Distribution-Aware Binarization of Neural Networks for Sketch Recognition

by

Prabhu Ameya Pandurang, Vishal Batchu, Sri Aurobindo Munagala, Rohit Gajawada, Anoop Namboodiri

in

IEEE Winter Conference on Applications of Computer Vision 2018
(*WACV-2018*)

Lake Tahoe, NV, USA

Report No: IIIT/TR/2018/-1

Centre for Visual Information Technology
International Institute of Information Technology
Hyderabad - 500 032, INDIA
March 2018

Distribution-Aware Binarization of Neural Networks for Sketch Recognition

Ameya Prabhu Vishal Batchu Sri Aurobindo Munagala Rohit Gajawada Anoop Namboodiri
Center for Visual Information Technology, Kohli Center on Intelligent Systems
IIIT-Hyderabad, India
{ameya.prabhu@research., vishal.batchu@students., s.munagala@research.,
rohit.gajawada@students., anoop@}iiit.ac.in

Abstract

Deep neural networks are highly effective at a range of computational tasks. However, they tend to be computationally expensive, especially in vision-related problems, and also have large memory requirements. One of the most effective methods to achieve significant improvements in computational/spatial efficiency is to binarize the weights and activations in a network. However, naive binarization results in accuracy drops when applied to networks for most tasks. In this work, we present a highly generalized, distribution-aware approach to binarizing deep networks that allows us to retain the advantages of a binarized network, while reducing accuracy drops. We also develop efficient implementations for our proposed approach across different architectures. We present a theoretical analysis of the technique to show the effective representational power of the resulting layers, and explore the forms of data they model best. Experiments on popular datasets show that our technique offers better accuracies than naive binarization, while retaining the same benefits that binarization provides - with respect to run-time compression, reduction of computational costs, and power consumption.

1. Introduction

Deep learning models are pushing the state-of-the-art in various problems across domains, but 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 the amount of computation required to train a network leads to high power consumption as well.

There have been many developments in the area of model compression in the last few years, with the aim of bringing down network runtimes and storage requirements to mobile-friendly levels. Compression strategies for Convolutional Neural Networks included architectural improvements [16, 20] and re-parametrization [27, 34] to pruning techniques [14, 25] and quantization [19, 40]. Among these approaches, quantization - especially, binarization - provided the most compact models as shown in Table 1.

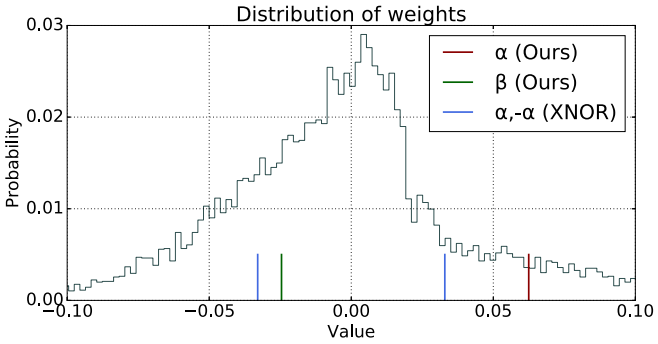

Figure 1: Weight distribution of a layer with corresponding α/β values, and the scaling factor α in the XNOR-Net implementation for comparison. α and β in our method have differing magnitudes, unlike in XNOR-Net.

Quantized networks - where weights/activations were quantized into low-precision representations - were found to achieve great model compression. Quantization has proven to be a powerful compression strategy, especially the most extreme form of quantization - Binarization. Binarization has enabled the use of XNOR-Popcount operations for vector dot products, which take much less time compared to full-precision Multiply-Accumulates (MACs), contributing to a huge speedup in convolutional layers [28, 19] 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 [28, 19] 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.

Later works have tended to move away from binary representations of weights/inputs to multi-bit representations. The reason for this was mainly the large accuracy degrada-

978-1-5386-4886-5/18/$31.00 ©2018 IEEE
DOI 10.1109/WACV.2018.00096

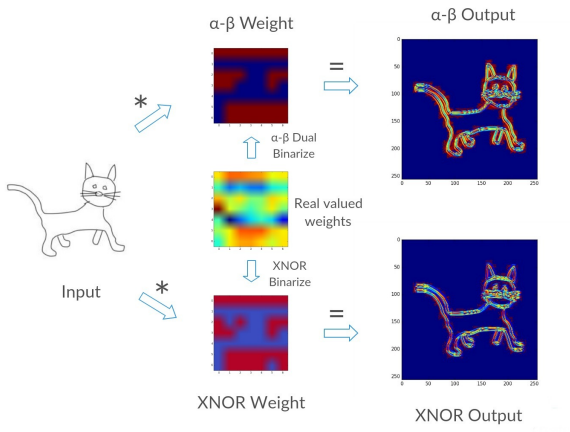

Figure 2: An example sketch passing through a convolutional layer filter, with the real-valued filter shown alongside corresponding α-β and XNOR-Net filters. Orange signifies the highest response areas. We can see that DAB-Net has significantly better responses when compared to XNOR-Net

tion observed in binary networks. While some works [32] have proposed methods to recover some of the lost accuracy, 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 optimally accurate binarization technique is worthwhile, due to the large gains in speedups (due to binary operations rather than full-prec MACs) and compression compared to multi-bit representations.

In our paper, we make the following contributions:

1. We show that binary representations are as expressive as full precision neural networks for polynomial functions, and offer theoretical insights into the same.
2. We present a generalized, distribution-aware representation for binary networks, and proceed to calculate the generalized parameter-values for any binary network.
3. We offer an intuitive analysis and comparison of our representation vis-a-vis previous representations, as illustrated in Figure 1.
4. We provide a provably efficient implementation of networks trained using this representation.
5. 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.

We also offer intuitions about how this technique might be effective in problems involving data that is inherently binary, such as sketches, as shown in Figure 2. Sketches are a universal form of communication and are easy to draw through mobile devices - thus emerging as a new paradigm

Method	Compression
Finetuned SVD 2 [34]	2.6x
Circulant CNN 2 [7]	3.6x
Adaptive Fastfood-16 [34]	3.7x
Collins *et al*. [8]	4x
Zhou *et al*. [39]	4.3x
ACDC [27]	6.3x
Network Pruning [14]	9.1x
Deep Compression [14]	9.1x
GreBdec [38]	10.2x
Srinivas *et al*. [30]	10.3x
Guo *et al*. [13]	17.9x
Binarization	**≈32x**

Table 1: Comparison of Binarization and other methods in terms of compression.

with interesting areas to explore, such as fast classification and sketch-based image retrieval.

Reproducibility: Our implementation can be found on GitHub [1]

2. Related Work

We ask the question: Do CNNs need the representational power of 32-bit floating point operations, especially for binary-valued data such as sketches? Is it possible to cut down memory costs and make output computations significantly less expensive? In recent years, several different approaches were proposed to achieve network compression and speedups, and special-purpose networks were proposed for sketch classification/retrieval tasks. These are summarized below:

Sketch Recognition: Many deep-network based works in the past did not lead to fruitful results before, primarily due to these networks being better suited for images rather than sketches. Sketches have significantly different characteristics as compared to images, and require specialized, fine-tuned networks to work with. Sketch-a-Net from Yu *et al*. [37] took these factors into account, and proposed a carefully designed network structure that suited sketch representations. Their single-model showed tremendous increments over the then state-of-the-art, and managed to beat the average human performance using a Bayesian Fusion ensemble. Being a significant achievement in this problem - since beating human accuracy in recognition problems is difficult - this model has been adopted by a number of later works Bui *et al*. [4], Yu *et al*. [35], Wang *et al*. [33].

Pruning Networks for Compression: Optimal Brain Damage [10] and Optimal Brain Surgeon [15] introduced a network pruning technique based on the Hessian of the

[1] https://github.com/erilyth/DistributionAwareBinarizedNetworks-WACV18

loss function. Deep Compression [14] also used pruning to achieve compression by an order of magnitude in various standard neural networks. It further reduced non-runtime memory by employing trained quantization and Huffman coding. Network Slimming [25] introduced a new learning scheme for CNNs that leverages channel-level sparsity in networks, and showed compression and speedup without accuracy degradation, with decreased run-time memory footprint as well. We train our binary models from scratch, as opposed to using pre-trained networks as in the above approaches.

Higher Bit Quantization: HashedNets [6] hashed network weights to bin them. 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 [41] uses 2-bit weights and scaling factors to bring down model size to 16x compression, with little accuracy degradation. Quantized Neural Networks[19] use low-precision quantized weights and inputs and replaces arithmetic operations with bit-wise ones, reducing power consumption. DoReFa-Net [40] used low bit-width gradients during backpropagation, and obtained train-time speedups. Ternary Weight Networks [22] optimize a threshold-based ternary function for approximation, with stronger expressive abilities than binary networks. The above works cannot leverage the speedups gained by XNOR/Pop-count operations which could be performed on dedicated hardware, unlike in our work. This is our primary motivation for attempting to improve binary algorithms.

Binarization: We provide an optimal method for calculating binary weights, and we show that all of the above binarization techniques were special cases of our method, with less accurate approximations. Previous binarization papers performed binarization independent of the distribution weights, for example [28]. The method we introduce is distribution-aware, i.e. looks at the distribution of weights to calculate an optimal binarization.

BinaryConnect [9] was one of the first works to use binary (+1, -1) values for network parameters, achieving significant compression. XNOR-Nets [28] followed the work of BNNs [18], binarizing both layer weights and inputs and multiplying them with scaling constants - bringing significant speedups by using faster XNOR-Popcount operations to calculate convolutional outputs. Recent research proposed a variety of additional methods - including novel activation functions [5], alternative layers [32], approximation algorithms [17], fixed point bit-width allocations [23]. Merolla et al. [26] and Anderson et al. [1] offer a few theoretical insights and analysis into binary networks. Further works have extended this in various directions, including using local binary patterns [21] and lookup-based compression methods [3].

3. Representational Power of Binary Networks

Many recent works in network compression involve higher bit weight quantization using two or more bits [2, 41, 22] 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*. [24, 29] 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 [11], 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 [24]. 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*

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

Proof of the above can be found in the supplementary material.

Conjecture III.2. of Rolnick *et al*. [29] says that this bound is approximately optimal. If this conjecture proves to be true, weight-binarized networks would have the same

representational power as full-precision networks, since the network that was essentially used to prove that the above theorem - that a network exists that can satisfy that bound - was a binary network.

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.

4. 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*.

4.1. 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:

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

A detailed proof of the above can be found in the supplementary material.

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.

4.2. 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 [28] or BNN [18]-like representations cannot be obtained from our formulation. However, in practice, we are able to simulate XNOR-Net by constraining $\mathbf{W}$ to be mean-centered and $K = \frac{n}{2}$, since roughly half the weights are above 0, the other half below, as seen in Figure 5 in Section 5.3.2.

Algorithm 1 Finding an optimal K value.

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

for I= 1 to D.size **do**
    $P_i = P_{i-1} + \mathbf{W}_i$
    $D_i = \frac{P_i^2}{i} + \frac{(T-P_i)^2}{n-i}$
    if $D_i \geq maxD_1$ **then**
        $maxD_1$ = D_i
        $optK_1$ = i

Sort($\mathbf{W}$, 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 $optK_{final}$

4.3. 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.

4.3.1 Parallel Prefix-Sums to Obtain K

Theorem 2 *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 1. This algorithm is integrated into our code, and will be provided alongside.

4.3.2 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 3 (Backward Pass) *The optimal gradient value $\widetilde{G}$ can be represented as,*

$$\widetilde{\mathbf{G}} = \widetilde{\mathbf{G_1}} + \widetilde{\mathbf{G_2}} \tag{2}$$

where,

$$\widetilde{\mathbf{G_1}} = \frac{sgn(\mathbf{T_k})}{K} \circ sgn(\mathbf{T_k}) + \frac{||\mathbf{T_k}||_{l1}}{K}.STE(\mathbf{T_k}) \tag{3}$$

$$\begin{aligned}\widetilde{\mathbf{G_2}} = &\frac{sgn(\mathbf{W} - \mathbf{T_k})}{n - K} \circ (1 - sgn(\mathbf{T_k})) \\ &+ \frac{||\mathbf{W} - \mathbf{T_k}||_{l1}}{n - K}.STE(\mathbf{W} - \mathbf{T_k})\end{aligned} \tag{4}$$

$$STE(\mathbf{T_k})^i = \begin{cases} {\mathbf{T_k}}^i, where \; |\mathbf{W}|^i <= 1 \\ 0, \; elsewhere \end{cases} \tag{5}$$

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.

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 1
        $\alpha_{lj} = \frac{topk(\mathbf{W}_{lj}, K_{lj})}{K_{lj}}$
        $\beta_{lj} = -\frac{(\mathbf{W}_{lj} - topk(\mathbf{W}_{lj}, K_{lj}))}{n - K_{lj}}$
        $\widetilde{\mathbf{W}}_{lj} = \alpha.sgn(topk(\mathbf{W}_{lj}, K_{lj}))$
            $+ \beta.(1 - sgn(topk(\mathbf{W}_{lj}, K_{lj})))$

$\hat{\mathbf{Y}} =$ **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. 3

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)

4.4. 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.

5. 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 [28] 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.

5.1. 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 [12] 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[37], ResNet-18[16] and GoogleNet[31] 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

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

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.

5.2. 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 2. On the TU-Berlin and Sketchy datasets in Table 2, 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 3. We find that our compressed models 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.

[2]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[2][36]	**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: A comparison between state-of-the-art single model accuracies of recognition systems on the TU-Berlin dataset.

5.3. 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).

5.3.1 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, 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.

5.3.2 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 4.

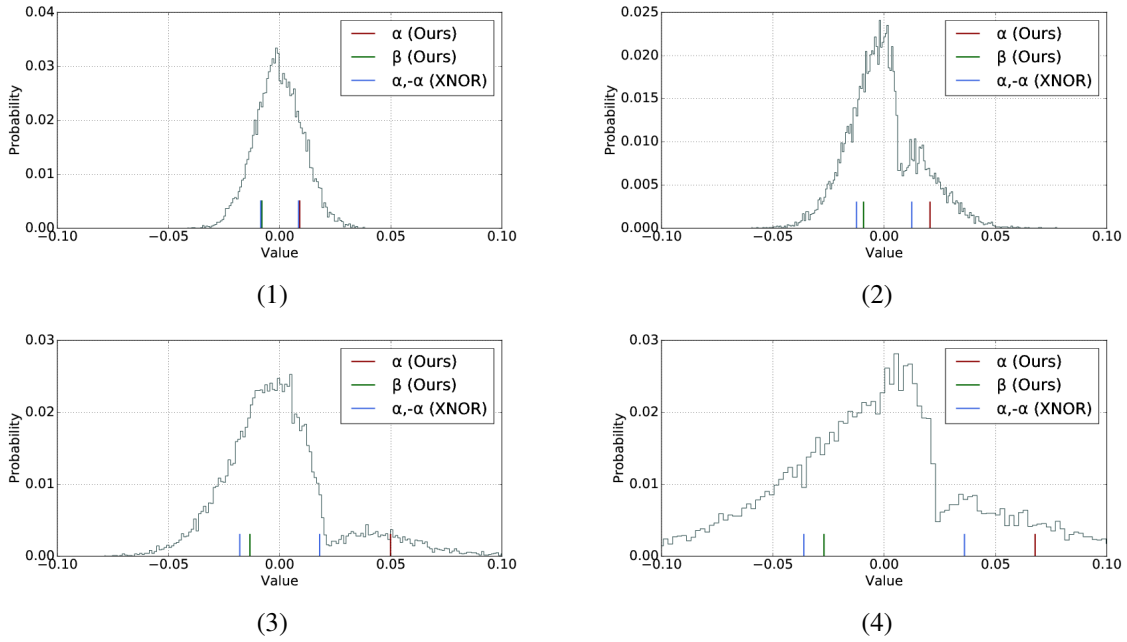

Figure 3: Sub-figures (1) to (4) show the train-time variation of α and β for a layer filter. Initially, α and β have nearly equal magnitudes, similar to the XNOR-Net formulation, but as we progress to (4), we see that α and β have widely different magnitudes.Having just one scaling constant (XNOR-Net) would be a comparatively poor approximator.

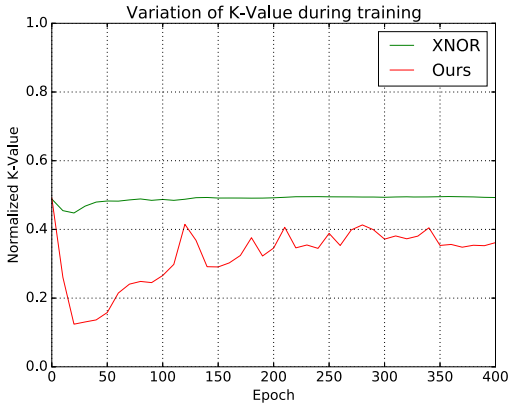

Figure 4: The variation of the normalized K-value over time during training. It falls initially but converges eventually to 0.35. The normalized K-value for XNOR-Net remains almost at 0.5 till the end.

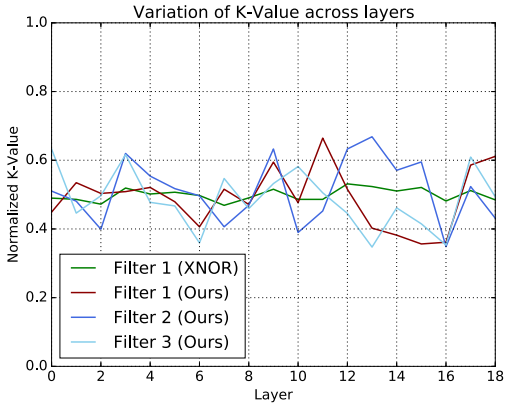

Figure 5: The variation of normalized K values on random filters across layers. The K-value corresponding to DAB-Net varies across layers based on the distribution of weights of the specific layer, which is not captured by XNOR-Net.

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 5.

6. Conclusion

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. We believe that this work provides a new perspective on network binarization, and that future work can gain significantly from distribution-aware explorations.

References

[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. *ICLR*, 2017.

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

[4] 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.

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

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

[7] 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.

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

[9] M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*, pages 3123–3131, 2015.

[10] Y. L. Cunn, J. S. Denker, and S. A. Solla. Nips. chapter Optimal Brain Damage. 1990.

[11] G. Cybenko. Approximation by superpositions of a sigmoidal function. *(MCSS)*, 2(4):303–314, 1989.

[12] M. Eitz, J. Hays, and M. Alexa. How do humans sketch objects? *ACM Trans. Graph. (Proc. SIGGRAPH)*, 31(4):44:1–44:10, 2012.

[13] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. In *NIPS*, pages 1379–1387, 2016.

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

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

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

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

[18] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. 2016.

[19] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv preprint arXiv:1609.07061*, 2016.

[20] 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.

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

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

[23] 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.

[24] H. W. Lin, M. Tegmark, and D. Rolnick. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168(6):1223–1247, 2017.

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

[26] 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.

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

[28] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, pages 525–542, 2016.

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

[30] S. Srinivas, A. Subramanya, and R. V. Babu. Training sparse neural networks. In *CVPRW*, pages 455–462, 2017.

[31] 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.

[32] W. Tang, G. Hua, and L. Wang. How to train a compact binary neural network with high accuracy? In *AAAI*, pages 2625–2631, 2017.

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

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

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

[36] 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*, 122(3):411–425, 2017.

[37] Q. Yu, Y. Yang, Y.-Z. Song, T. Xiang, and T. Hospedales. Sketch-a-net that beats humans. *BMVC*, 2015.

[38] X. Yu, T. Liu, X. Wang, and D. Tao. On compressing deep models by low rank and sparse decomposition. In *CVPR*, pages 7370–7379, 2017.

[39] H. Zhou, J. M. Alvarez, and F. Porikli. Less is more: Towards compact cnns. In *ECCV*, pages 662–677, 2016.

[40] 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.

[41] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. *ICLR*, 2017.