2018 15th International Conference on
Control, Automation, Robotics and Vision (ICARCV)
Singapore, November 18-21, 2018

QL-Net: Quantized-by-LookUp CNN

Kamila Abdiyeva, Kim-Hui Yap, Wang Gang, Narendra Ahuja and Martin Lukac

Abstract— **Convolutional Neural Networks (CNNs) have achieved a state-of-the-art performance in the different computer vision tasks. However, CNN algorithms are computationally and power intensive, which makes them difficult to run on wearable and embedded systems. One way to address this constraint is to reduce the number of computational operations performed. Recently, several approaches addressed the problem of the computational complexity in the CNNs. Most of these methods, however, require a dedicated hardware. We propose a new method for the computation reduction in CNNs that substitutes Multiply and Accumulate (MAC) operations with a codebook lookup and can be executed on the generic hardware. The proposed method called QL-Net combines several concepts: (i) a codebook construction, (ii) a layer-wise retraining strategy, and (iii) a substitution of the MAC operations with the lookup of the convolution responses at inference time. The proposed QL-Net achieves a 98.6% accuracy on the MNIST dataset with a 5.8x reduction in runtime, when compared to MAC-based CNN model that achieved a 99.2% accuracy.**

I. Introduction

While Convolutional Neural Networks (CNNs) have achieved the state-of-the-art performance in different computer vision tasks, such as classification [1]–[3], detection [4], [5] and semantic segmentation [6]–[8], they suffer from the excessive computational cost and the power consumption, making them challenging to use without a powerful Graphics Processing Unit (GPU). Additionally, with new intelligent devices appearing every day, the systems without such powerful GPUs are very common. Thus, in order to allow small and embedded devices to use the CNN technology, their power consumption, and the computational complexity should be reduced.

To overcome this issue, several optimization methods have been developed [9]–[13]. The early approaches to optimize networks involved restricting weights to be a number of the form 2^n, where n is an integer. This allows substitution of all the multiplications by binary shifts [14]. In [15], [16] the authors proposed the CNN computation reduction through reducing the precision of parameters. [9] proposed a stochastic binarization of the network weights and replacing the multiply and accumulate (MAC) operation used in convolution by the sign changes. Further acceleration was achieved by binarizing both the network's parameters and the feature maps [11]. In XNOR-Net [10] the binary parameters were scaled with a scaling factor to increase the network capacity, while still being able to exploit faster bitwise operations (XOR or AND) instead of the conventional MAC.

While reduced-precision networks minimize the number of computations and increase the inference speed, they require a dedicated hardware. The current CPU and GPU are not well suited for the bit-wise operations and resources are wasted because the full power of these computational devices is not exploited. For optimal use, the binarized CNNs require a low power computation hardware with a limited reconfigurability. However, the design cost of such specific hardware is much higher than of a standard high-speed FPGA or CPUs. Hence, it is challenging to minimize the CNN computational requirements without introducing hardware restrictions.

To address the problem of reducing the required number of operations while preserving the task accuracy and not imposing extra hardware limitations, we propose a novel framework, called QL-Net. The closest to our work is LCNN [17], in which the acceleration is achieved by approximating 3×3 filters with jointly learned linear combination of 1×1 dictionary vectors. Such parameter sharing technique allows to reduce both a storage capacity and the amount of operations, however, the MAC operations should still be performed. In contrast to this approach, we do not approximate network parameters, but rather the activation maps using an L1-distance. QL-Net integrates three main concepts: (i) a dictionary construction to approximate patches of CNN feature maps, (ii) a layer-wise retraining strategy to minimize the approximation error, and (iii) performing dictionary lookup instead of convolutional MAC operations to reduce the number of operations during the inference.

QL-Net takes an image as an input and extracts the feature maps, similar to the conventional CNN. Unlike the conventional CNN, QL-Net extracts feature maps by look-up operations rather than by convolution. In conventional CNN, when a convolutional filter is applied to the layer patch, MAC operations between the filter and the input patch are performed to calculate the filter response. When multiple filters are applied, responses of all filters are concatenated together. In contrast, QL-Net approximates the convolution between a layer input patch and a set of filters by a dictionary look-up using the patch as the key in the Look-Up-Table (LUT). By implementing a hierarchical tree as a LUT, we are able to find the nearest neighbor in a smaller amount of computations than it would take to perform the convolutional MAC operations on a set of filters. In addition, unlike in the previous methods, QL-Net can be used on a general-

Kamila Abdiyeva is with EEE, NTU, Singapore and with Advanced Digital Sciences Center (ADSC), Singapore abdi0001@e.ntu.edu.sg
Kim-Hui Yap is with EEE, NTU, Singapore ekhyap@ntu.edu.sg
Gang Wang is with Alibaba Group
Narendra Ahuja is with University of Illinois Urbana Champaign, Champaign, IL, USA and with Advanced Digital Sciences Center (ADSC), Singapore n-ahuja@illinois.edu
Martin Lukac is with Nazarbayev University, Astana, Kazakhstan martin.lukac@nu.edu.kz

purpose CPU because all the operations are performed using the same precision as in normal CNN. The QL-Net was evaluated on the MNIST dataset, and the implementation resulted in a 5.8x reduction in computational time complexity while the accuracy drop was only 0.8% when compared with the performance of the current MAC-based CNN.

The rest of the paper is organized as follows. Section II provides the background. Section III describes the key components and the main steps of the proposed model. Section IV describes and discusses the experimentation and results. Finally, the paper is concluded by the Section V.

II. Background

Definition 1: Tensor Object: a $D_1 \times D_2 \times D_3$ dimensional object is called a tensor object, where D_2 and D_3 are spatial dimentions and D_1 is the depth of the tensor.

Figure 1 illustrates the main components of a convolutional layer in a conventional CNN model. The main objective of the layer is to transform tensor objects (See Definition 1). Let's look at the layer l in CNN with the total number of layers L. The input to the layer l is a $D \times X \times Y$ dimensional tensor object A. Each $X \times Y$ plane in A along the depth dimension D is called a feature map. A can be either an image (for l = 1) or a tensor object obtained from a previous layer $l-1$. The layer l transfers A (Figure 1(a)) into a new tensor object O (Figure 1(f)), of size $C \times X' \times Y'$.

Let's denote a 3D region in A as a Feature Patch P (Figure 1 (b)) with dimensions $D \times M \times M$, where $M < X$ and $M < Y$. Additionally, let's refer to a set of tensor objects $W = \{W_1, \ldots, W_C\}$ as a set of filters (Figure 1(c)), with each W_i having dimensions $D \times M \times M$.

The mapping of A to O is done as follows. At layer l, A is convolved with W. Each convolution involves one P, extracted iteratively from A along the spatial dimensions X and Y, and all filters from W. The output of convolution between one P and a filter W_i is a single pixel value g_i (Figure 1 (d)). The convolutional output between P and W is a tensor object G of size $C \times 1 \times 1$ (Figure 1 (e)). Combining all Gs for each P along the spatial coordinates results in the output tensor object O.

III. Proposed QL-Net

The proposed QL-Net replaces the convolution between the feature patch P and the set of filters W by performing a look-up operation in the codebook, constructed from the training data. The codebook is constructed for each layer l.

A. Codebook Construction

In order for the lookup to effectively approximate the convolution responses, the codebook needs to satisfy two major requirements: a low approximation error and an efficient codeword search.

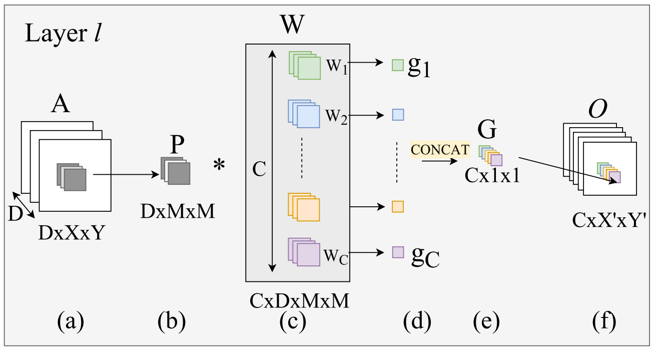

Fig. 1. Conventional CNN layer. (a) Input feature maps. (b) Feature Patch. (c) Set of Filters. (d) Convolutional output. (e) Convolution response tensor. (f) Output feature maps.

The codebook generation process consists of the following steps:

1) generating codewords by clustering,
2) calculating lookup responses by convolving the codebook codewords.

1) Codewords Generation: We start the codewords generation process by sampling a subset of images from the training set and applying a pre-trained CNN model on it to extract the feature maps A. Afterward, several Feature Patches P are selected from A to obtain the data samples for the codebook construction.

1.1) Data samples Partitioning: To improve the representational power of the constructed codebook we propose to use small dimensional codewords. Hence, sampled P are partitioned along the spatial dimensions into $N = M^2$ lower dimensional tensors (Figure 2 (a)) referred to as the Feature Patch Column $P^{(j)} \in \mathbb{R}^{D\times1\times1}$ for $j = 1 \ldots N$. This type of partitioning will be referred to as a spatial partitioning.

1.2) Codewords Generation by Clustering: The codebook is generated by clustering the collected feature patch columns. In particular, the hierarchical k-means clustering (H-K-means) was used since it provides an efficient codewords' search. H-K-means algorithm subdivides the sampled feature patch columns into k subsets using the conventional k-means algorithm. Subsequently, each subset is divided into other k subsets. The process is repeated recursively until the termination criteria are attained. In our implementation, the first termination criterion is the depth of the tree H. The second termination condition is the density threshold ρ. ρ was introduced to avoid the creation of clusters containing only a single data sample. The last criterion will guarantee that the clustering is applied to the subset if its density exceeds ρ. The clustering will result in a codebook B with the maximum $Q = k^H$ codewords $K_v \in \mathbb{R}^{D\times1\times1}$ (for $v = 1, \ldots, Q$).

2) Lookup Response Generation: After codewords were constructed, the look-up responses for each codeword should be calculated.

2.1) Filters splitting: The look-up response calculation for every codeword consists of the following steps:

a) partition the filter W_i into N sub-filters,

b) compute a look-up response.

On account of introducing the spatial partitioning, the filter vectors W_i should be subdivided into N sub-filters $W_i^{(j)}$ (for $i = 1 \dots C$ and $j = 1 \dots N$) in a similar to the feature patch partitioning manner (Figure 2 (a)). Afterwards, the look-up responses are calculated for each sub-filter position separately.

2.2) Lookup Response Calculation: Each sub-filter is applied to each codeword (K_v for $v = 1 \dots Q$) in the codebook. The respective convolutional output is obtained by eq. 1.

$$r_{i,v}^{(j)} = K_v * W_i^{(j)} \tag{1}$$

where $r_{i,v}^{(j)}$ is the scalar convolutional output between the $v-th$ codeword in the codebook and the $j-th$ sub-filter of the W_i filter. The convolutional outputs between a codeword and the sub-filters at the same position j for all C filters in the set W are concatenated together into one look-up response $R_v^{(j)} \in \mathbb{R}^{C\times1\times1}$ (Eq. 2).

$$R_v^{(j)} = r_{1,v}^{(j)} \| \dots \| r_{C,v}^{(j)}, \tag{2}$$

where $\|$ denotes the concatenation operation. Consequently, each codeword will have N look-up responses $R_v^{(j)}$, one for each j-th sub-filters position.

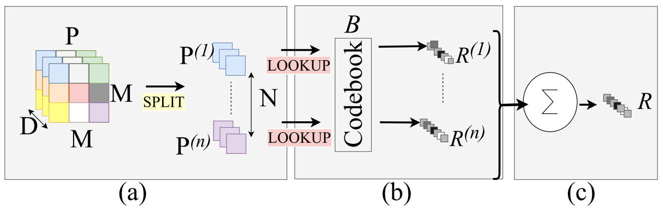

Fig. 2. QL-Net Lookup Function. (a) Partition Feature Patch into the Feature Patch Columns. (b) Lookup Feature Patch Column for Lookup Response. (c) Sum Lookup Responses.

B. QL-Net Training

So far, we have presented an algorithm for a codebook construction. This codebook is used to quantize the CNN feature maps. However, after quantizing the layer feature maps, the QL-Net model must be retrained.

The reason and challenges for retraining a quantized network are:

1) A quantization error is introduced each time A is approximated by the codewords. This quantization error causes the classification accuracy to drop. Therefore, CNN filters need to be retrained to minimize this error.

2) The quantization of a current layer dependents on the quantization of earlier layers.

To address these challenges we propose the following QL-Net training procedure:

We start with a pre-trained baseline model and perform the quantization process for one layer at a time. Let's denote the layer for quantization as a layer l and the total number of layers in the network by L. We start quantization and retraining with the Layer 2 and finish at the Layer $l_q = L-1$. Every time a layer l is quantized, all layers from 1 to $l-1$ are treated as a fixed feature extractor, due to the following reasons:

1) retraining filters of layers 1 to $l-1$ will change the input feature maps for the layer l A_l. Therefore, the codebook constructed for the layer l B_l must be build again, which will become computationally expensive if performed each time any earlier layers are modified.

2) look-up operation is not a differentiable function, thus, an exact factor to update filters into the layers before the layer l is hard to calculate.

A similar procedure for the quantization and re-training is repeated for every $1 < l < L$ layer :

1) approximate A_l with codewords from the codebook B_l,

2) retrain parameters for layers l to L.

We start the QL-Net training procedure by approximating A_l with the codewords from the layer l codebook B_l. Let's denote this quantized input as $\hat{A}_l$. The $\hat{A}_l$ is used instead of A_l to calculate the layer output feature maps. The substitution of A_l with $\hat{A}_l$ increases the classification error. To minimize this error, the retraining of the set of filters, for the layer l till L, should be performed. Therefore, precomputed look-up responses for the layer l cannot be used, and are computed only after the layer retraining.

After the parameters of layers l till L were modified, the retrained model is used to quantize the consecutive layers, from $l+1$ to l_q. For instance, to extract A_{l+1} for the layer $l+1$, the retrained QL-Net model is used, with the layers 2 to l already quantized and properly restrained, and with look-up responses for these layers calculated.

C. Inference

During the inference (Figure 2) we will have l_q number of Codebooks, each for each quantized layer. With the spatial partitioning strategy introduced, the convolution, between the $D\times M\times M$ dimensional feature patches and an equally sized filter W_i, is substituted by a sum of N separate convolutions between the lower dimensional sub-filters $W_i^{(j)}$ and the feature patch columns $P^{(j)}$ (Figure 2(a)). $P^{(j)}$ and $W_i^{(j)}$ are j-th components of the Feature Patch and the filter W_i, respectively. Accordingly, to replace the convolution by a lookup operation, we extract the look-up responses for each $P^{(j)}$ ($j = 1 \dots n$) independently (Figure 2(b)) and sum them together (Figure 2(c)), to get the final look-up response for the whole feature patch P. Since P is extracted from the A

in a spatially-overlapping manner, the same codewords can be re-used to estimate the look-up responses for the multiple positions.

IV. Experiments

The proposed model was evaluated on the MNIST [18] image classification benchmark dataset. The experimental evaluation is provided in the Figure 4.

MNIST Dataset is a benchmark image classification dataset [18]. It consists of 60000 training and 10000 test 28x28 gray-scale images representing digits ranging from 0 to 9. Sample images are provided in the Figure 3.

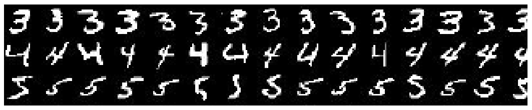

Fig. 3. MNIST dataset sample images

A. Experimental Settings

For all the experiments conducted, the utilized architecture is described in the Table I. The first column in the Table I shows

TABLE I
CNN Architecture used in the experiments

Layer	Size	Filters	Sub-Codewords size
CONV	$1 \times 5 \times 5$	20	-
MP	2×2	-	-
CONV	$20 \times 5 \times 5$	40	$20 \times 1 \times 1$
MP	2×2	-	-
CONV	$40 \times 4 \times 4$	50	$40 \times 1 \times 1$
FC	50	10	-

the type of the layer, with either CONV for a convolutional layer, MP for max pooling or FC being a fully connected layer. The second column indicates a size of the filters for the CONV layers, a size of the connectivity matrix for the FC layers and a downsampling ratio for MP layer. The third column indicates the number of filters for each layer. The last column demonstrates the codewords size. During training, the cross-entropy loss is minimized with ADAM [19]. Batch Normalization [20] with a minibatch size 100 were exploited. To construct the H-K tree we randomly selected 80000 datapoints extracted from 20000 images from the training set. The number of clusters each subset is divided into is k = 2. The following settings were used to calculate the total codebook size: (2,62), (30,62), (62,62), (62,126), (62,254), where the first number indicates the number of codewords for the second CONV layer, and the second number for the third CONV layer, according to the Table I). The centroids for K-means clustering were initialized randomly, and we haven't observed any significant influence of the initial centroids' values on the performance of the proposed algorithm.

The storage required for the dictionary can be calculated as $storage = Q \times (D \times 1 \times 1 + C \times M \times M) \times 32bit$, where Q - number of codewords. Hence, to store the dictionaries for fastest performing configuration we would require around 0.21Mb, which is equivalent to the storage size of the original network parameters. However, to store the dictionaries for the best performing model would require 1.1Mb, making the storage requirement one of the limitations for the model.

B. Experimental results

Experiments on the MNIST dataset show that the proposed model achieved 98.6% accuracy, compared to the MAC-based CNN model with 99.2% accuracy. We run a separate set of experiments to evaluate the possible reduction in computational time by comparing the time required for the MAC operations with the time taken by the look-up method. The experimental settings were similar to the one used for the MNIST experiments. The acceleration obtained using the same settings as for the model with the highest accuracy was 5.8 times. The reason for this separate experiments is that the look-up operation implemented is much less optimal than the highly optimized convolution implementation. Thus to obtain a fair comparison on the level of arithmetic operators we implemented the separate experiments designed to evaluate the relative acceleration.

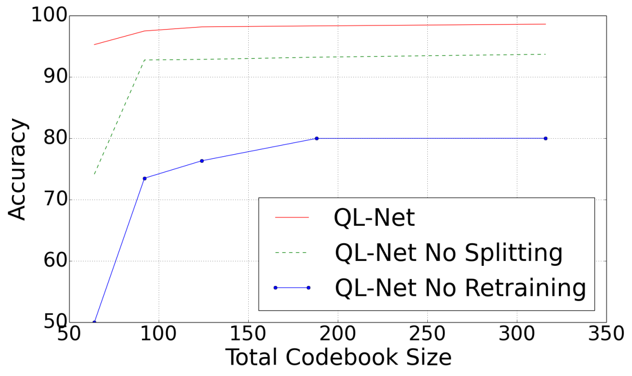

Fig. 4. Accuracy vs Total Codebook size

Figure 4 shows the relation between the dictionary size and the accuracy. In the figure, "QL-Net" is the proposed model with the spatial partitioning and retraining strategies applied. "QL-Net no splitting" is a model where only the retraining was applied, and the codebooks were constructed without the spatial partitioning. "QL-Net no Retraining" is the model where the codebooks were constructed using the spatial partitioning, but no retraining was applied. As shown in the Figure 4, the increase in the codebook size improves the model performance. However, after the total codebook size exceeds 100 codewords, the accuracy graph flattens. In fact there is improvements with increase in the codebook size but the accuracy improvement compared to the size increase is negligible. The possible explanation for this behaviour is that the dictionary representational capacity is not improving anymore and more advanced method should be used for the codebook construction.

The effect of the retraining can be observed in the Figure 4, without the layer-wise retraining the accuracy is around 20-22% lower than the accuracy of the retrained QL-Net. Experiments in Figure 4 demonstrate as well that the QL-Net performance is higher when the approximation of the Feature Patches is done using the multiple codewords. Additionally, the retrained QL-Net with smaller codewords is only slightly affected by the different size of codebook and thus can achieve the highest relative speed up. Others versions of the QL-Net are, however, much more sensitive to the codebook size variations.

Figure 5 shows the acceleration obtained when evaluating the MAC operation w.r.t. the look-up operation. As can be seen from the Figure 5, a smaller codebook provides a higher acceleration, as a smaller number of operations are required to find a respective codeword. This is as expected because the number of MAC operations required for the one convolutional layer is constant. While a codebook with a smaller dictionary requires a smaller number of comparisons between the codewords in the codebook and the feature patch. Acceleration is achieved when the number of look-up comparisons is smaller than the number of operations required for the convolution.

For instance, for a feature patch of size $3 \times 3 \times 3$ and for 512 filters of size $3 \times 3 \times 3$ the resulting tensor object will be of the size $512 \times 1 \times 1$. In such case, the number of operations to perform the convolution is $512 \times 3 \times 3 \times 3$ multiplications and $512 \times (3 \times 3 \times 3 - 2)$ addition operations.

As we used H-K-means tree, the codebook is a hierarchical structure. Thus, a dictionary with 62 codewords has 5 levels in the tree. Each level contains 2 codewords. Therefore, at maximum, we need to perform 10 comparisons to find the codeword that approximates the feature patch. The comparison was performed using the l^1-norm. Thus to transform a feature patch of size $3{\times}3{\times}3$ into a tensor object of size $512 \times 1 \times 1$ we perform at maximum 90 comparisons of codewords of size $3 \times 1 \times 1$. Each comparison requires 3 differences and 2 summations. Thus in theory, the acceleration can be estimated as $\frac{3 \times 3 \times 3 \times 512}{1 \times 1 \times 3 \times 90} \approx 51$. This shows that in the best hypothetical case there is a fifty times acceleration using our method. Naturally this is only an approximate count because the different arithmetic operations have different implementation costs. It is a valid estimator of the cost ratio as it overestimates the maximum acceleration and thus provides an upper bound.

Looking closer at both Figure 4 and Figure 5 it can be observed that the effective acceleration obtained by the proposed model is around 4 to 6 times. This corresponds to the best accuracy of the QL-Net in Figure 4. Note that this acceleration and dictionary size corresponds to one benchmark and different accelerations can be obtained on various datasets.

Conducted experiments demonstrate that quantization causes the model accuracy decrease, however, the proper retraining and the efficient codebook design can minimize its effect.

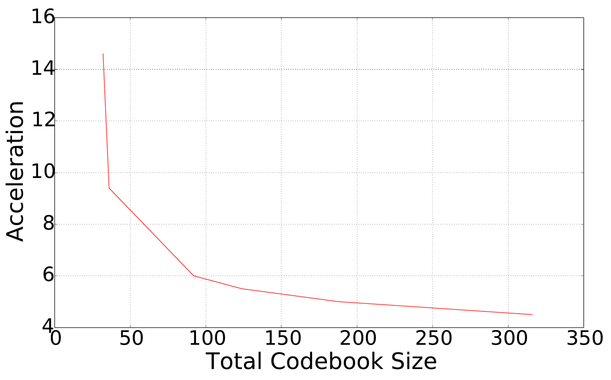

Fig. 5. Acceleration vs Total Codebook Size

V. Conclusion

We have introduced a novel quantization approach to reduce the number of operations during the inference, called QL-Net. We have evaluated the proposed algorithm on the MNIST dataset and got auspicious results. We demonstrated that the quantization by look-up is a promising solution for the network acceleration if the codebook can capture the data diversity and the network is able to compensate the information loss. Future works should extend those results to other deeper models and bigger datasets. In addition, techniques to improve the codebook quality and alternative approaches for lookup, such as fast hash-functions, will be investigated.

Acknowledgement

This work was funded by the research grant from Singapore Agency for Science, Technology and Research (A*STAR) through the SINGA program. This research was carried out at both the Advanced Digital Sciences Center (ADSC), Illinois at Singapore Pt Ltd, Singapore, and at the Rapid-Rich Object Search (ROSE) Lab at the Nanyang Technological University, Singapore. The ROSE Lab is supported by the Infocomm Media Development Authority, Singapore.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 1097–1105.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," 2016. [Online]. Available: http://arxiv.org/abs/1603.05027

[3] ——, "Deep residual learning for image recognition," *CVPR*, 2016.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Neural Information Processing Systems (NIPS)*, 2015.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[6] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 5 2015.

[7] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *TPAMI*, 2016.

[8] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *arXiv preprint arXiv:1703.06870*, 2017.

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

[10] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, *XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks*, 2016.

[11] M. Courbariaux, I. Hubara, D. Soudry, R. E. Yaniv, and Y. Bengio, "Binarized neural networks," 2016. [Online]. Available: http://arxiv.org/abs/1602.02505

[12] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, "Neural networks with few multiplications," *ICLR*, 2016.

[13] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016. [Online]. Available: http://arxiv.org/abs/1606.06160

[14] M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini, "Fast neural networks without multipliers," *Trans. Neur. Netw.*, vol. 4, no. 1, pp. 53–62, Jan. 1993. [Online]. Available: http://dx.doi.org/10.1109/72.182695

[15] F. Li and B. Liu, "Ternary weight networks," *CoRR*, vol. abs/1605.04711, 2016. [Online]. Available: http://arxiv.org/abs/1605.04711

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

[17] H. Bagherinezhad, M. Rastegari, and A. Farhadi, "LCNN: lookup-based convolutional neural network," *CoRR*, vol. abs/1611.06473, 2016. [Online]. Available: http://arxiv.org/abs/1611.06473

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86(11), pp. 2278–2324, 1998.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[20] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167