside additional results that could not be added in the paper due to space constraints.

B.1 Explicit Expanders

Many constructions of exmain paperpander graphs have been explored in the past. We will be using a construction that is comparatively easy to describe and implement. We will be considering Cayley graphs, which are obtained from the theory of finite Fields/Groups. The vertex set of such graphs is a group and the edges are defined by addition operation on the vertices. Also we will be describing expanders that are simple undirected graphs. Given an undirected graph $G = (V, E)$, we can obtain a bipartite expander $G = (V, V', E)$, by making a copy of the vertices V on the other side and adding edges according to E.

B.1.1 Cayley Expander

Let V be a group under an operation $+$ and let $H \subset V$ be a set of generators of the group V. The Cayley graph defined by V, H is the graph with vertex set V and the edges $E = \{(x, x + h) : h \in H\}$. For our construction, we will consider the group $\{0, 1\}^n$ under coordinate-wise XOR operations. For $H \subset \{0, 1\}^n$, the Cayley graph defined by H is $|H|$-regular. It is a well-known result in spectral graph theory that there are set of generators H, for which the Cayley graph defined by $\{0, 1\}^n, H$ forms an expander with spectral gap γ.

Theorem 8 (Alon, Roichman, Proposition 4) *For every ϵ, there exists explicit $H \subset \{0, 1\}^n$ of size $\leq O(n^2/\epsilon^2)$ such that the Cayley graph defined by $\{0, 1\}^n, H$ is an expander with spectral gap $\gamma = 1 - \epsilon$.*

B.2 Sensitivity in Expanders

In this section, we will give details about some properties of expander graphs that are required for proving Theorems 1 and 2. Expander graphs are characterized by the spectral gap, which in turn implies the connectivity properties listed in Section 3.4.

The properties given in Section 3.4 follows from the fact that expanders have a spectral gap. For the sensitivity proofs, we need the following lemmas. The first lemma states that having a spectral gap, implies that all set of vertices of size $\leq n/2$ expands.

Theorem 9 (Theorem 4.6 [73]) *If $G = (V, E)$ is an expander with spectral gap of γ then for every subset $S \subset V$ of size $\leq |V|/2$, the size of the set of neighbors $|N(S)| \geq (1 + \gamma)|S|$.*

The next lemma is known as the expander mixing lemma, deals with the uniform connectivity properties of the expander.

Theorem 10 (Lemma 4.15 [73]) *If $G = (V, E)$ is an D-regular expander with spectral gap of γ then for every subset $S, T \subset V$,*

$$|E(S, T) - D \cdot |S| \cdot |T|/n| \leq (1 - \gamma)\sqrt{|S| \cdot |T|}$$

where $E(S, T)$ is the set of edges from S to T.

In this section, we prove Theorem 1 and Theorem 2 using the properties described above along with the connectivity properties defined in Section 3.1 of the paper.

Theorem 11 (Sensitivity of X-Nets) *Let n be the number of input as well as output nodes in the network and $G_1, G_2, \cdots, G_t$ be D regular bipartite expander graphs with n nodes on both sides. Then every output neuron is sensitive to every input in a Deep X-Linear Network defined by G_i's with depth $t = O(\log n)$.*

Proof: For showing sensitivity, we show that for every pair of input and output (u, v), there is a path in the X-Net with the i^{th} edge from the i^{th} graph. We use the expansion property of the expander graphs. Let $N_1(u)$ be the set of neighbors of u in G_1 and $N_i(u)$ be the set of neighbors of $N_{i-1}(u)$ in the graph G_i. Since each of the graphs are expanding $|N_i(u)| \geq (1 + \gamma) \times |N_{i-1}(u)|$, using Lemma 9. Since $(1 + \gamma) > 1$, we can obtain that for $i = O(\log n)$, $|N_i(u)| \geq n/2$. Similarly we can start from v and define $N_1(v)$ as the set of neighbors of v in G_t and $N_i(v)$ be the set of neighbors of $N_{i-1}(v)$ in the graph G_{n-i-1}. Due to the expansion property, for $j = O(\log n)$, $|N_j(v)| \geq n/2$. Now we choose $t = i + j = O(\log n)$ so that the i^{th} graph from 1 is the same as $j + 1^{th}$ graph from t. Then we will have that $N_i(u) \cap N_j(v) \neq \phi$. That is there is some vertex in the i^{th} graph that is both connected to u and v, which implies that there is a path from u to v.

Theorem 12 (Mixing in Deep Expander Networks) *Let n be the number of input as well as output nodes in the network and $G_1, G_2, \cdots, G_t$ be D regular bipartite expander graphs with n nodes on both sides. Let S, T be subsets of input and output nodes in the X-Linear Network defined by the G_i's. The number of paths between S and T is $\approx D|S||T|/n$*

Proof: First we prove the theorem for the case when $G_1 = G_2 \cdots = G_t = G$. Note that for $t = 1$, the theorem is same as the expander mixing lemma (Lemma 10). For $t > 1$, consider the graph of t length paths denoted by G^t. An edge in this graph denotes that there is a path of length t in G. Observe that the adjacency matrix of G^t is given by the tth power of the adjacency matrix of G and hence the spectral gap of G^t, $\gamma_t = \gamma^t$. Now applying the expander mixing lemma on G^t (Lemma 10), proves the theorem.

For the case when the graph G_i's are different, let $\gamma_{\min}$ be the minimal spectral gap among the graphs. Since all the graphs are D-regular, the largest eigenvector is the all ones vector with eigenvalue D for all the graphs. The eigenvector corresponding to second largest eigenvalue can be different for each graph, but they are orthogonal to the all ones vector. Hence the spectral gap of the t length path graph G is at least $\gamma^t_{\min}$. Finally we apply the expander mixing lemma on G^t to prove the theorem.

B.3 Model Details

We discuss the model structures in detail, along with tabulated values of size and flops of various architectures presented as graphs in the main paper.

AlexNet	**Filter Shape**	**Filter Shape** (**X-AlexNet-1**)	**Filter Shape** (**X-AlexNet-2**)
Conv2d	64 x 3 x 11 x 11	64 x 3 x 11 x 11	64 x 3 x 11 x 11
Conv2d	192 x 64 x 5 x 5	192 x 64 x 5 x 5	192 x 64 x 5 x 5
Conv2d	384 x 192 x 3 x 3	384 x 192 x 3 x 3	384 x 192 x 3 x 3
Conv2d	256 x 384 x 3 x 3	256 x 384 x 3 x 3	256 x 384 x 3 x 3
Conv2d	256 x 256 x 3 x 3	256 x 256 x 3 x 3	256 x 256 x 3 x 3
Linear	9216 x 4096	1024 x 4096	512 x 4096
Linear	4096 x 4096	512 x 4096	512 x 4096
Linear	4096 x 1000	1024 x 1000	1024 x 1000

Table B.1: Filter sizes for the AlexNet model. Notice the filter sizes of the linear layers of the original model has $|V|\times|U|$ parameters, whereas X-AlexNet models have $|V|\times D$ parameters. Note that $D << |U|$ as stated in Section 3.2. Hence, expander graphs model connections in linear layers (X-Linear) effectively.

B.3.1 Filter structure of AlexNet and VGG

In Tables B.1 and B.2, the detailed layer-wise filter structure is tabulated as stated in Section 5.3 of the paper. We compare the sizes of input channels between the filters, and show that with X-Conv

VGG	Filter Shape	Filter Shape (**X-VGG16-1**)	Filter Shape (**X-VGG16-2**)
Conv2d	64 x 3 x 3 x 3	64 x 3 x 3 x 3	64 x 3 x 3 x 3
Conv2d	64 x 64 x 3 x 3	64 x 64 x 3 x 3	64 x 64 x 3 x 3
Conv2d	128 x 64 x 3 x 3	128 x 64 x 3 x 3	128 x 64 x 3 x 3
Conv2d	128 x 128 x 3 x 3	128 x 64 x 3 x 3	128 x 64 x 3 x 3
Conv2d	256 x 128 x 3 x 3	256 x 32 x 3 x 3	256 x 16 x 3 x 3
Conv2d	256 x 256 x 3 x 3	256 x 32 x 3 x 3	256 x 16 x 3 x 3
Conv2d	256 x 256 x 3 x 3	256 x 32 x 3 x 3	256 x 16 x 3 x 3
Conv2d	512 x 256 x 3 x 3	512 x 32 x 3 x 3	512 x 16 x 3 x 3
Conv2d	512 x 512 x 3 x 3	512 x 32 x 3 x 3	512 x 16 x 3 x 3
Conv2d	512 x 512 x 3 x 3	512 x 32 x 3 x 3	512 x 16 x 3 x 3
Conv2d	512 x 512 x 3 x 3	512 x 32 x 3 x 3	512 x 16 x 3 x 3
Conv2d	512 x 512 x 3 x 3	512 x 32 x 3 x 3	512 x 16 x 3 x 3
Conv2d	512 x 512 x 3 x 3	512 x 32 x 3 x 3	512 x 16 x 3 x 3
Linear	512 x 512	128 x 512	128 x 512
Linear	512 x 10	512 x 10	512 x 10

Table B.2: Filter sizes for the VGG-16 model on CIFAR-10 dataset. The filter sizes given are $|V|\times|U|\times c\times c$ in original VGG network, $|V|\times D\times c\times c$ in our X-VGG16 models. Note that $D << |U|$ as stated in Section 3.2. Hence, expander graphs model connections in Convolutional layers (X-Conv) effectively.

and X-Linear layers, we can train models effectively even with upto 32x and 18x times smaller filters in input dimension in VGG16 and AlexNet models respectively. Hence, modeling connections as weighted adjacency matrix of an Expander graph is an effective method to model connections between neurons, producing highly efficient X-Nets.

B.3.2 Results

As stated in Section 5.2, Tables B.3 and B.4 presented below give the detailed accuracy, parameters and FLOPs of models displayed in the Figure 3 in the paper. Details of other models are also provided in the same table, which could not be displayed in the paper due to lack of space.

Table B.3 displays the performance of X-DenseNet models on the CIFAR-10 and CIFAR-100 datasets. If we compare models that have similar number of parameters, we achieve around 0.2% and 0.6% increase in accuracy over DenseNet-BC models on CIFAR-10 and CIFAR-100 datasets respectively. In the same manner, we can achieve upto using only two-thirds of the parameter and runtime cost cost respectively, keeping accuracy constant on CIFAR-10 and CIFAR-100 datasets as stated in the paper.

Similarly, Table B.4 displays the performance of X-ResNet and X-DenseNet models on the ImageNet datasets. We can observe that we achieve around 3.2% and 1% increase in accuracy over ResNet and DenseNet-BC models in the same computational budget. Also, we can observe that we require approximately 15% less FLOPs for achieving similar accuracies over DenseNet models and 15% less parame-

Model	Accuracy	#Params	#FLOPs
CIFAR10		(in M)	(in 100M)
X-DenseNetBC-2-40-24	**94.83%**	**0.4M**	**1.44**
DenseNetBC-40-24	94.79%	0.7M	2.88
X-DenseNetBC-2-40-36	94.98%	0.75M	3.24
X-DenseNetBC-2-40-48	**95.48%**	**1.4M**	**5.75**
DenseNetBC-40-36	95.26%	1.5M	6.47
X-DenseNetBC-2-40-60	**95.71%**	**2.15M**	**8.98**
DenseNetBC-40-48	95.64%	2.8M	11.50
DenseNetBC-40-60	95.91%	4.3M	17.96
CIFAR100			
X-DenseNetBC-2-40-24	74.37%	0.4M	1.44
DenseNetBC-40-24	76.05%	0.7M	2.88
X-DenseNetBC-2-40-36	**76.69%**	**0.75M**	**3.24**
DenseNetBC-40-36	77.84%	1.5M	6.47
X-DenseNetBC-2-40-60	78.53%	2.15M	8.98
X-DenseNetBC-4-70-60	**79.56%**	**2.6M**	**10.26**
DenseNetBC-40-48	79.03%	2.8M	11.50
DenseNetBC-40-60	79.87%	4.3M	17.96
X-DenseNetBC-2-70-60	80.89%	5.18M	20.52
DenseNetBC-70-60	81.28%	10.36M	41.05

Table B.3: Results obtained on the state-of-the-art models on CIFAR-10 and CIFAR-100 datasets, ordered by FLOPs per model. X-Nets give significantly better accuracies with corresponding DenseNet models in the same limited computational budget and correspondingly significant parameter and FLOP reduction for models with similar accuracy.

ters for achieving similar accuracies over the ResNet model respectively as stated in the paper. We also observe that X-DenseNetBC models outperform ResNet and X-ResNet models in both compression, parameters and FLOPs and achieve comparable accuracies with MobileNets [30] and ShuffleNets [**?**] in the same parameter budget, albeit with much higher computational cost due to architectural constraints.

Table B.5 displays accuracies, parameters and FLOPs of all the wider and deeper networks trained on CIFAR-100 dataset as discussed in Section 5.4. They are listed in increasing compression order from DensenetBC (1x) to the highest compressed X-DenseNet. The figure indicates that Expander Graphs modeling can scale up to high compression ratios without drastic drops in accuracies, enabling us to train deeper and wider networks retaining similar FLOPs and parameters effectively. We believe this modeling can open up a interesting exploration of training significantly deeper and wider range of models, unlike the current compression techniques as X-Nets are highly compressed networks since definition.

Model	Accuracy	#Params	#FLOPs
ResNet		(in M)	(in 100M)
X-ResNet-2-34	69.23%	11M	35
X-ResNet-2-50	**72.85%**	**13M**	**40**
ResNet-34	71.66%	22M	70
X-ResNet-2-101	**74.87%**	**22.5M**	**80**
ResNet-50	74.46%	26M	80
ResNet-101	75.87%	45M	160
DenseNetBC			
MobileNet [30]	70.6%	4.2M	5.7 [1]
ShuffleNet [82]	70.9%	5M	5.3 [2]
X-DenseNetBC-2-121	**70.5%**	**4M**	28
X-DenseNetBC-2-169	71.7%	7M	33
X-DenseNetBC-2-201	72.5%	10M	43
X-DenseNetBC-2-161	**74.3%**	14.3M	**55**
DenseNetBC-121	73.3%	8M	55
DenseNetBC-169	74.8%	14M	65
DenseNetBC-201	75.6%	20M	85
DenseNetBC-161	76.3%	28.5M	110

Table B.4: Results obtained on the state-of-the-art models on ImageNet dataset, ordered by FLOPs. We also observe that X-DenseNetBC models outperform ResNet and X-ResNet models in both compression, parameters and FLOPs and achieve comparable accuracies with the highly efficient MobileNets and ShuffleNets in the same parameter budget, albeit with much higher FLOPs due to architectural constraints.

B.3.3 Experimental Details

To ensure better reproducibility, we used the same hyper-parameters, models, training schedules and dataset structures from the official PyTorch repository for ResNet and DenseNet-BC ImageNet experiments. Similarly, we followed the pytorch-mobilenet repository including all hyperparameters for all the mobilenet experiments. Likewise, we followed the densenet-pytorch repository by Andreas Veit for all DenseNet-BC experiments on CIFAR-10/100 datasets, and the pytorch-vgg-cifar10 repository by Cheng-Yang Fu for VGG16 experiments on the CIFAR-10 dataset. We deleted a linear layer and remove dropouts from the VGG16 model. There were two differences between our code and the PyTorch ImageNet repository. Our ImageNet training code used a compressed version of Imagenet with images resized to 256x256, we used a batchsize of 128 for all experiments, hence typically results in slightly lower accuracies. Our AlexNet model used BatchNorm layers to stabilize training, with a batchsize of 384.

We trained all the models on a setup consisting of 10 Intel Xeon E5-2640 cores and 2 GeForce GTX 1080 Ti GPUs. All networks are trained from scratch. We modeled connections as random expanders for all experiments. The X-Conv layers did not have a bias. No dropouts were used. Additional details regarding all models, accuracies, and numbers of parameters and FLOPs is tabulated in the supplemen-

tary material for reference due to lack of space.

All models in the plot were trained from the ImageNet code available on Pytorch Official repository including the original DenseNet and ResNet models. Note that this training code is common for all models in the repository and not fine-tuned to any specific model like ResNet or DenseNet. Hence the accuracies we report is slightly lower that those reported using model-specific training code. However, note that we have used the same code for training both the original models and the expander versions of these models. Hence the comparison is a fair one. Same is true with the AlexNet model. Note that we use the original AlexNet architecture, and not CaffeNet.

In contrast, training with fine-tuned code is expected to give a 1-2% improvement over MobileNets in Table B.4. Training schedules suited to X-Nets, hyper-parameter tuning, Activation layers tailored to X-Nets could further improve the accuracies. Overall, we believe that investigating training methods for X-Nets has a lot of potential for improving their performance.

Model	**Accuracy**	**#Params**	**#FLOPs**
Wider		**(in M)**	**(in 100M)**
DenseNetBC-40-60	79.87%	4.3M	17.96
X-DenseNetBC-2-40-60	78.53%	2.15M	8.98
X-DenseNetBC-4-40-60	77.54%	1.08M	4.49
X-DenseNetBC-8-40-60	75.29%	0.54M	2.24
X-DenseNetBC-16-40-60	74.44%	0.27M	1.12
DenseNetBC-40-100	80.9%	11.85M	49.85
X-DenseNetBC-4-40-100	78.87%	2.9M	12.46
X-DenseNetBC-8-40-100	77.75%	1.48M	6.23
X-DenseNetBC-16-40-100	76.2%	0.74M	3.12
DenseNetBC-40-200	81.62%	47.19M	199.28
X-DenseNetBC-4-40-200	80.66%	11.79M	49.82
X-DenseNetBC-10-40-200	79.46%	4.71M	19.93
X-DenseNetBC-20-40-200	78.33%	2.3M	9.96
X-DenseNetBC-30-40-200	77.29%	1.6M	6.64
X-DenseNetBC-50-40-200	75.7%	0.9M	3.99
X-DenseNetBC-80-40-200	73.26%	0.5M	2.49
Deeper			
DenseNetBC-40-60	79.87%	4.3M	17.96
X-DenseNetBC-2-40-60	78.53%	2.15M	8.98
X-DenseNetBC-8-40-60	77.54%	0.54M	2.24
X-DenseNetBC-16-40-60	75.29%	0.27M	1.12
DenseNetBC-58-60	80.79%	7.66M	30.96
X-DenseNetBC-2-58-60	80.29%	3.83M	15.48
X-DenseNetBC-4-58-60	78.74%	1.9M	7.74
X-DenseNetBC-8-58-60	77.98%	0.95M	3.87
X-DenseNetBC-16-58-60	75.87%	0.47M	1.93
DenseNetBC-70-60	81.28%	10.36M	41.05
X-DenseNetBC-2-70-60	80.89%	5.18M	20.52
X-DenseNetBC-4-70-60	79.56%	2.6M	10.26
X-DenseNetBC-8-70-60	77.48%	1.3M	5.13
X-DenseNetBC-16-70-60	77.23%	0.65M	2.57

Table B.5: We display accuracies, parameters and FLOPs of all the wider and deeper networks on CIFAR-100 listed in increasing compression order. This proves that efficiently designing layers like X-Conv and X-Linear allows us to train wider and deeper networks frugally.

Bibliography

[1] A. G. Anderson and C. P. Berg. The high-dimensional geometry of binary neural networks. *arXiv preprint arXiv:1705.07199*, 2017.

[2] Z. Aojun, Y. Anbang, G. Yiwen, X. Lin, and C. Yurong. Incremental network quantization: Towards lossless cnns with low-precision weights. In *ICLR*, 2017.

[3] H. Bagherinezhad, M. Rastegari, and A. Farhadi. Lcnn: Lookup-based convolutional neural network. *CVPR*, 2017.

[4] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[5] T. Bui, L. Ribeiro, M. Ponti, and J. P. Collomosse. Generalisation and sharing in triplet convnets for sketch based visual search. *CoRR*, abs/1611.05301, 2016.

[6] Z. Cai, X. He, J. Sun, and N. Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. *CVPR*, 2017.

[7] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *ICML*, 2015.

[8] J. Cheng, P.-s. Wang, G. Li, Q.-h. Hu, and H.-q. Lu. Recent advances in efficient computation of deep convolutional neural networks. *Frontiers of Information Technology & Electronic Engineering*, 2018.

[9] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary, and S.-F. Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *CVPR*, pages 2857–2865, 2015.

[10] H. M. Chenzhuo Zhu, Song Han and W. J. Dally. Trained ternary quantization. In *ICLR*, 2017.

[11] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017.

[12] M. D. Collins and P. Kohli. Memory bounded deep convolutional networks. *arXiv preprint arXiv:1412.1442*, 2014.

[13] M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*, 2015.

[14] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. In *ICML*, 2016.

[15] G. Cybenko. Approximation by superpositions of a sigmoidal function. *(MCSS)*, 1989.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009.

[17] M. Denil, B. Shakibi, L. Dinh, N. de Freitas, et al. Predicting parameters in deep learning. In *NIPS*, pages 2148–2156, 2013.

[18] M. Eitz, J. Hays, and M. Alexa. How do humans sketch objects? *ACM Trans. Graph. (Proc. SIGGRAPH)*, 2012.

[19] S. Gray, A. Radford, and D. P. Kingma. Gpu kernels for block-sparse weights. *arXiv preprint arXiv:1711.09224*, 2017.

[20] Y. Guo. A survey on methods and theories of quantized neural networks. *arXiv preprint arXiv:1808.04752*, 2018.

[21] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. In *NIPS*, 2016.

[22] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016.

[23] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *NIPS*, pages 1135–1143, 2015.

[24] B. Hassibi, D. G. Stork, G. Wolff, and T. Watanabe. Optimal brain surgeon: Extensions and performance comparisons. In *NIPS*, 1993.

[25] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[26] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. *CVPR*, 2017.

[27] S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *BULL. AMER. MATH. SOC.*, 2006.

[28] M. Horowitz. Computing's energy problem (and what we can do about it). In *International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014.

[29] L. Hou, Q. Yao, and J. T. Kwok. Loss-aware binarization of deep networks. *ICLR*, 2017.

[30] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[31] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017.

[32] G. Huang, S. Liu, L. van der Maaten, and K. Q. Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *CVPR*, 2018.

[33] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[34] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 2017.

[35] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *ICLR*, 2017.

[36] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

[37] J. Johnson, A. Karpathy, and L. Fei-Fei. Densecap: Fully convolutional localization networks for dense captioning. In *CVPR*, 2016.

[38] F. Juefei-Xu, V. N. Boddeti, and M. Savvides. Local binary convolutional neural networks. *CVPR*, 2017.

[39] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

[40] A. Krizhevsky. Cuda-convnet: High-performance c++/cuda implementation of convolutional neural networks. 2012.

[41] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*. 2012.

[43] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

[44] V. Lebedev and V. Lempitsky. Fast convnets using group-wise brain damage. In *CVPR*, 2016.

[45] Y. LeCun, J. S. Denker, and S. A. Solla. Nips. chapter Optimal Brain Damage. 1990.

[46] F. Li, B. Zhang, and B. Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

[47] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *ICLR*, 2017.

[48] D. Lin, S. Talathi, and S. Annapureddy. Fixed point quantization of deep convolutional networks. In *Proceedings of The 33rd International Conference on Machine Learning*, 2016.

[49] H. W. Lin, M. Tegmark, and D. Rolnick. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 2017.

[50] C. Liu, B. Zoph, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. *arXiv preprint arXiv:1712.00559*, 2017.

[51] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017.

[52] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. *ICCV*, 2017.

[53] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *CVPR*, 2017.

[54] M. Masana, J. van de Weijer, L. Herranz, A. D. Bagdanov, and J. Malvarez. Domain-adaptive deep network compression. *Network*, 2017.

[55] P. Merolla, R. Appuswamy, J. V. Arthur, S. K. Esser, and D. S. Modha. Deep neural networks are robust to weight binarization and other non-linear distortions. *CoRR*, 2016.

[56] A. Mishra, K. Alahari, and C. Jawahar. Top-down and bottom-up cues for scene text recognition. In *CVPR*, 2012.

[57] M. Moczulski, M. Denil, J. Appleyard, and N. de Freitas. Acdc: A structured efficient linear layer. *ICLR*, 2016.

[58] L. Neumann and J. Matas. Real-time scene text localization and recognition. In *CVPR*, 2012.

[59] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov. Tensorizing neural networks. In *NIPS*, 2015.

[60] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

[61] D. Rolnick and M. Tegmark. The power of deeper networks for expressing natural functions. *arXiv preprint arXiv:1705.05502*, 2017.

[62] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *ICASSP*. IEEE, 2013.

[63] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. In *CVPR*, 2018.

[64] S. Scardapane, D. Comminiello, A. Hussain, and A. Uncini. Group sparse regularization for deep neural networks. *Neurocomputing*, 241:81–89, 2017.

[65] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.

[66] D. A. Spielman. Spectral graph theory and its applications. In *FOCS*, 2007.

[67] S. Srinivas and R. V. Babu. Data-free parameter pruning for deep neural networks. In *BMVC*, 2015.

[68] S. Srinivas, A. Subramanya, and R. V. Babu. Training sparse neural networks. In *CVPRW*, 2017.

[69] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 2017.

[70] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[71] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *CVPR*, 2014.

[72] W. Tang, G. Hua, and L. Wang. How to train a compact binary neural network with high accuracy? In *AAAI*, 2017.

[73] S. P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 2012.

[74] X. Wang, X. Duan, and X. Bai. Deep sketch feature for cross-domain image retrieval. *Neurocomputing*, 2016.

[75] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *NIPS*, pages 2074–2082. 2016.

[76] Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. Smola, L. Song, and Z. Wang. Deep fried convnets. In *CVPR*, pages 1476–1483, 2015.

[77] Q. You, H. Jin, Z. Wang, C. Fang, and J. Luo. Image captioning with semantic attention. In *CVPR*, 2016.

[78] Q. Yu, F. Liu, Y.-Z. Song, T. Xiang, T. M. Hospedales, and C.-C. Loy. Sketch me that shoe. In *CVPR*, 2016.

[79] Q. Yu, Y. Yang, F. Liu, Y.-Z. Song, T. Xiang, and T. M. Hospedales. Sketch-a-net: A deep neural network that beats humans. *International Journal of Computer Vision*, 2017.

[80] Q. Yu, Y. Yang, Y.-Z. Song, T. Xiang, and T. Hospedales. Sketch-a-net that beats humans. *BMVC*.

[81] X. Yu, T. Liu, X. Wang, and D. Tao. On compressing deep models by low rank and sparse decomposition. In *CVPR*, 2017.

[82] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018.

[83] Z. Zhong, J. Yan, and C.-L. Liu. Practical network blocks design with q-learning. *arXiv preprint arXiv:1708.05552*, 2017.

[84] H. Zhou, J. M. Alvarez, and F. Porikli. Less is more: Towards compact cnns. In *ECCV*, 2016.

[85] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *ICLR*, 2016.

[86] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017.