EE 367 Report: A Survey of Gradient Estimators for Binary Neural Networks for Image Classification

Haley So

Abstract—The emergence of new sensors that provide the capability for on sensor computation opens doors to new imaging algorithms. Some of the most common tasks in the imaging and computer vision world have been tackled using convolutional neural networks. However, large model sizes make it impossible to run on the sensors with limited memory. Binary neural networks offer a promising approach to compressing networks to sizes that allow the weights to be fully stored on the sensor. In this study, we look into one of the most challenging problems in training binary neural networks, how to deal with the gradient during backpropagation. We present comparisons and studies changing different parameters such as gradient estimators, kernel size, layer numbers, and network depth to try to train quantized networks with comparable performance. However, there is still a large gap between the full precision networks and binary networks. More research into gradient estimation and architecture search is needed to make running CNNs on these sensors a reality.

Index Terms—Computational Photography

1 Introduction

Convolutional neural networks (CNNs) have become ubiquitous in many computational imaging tasks. However, these can take ample time to run and large amounts of memory to store the models. For example, VGG-16 requires over 500 MegaBytes of memory to store the weights. Recently, with the advancement of 3D fabrication and wafer level bonding, there's been an emergence of sensors, such as Sony's new stacked CMOS sensor with 2-layer transistor pixels, with pixels that have nearly 100% fill factor and in per pixel circuitry. Together this allows for computation to be pushed onto the focal plane. This opens doors for new clever imaging algorithms. On sensor computation also allows for us to extract only the salient information, giving not only speedups up, but also to tackle the limited bandwidth problem when it comes to transferring data. However, with a limited memory capacity, such state-of-the-art CNNs would not be able to be stored on the sensor. Network compression becomes a crucial problem.

In the past few years, researchers have started looking into binary networks or highly quantized networks to achieve tasks such as image classification or object detection. Through binarization, each weight would be represented with a single bit, as opposed to the full 32-bit floating point precision case. The main challenges that come from training quantized networks is that with non-differentiable functions such as the sign function that would binarize weights to -1 and 1, or the rounding function for more bits, back propagation becomes difficult. One way to approximate the gradient is through various gradient estimation techniques. In this work, we perform a survey of gradient estimators, training a simple architecture for image classification on the CIFAR-10 image dataset [1]. We want to distill the effect of the estimators and see trade-offs between memory footprint and classification precision.

• *H. So is with the Department of Electrical Engineering, Stanford University, Stanford, CA, 94305.*
E-mail: haleyso@stanford.edu

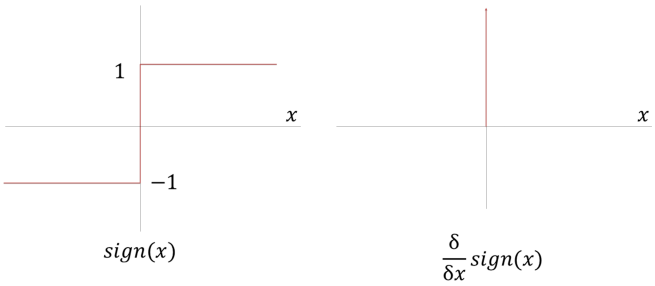

Fig. 1. The sign function binarizes our values to -1 or 1. However, it is non differentiable. Its derivative is 0 or undefined everywhere.

Fig. 2. Examples of CIFAR10 dataset images [1] we train and test on.

2 Related Work

Network compression techniques fall into roughly five categories: 1. parameter reduction by pruning redundancy, 2. low rank parameter factorization, 3. carefully designing structured convolutional filters, 4. creating smaller models, and 5. parameter quantization, which is the technique we work with here. The most extreme form of parameter quantization is by binarizing networks. Naive approaches quantize weights after training, which gives limited performance.

BinaryConnect by Courbariaux et. al. [2] was one of the first to train a deep neural network (DNN) using binary weights. They quantize in the forward pass, and during backpropagation, they use the straight-through-estimator (STE) which will be detailed in the Methods section. Here they keep full precision activations while the weights are binary. With binary weights and 32-bit precision activations, there is no longer a need for full precision multiplications, just full precision accumulation. Binary Neural Networks (BNNs) either quantize just the weights or both weights and activations, the latter of which is more popular since it allows for more compression. Pioneering followup works include Binarized Neural Networks (BNNs) [3] and XNOR-net [4]. XNOR-net presented the importance of binary encoding. Using -1 and 1, as opposed to 0 and 1 allowed for turning convolutions into XNOR and bit-counting operations. Turning networks binary this way not only uses 32 times less memory, but computational, there is a 58 times speedup since there is no longer the need for heavy floating-point multiplication and addition. There have been studies on the rate of oscillation between -1 and 1 [5] as the learning rate changes, the introduction of a gain term [6], using multiple bases and a learned bias for thresholding [7], and looking into alternatives to STE [8], [9], [10]. In addition, in the past few years, many bodies of work in the BNN and quantized neural network (QNN) space have been coming out [11], [12], [13], [14], [15], and [16].

3 METHOD

As suggested by [4], we use the -1/1 encoding. The quantization function in this work follows that of the predecessors.

$$x_b = sign(x) \tag{1}$$

where x_b is the binarized value. We do a study of differing levels of compression, a) binarizing solely the weights, and b) binarizing both weights and activations. The binarization function 1 is non differentiable. Everywhere, the gradient is 0 except for at 0, where the gradient is undefined. The sign function and gradient are in figure 1. To enable training, we must deal with the non-differentiability carefully. To perform this study, we use a base network architecture and swap in gradient estimators to try to find the best approximation. The pipeline is shown in figure 3.

3.1 Base Network Architecture

In the base network architecture, shown in figure 5, we have a simple 2 convolutional layer and 3 linear layers network with max pooling layers after each convolutional layer to perform image classification on the CIFAR-10 image dataset [1]. A sample of CIFAR images is shown in figure 2. The details of the model architecture are shown in figure 3. In each comparison, we write a custom convolutional layer module that uses the sign function 1 in the forward pass and we implement one of the gradient estimators in the backward pass. In the CIFAR-10 dataset, there are 10 categories of images, so we train with a Cross Entropy Loss for 300 epochs using the SGD optimizer in Pytorch, a momentum of 0.9 and weight decay of 1e-4. The starting learning rate for each model is tuned for each model for best performance, but a cosine annealing of the learning rate is applied to the training. We report the top-1 precision, meaning whether our network correctly predicted the correct class. For a baseline, we train the full precision model as well.

3.2 Gradient Estimators

Straight-Through Estimator (STE): The straight through estimator essentially ignores the non-differentiability by bypassing the binarization. The gradient through the binarization is set to the identity function.

$$\frac{\delta sign(x)}{\delta x} = \mathbb{I} \tag{2}$$

In some cases, some variants only propagate the gradient if the magnitude is less than 1.

$$\frac{\delta sign(x)}{\delta x} \approx \begin{cases} 1 & |x| \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

The gradients may get large, so sometimes the hard tanh function is used to keep the gradients in a reasonable range. Here, we simply send the incoming gradient to the outgoing gradient via 3.

Second Order Approximation: In BiRealNet [10], Liu et. al used a second order approximation of the sign function given by

$$sign(x) \approx \begin{cases} -1 & x < -1 \\ 2x + x^2 & -1 \leq x < 0 \\ 2x - x^2 & 0 \leq x \leq 1 \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

The corresponding gradient estimator is then the following piecewise function.

$$\frac{\delta sign(x)}{\delta x} \approx \begin{cases} 2 + 2x & -1 \leq x < 0 \\ 2 - 2x & 0 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Using tanh(x) to approximate the sign function: There are other works that approximate the sign function with a differentiable function like the tanh function. The corresponding gradient estimator is

$$\frac{\delta sign(x)}{\delta x} \approx \frac{\delta tanh(x)}{\delta x} = 1 - \tanh^2 x \tag{6}$$

Other approximations of the gradient: [8] and others have looked to try to make a smoother delta function to approximate the shape of the gradient of the sign function.

$$\frac{\delta sign(x)}{\delta x} \approx \left(\frac{2}{\cosh x} \right)^2 \tag{7}$$

In all cases, each model was trained using the same training procedure, with the exception of some hyperparamter tuning. The sign function approximations and corresponding gradient estimators are shown in figure 4.

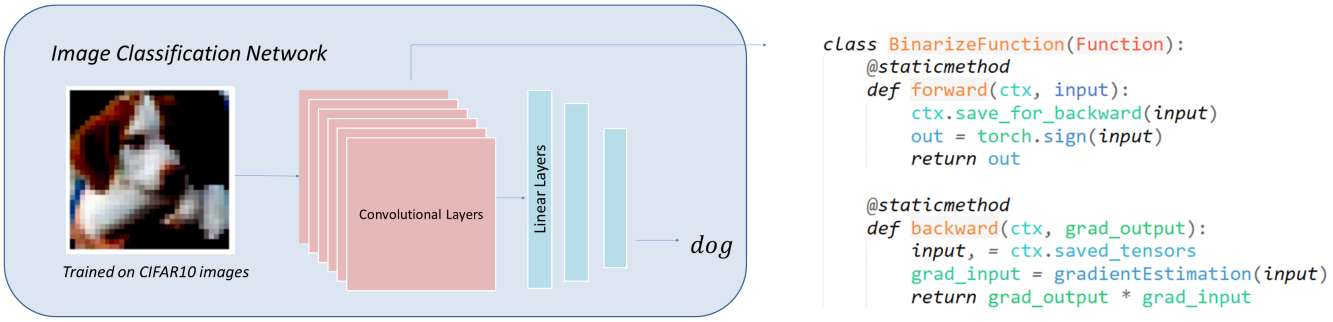

Fig. 3. **Pipeline:** The overview of the pipeline for image classification with the modified convolutional layers. An image is inputted into the network where the forward pass is the sign function and the backward pass is a gradient estimation function. The output of the network is a prediction of what the label of the image is.

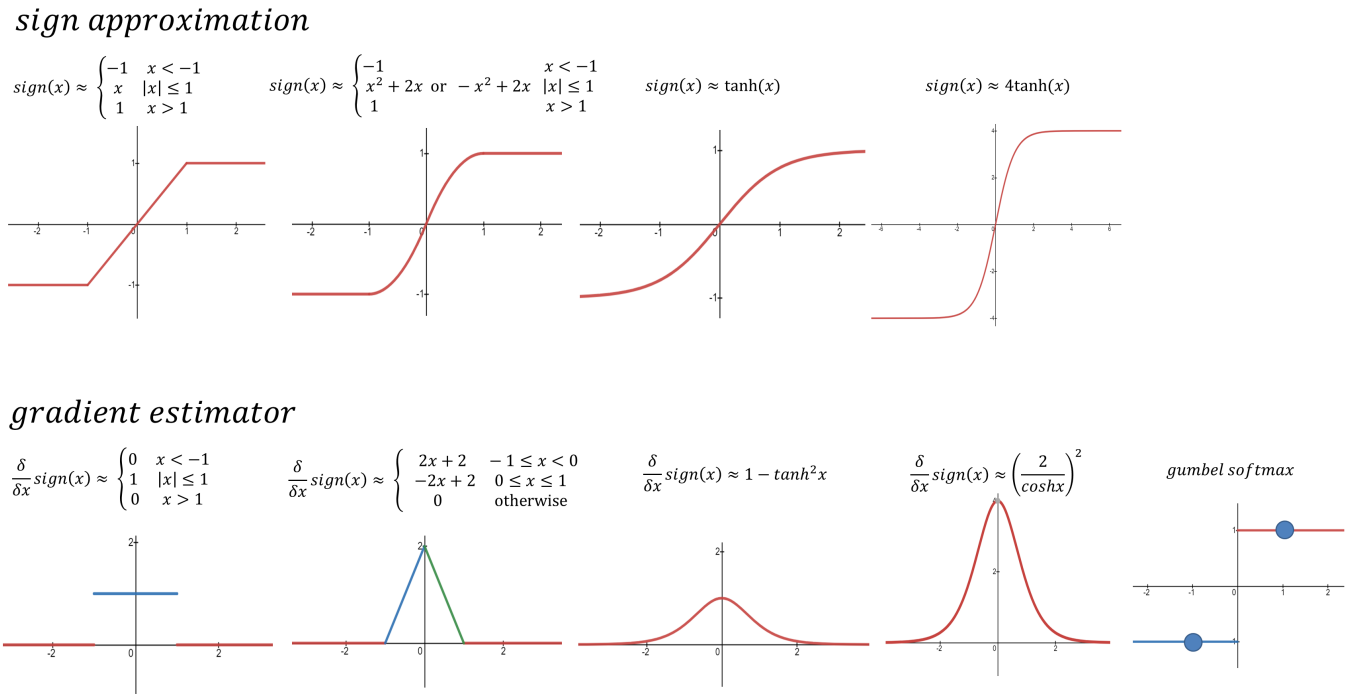

Fig. 4. **Signs functions and gradient functions:** On the top row, we show the approximation of the sign function with the corresponding gradient estimators. In the first three cases, the gradient estimator comes from taking the derivative of the sign approximation. In the fourth case, the gradient estimator approximates the gradient of the sign function. We show the corresponding sign function approximation for completeness. The final method to binarize is through gumbel softmax, which treats -1 and 1 as categorical variables.

3.3 Memory Footprint and Precision Tradeoff

A goal for this study was also to see what insights we could draw from increasing the depth of the binary networks to the same memory footprint as the full precision network. To calculate the memory footprint, we followed the same convention as previous BNN works. In the base network, using 2 full precision convolutional layers with input-channels/output-channels/kernel-size of 3/6/5 and 6/16/5, the total parameters without bias totaled to 2,850 parameters. Each parameter uses 32 bits, which gives a total of 91,200 bits. On the other hand, the fully binary network uses 1 bit per value, giving us 2,850 bits. This leaves us with 88,350 bits to add depth to our architecture. As reference, 6/6/5 convolutional layers has 900 bits, so we could add up to 98 of these layers to the binary network.

3.4 Additional studies with complex architectures

We train several models with differing depths and kernel sizes. Results are shown in table 2. In the base model, we only have a few filters per layer. Simply adding more layers may not be an effective way to increase precision. In the deep learning community, architecture type plays an important role in increasing precision. For an additional study, we train a model from IR-Net [9] from scratch and do comparisons between the gradient estimators on the more complex architecture. The network architecture is shown in figure 6. Some hyperparamter tuning was performed on each model, but the training procedure stays the same.

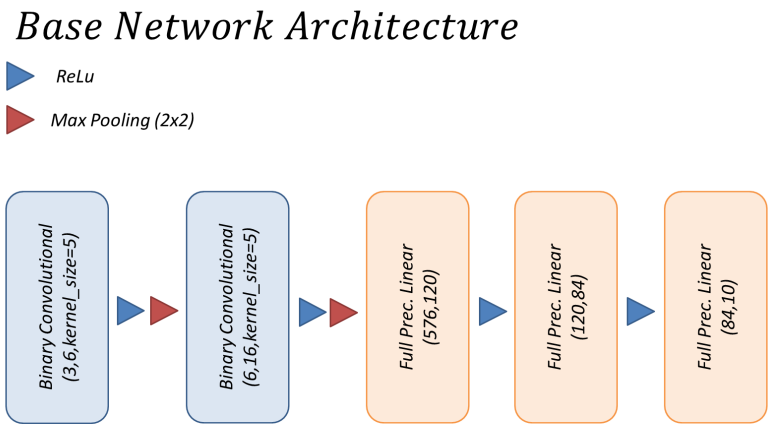

Fig. 5. Above is the base network architecture used in the first set of experiments. Here, we swap in each of the modified binary convolutional layers with their corresponding gradient estimators. The linear layers are left at full precision.

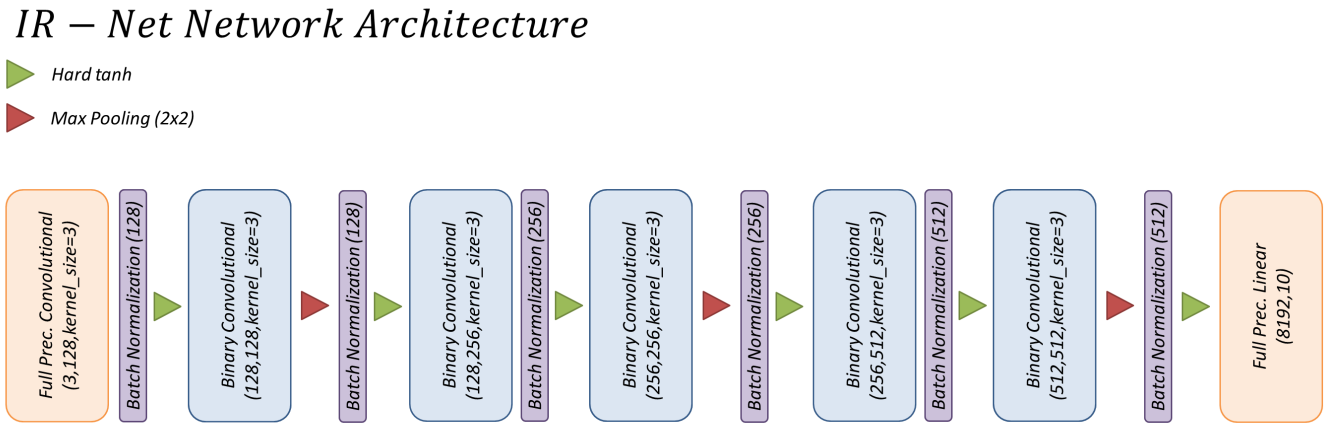

Fig. 6. We take a model from IR-Net [9], swapping the gradient estimators in the binary convolutional layers. This architecture uses over 4 million parameters.

The model from IR-Net's paper we use consists of 6 convolutional layers with input-channels/output-channels/kernel-sizes of 3/128/3, 128/128/3, 128/256/3, 256/256/3, 256/512/3, and 512/512/3 with batch normalization and a hard tanh nonlinearity applied after each convolutional layer. The final layer is a single linear layer. Note, the first convolutional layer and the last linear layer are not binarized. The number of filters is much larger than the base model we previously used, allowing for more feature extraction. However, even with most of the weights and activations fully quantized, the model requires over 4 million bits, which depending on the sensor, may be difficult to use. We still perform the study to compare the effectiveness of the estimator.

4 Experimental Results and Discussion

The three sets of experiments are evaluated on the precision of predicting the correct label.

Experiments on the base model: Table 1 shows the results for the experiments on the base model. While the gradient estimators achieved performance better than the naive approach of just quantizing the weights after the network is fully trained, there is still a large gap between the full precision network and the binarized networks. The full precision model had a 75.22% accuracy, while the top estimators, the STE and tanh estimator, applied to the weights performed with just under 60% accuracy. With additional quantization of the activations, the performance had an additional noticeable drop. Interestingly, the second order estimator actually had a better performance when both the weights and the activations were binarized. This may indicate more hyper parameter tuning is needed, or that in this space, we may have gotten stuck in a local minima.

Experiments on increasing memory footprint: Several models were created in this study to try to understand the effects of each parameter. Number of filters, kernel sizes, number of convolutional layers, etc. parameters were

TABLE 1
Experiment 1: Interchanging gradient estimators on the base model
Reported percentages are the best trained with parameter tuning. IR-Net, BiRealNet ResNet50 were also trained with 87.6 and 83.9 accuracy but they had very specific architectures and training procedures. This table only has comparable architectures to show the effect of the gradient estimator.

Gradient Estimator/Model	Weight/Activation (Bits)	Convolutional Layers Memory Size	Precision Top-1 %
None (Full Precision) / Base Model (BM)	32/32	91,200 bits	75.22
Naïve: Quantized Weights at the End / BM	1/32	2,850 bits	11.86
Straight Through Estimator (STE) / BM	1/32	2,850 bits	**59.98**
Straight Through Estimator (STE) / BM	1/1	2,850 bits	53.00
Second Order Approximation / BM	1/32	2,850 bits	37.33
Second Order Approximation / BM	1/1	2,850 bits	52.22
Tanh estimator / BM	1/32	2,850 bits	**59.44**
Tanh estimator / BM	1/1	2,850 bits	51.46
2/coshx estimator / BM	1/32	2,850 bits	**58.10**
2/coshx estimator / BM	1/1	2,850 bits	51.23
Gumbel Softmax /BM	1/32	2,850 bits	**58.05**
Gumbel Softmax /BM	1/1	2,850 bits	34.71

TABLE 2
Experiment 2: Changing Memory Footprint

Gradient Estimator (and changes)	Weight/Activation (Bits)	Convolutional Layers Memory size	Precision Top-1 %
STE(+ 50 conv2d layers)	1/32	47,850 bits	10.00
STE (+10 conv2d layers)	1/32	7,350 bits	9.99
STE (+5 conv2d layers)	1/32	1,026 bits	9.98
STE (+1 conv2d layer)	1/32	3,750 bits	32.72
STE (+1 conv2d layer, kernel size=3)	1/32	1,350 bits	28.53
STE (kernel size=3)	1/32	1,026 bits	54.17
STE (Increase output channels of covn2d to over 256)	1/32	414,400 bits	56.89

TABLE 3
Experiment 3: IR-Net architecture with different gradient estimates

Gradient Estimator	Weight/Activation (Bits)	Precision Top-1 %
IR-Net using the original decaying tanh estimator	1/1	**87.89**
IR-Net with static tanh estimator	1/1	77.23
IR-Net with STE	1/1	78.34
IR-Net with gumbel softmax	1/1	10.11
IR-Net with gumbel softmax	1/32	17.31
IR-Net with 2/coshx estimator	1/1	**88.25**
IR-Net with second order estimator	1/1	**88.55**

changed to see if there would be a direct relationship between memory footprint and precision. See table 2 for results. Experiments included adding 1 to 50 more 6/6/5 convolutional layers, and results showed none were able to reach good performance. The models used the STE method, which performed best in the previous study. Reported are the best precisions done, though with deeper networks, hyperparameter tuning becomes increasingly important and difficult. Here, it seemed we reached a local minima and across the board achieved about 10% accuracy, which given the 10 categories, is the same as guessing. More work needs to be done in this space. Changing kernel sizes to 3x3 instead of 5x5 gave comparable performance, but then with the addition of just one more convolutional layer, the accuracy dropped over 20%. Another experiment on the number of filters, as opposed to changing the kernel size, was done. Going from 3/6/5 and 6/16/5 to 3/64/5 and 64/256/5 gave comparable performance as well, but with a larger memory footprint. With no definitive improvement in tweaking these parameters, we thought it may be the case that architecture together with the gradient estimator and the training procedure play an important role, which leads us to the third round of experiments.

Tailored architecture: Table 3 shows the results of the final set of experiments. We kept the architecture detailed in IR-Net and in the methods section. For each model, we changed the gradient estimator. The accuracies across the board are near 75-90%, with the exception of gumbel-softmax.

Although trained with hyper paramter tuning, the 1 bit weight and 1 bit activation version always got stuck around 10% while the 1 bit weight and full precision activation reached 17%. As shown in the first set of experiments, gumbel-softmax can achieve numbers near those of other methods. But, gumbel-softmax is more finicky to train and often takes many more iterations of parameter tuning. Here, we see the architecture does play a large part in the accuracy. However, as mentioned before, this model requires orders of magnitude more memory. This leads to two main possibilities: 1. continue looking for a better gradient estimator, or 2. focus on architecture search for compressing network architectures.

5 Conclusion, Limitations, & Future Work

Binary Neural Networks are essential for running on the edge or on sensors. Here, we study different gradient estimators in hopes to train effective BNNs that are low in memory. From the three sets of experiments, we conclude these gradient estimators alone are not enough to obtain good precision. Additional architecture search or specialized training procedures were used which helped the accuracy at the cost of memory footprint. **Limitations:** We were limited in the number of gradient estimators we tested since each model, with hyperparameter tuning and training for 300 epochs, was time intensive. More gradient estimators should be tested for a thorough survey of all the methods out there. The network architectures were also limited. Here, we used the CIFAR10 dataset, but changing the task or the number of possible categories will require new training and new models. **Future works** include doing exactly this and testing out every estimator or creating new functions that look like a delta function of sorts to estimate the gradient. Also, performing this architecture search in tandem with improving gradient estimators to get light weight CNNs with performances closer to the full precision networks, or looking into other compression mechanisms would be a good step to further the study.

Acknowledgments

The author would like to thank Professor Gordon Wetzstein, David Lindell, and Mark Nishimura for a great EE367 course!

References

[1] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[2] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," *arXiv e-prints*, p. arXiv:1511.00363, Nov. 2015.

[3] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16. Red Hook, NY, USA: Curran Associates Inc., 2016, p. 4114–4122.

[4] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 525–542.

[5] W. Tang, G. Hua, and L. Wang, "How to train a compact binary neural network with high accuracy?" in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ser. AAAI'17. AAAI Press, 2017, p. 2625–2631.

[6] 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," *arXiv e-prints*, p. arXiv:1606.06160, Jun. 2016.

[7] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 344–352.

[8] J. Kim, "Binarized neural network," https://github.com/penpaperkeycode/Binarized_Neural_Network_Pytorch, 2020.

[9] H. Qin, R. Gong, X. Liu, M. Shen, Z. Wei, F. Yu, and J. Song, "Forward and backward information retention for accurate binary neural networks," in *IEEE CVPR*, 2020.

[10] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K.-T. Cheng, "Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm," in *ECCV*, 2018.

[11] Q. Jin, J. Ren, R. Zhuang, S. Hanumante, Z. Li, Z. Chen, Y. Wang, K. Yang, and S. Tulyakov, "F8net: Fixed-point 8-bit only multiplication for network quantization," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=_CfpJazzXT2

[12] T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer, "8-bit optimizers via block-wise quantization," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=shpkpVXzo3h

[13] S. Lee, J. Park, and D. Jeon, "Toward efficient low-precision training: Data format optimization and hysteresis quantization," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=3HJOA-1hb0e

[14] H. Qin, Y. Ding, M. Zhang, Q. YAN, A. Liu, Q. Dang, Z. Liu, and X. Liu, "BiBERT: Accurate fully binarized BERT," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=5xEgrl_5FAJ

[15] M. Bahri, G. Bahl, and S. Zafeiriou1, "Binary graph neural networks," in *CVPR*, 2021.

[16] H. Qin, Z. Cai, M. Zhang, Y. Ding, H. Zhao, S. Yi, X. Liu, and H. Su, "Bipointnet: Binary neural network for point clouds," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=9QLRCVysdlO