# FIXED-POINT OPTIMIZATION OF DEEP NEURAL NETWORKS WITH ADAPTIVE STEP SIZE RETRAINING

*Sungho Shin, Yoonho Boo and Wonyong Sung*

Department of Electrical and Computer Engineering
Seoul National University
Seoul, 08826 Korea
Email: sungho.develop@gmail.com, yhboo.research@gmail.com, wysung@snu.ac.kr

## ABSTRACT

Fixed-point optimization of deep neural networks plays an important role in hardware based design and low-power implementations. Many deep neural networks show fairly good performance even with 2- or 3-bit precision when quantized weights are fine-tuned by retraining. We propose an improved fixed-point optimization algorithm that estimates the quantization step size dynamically during the retraining. In addition, a gradual quantization scheme is also tested, which sequentially applies fixed-point optimizations from high- to low-precision. The experiments are conducted for feed-forward deep neural networks (FFDNNs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs).

***Index Terms***— deep neural networks, recurrent neural networks, fixed-point quantization, step size adaptation

## 1. INTRODUCTION

Deep neural networks (DNNs) show very high performance in various fields such as speech recognition [1] and image classification [2]. However, real-time implementation of DNNs usually demands many arithmetic and weight fetch operations. Thus, word-length optimization is needed in embedded applications to reduce the strength of arithmetic and the size of the weight storage. However, reducing the word length too much tends to degrade the performance. Thus, developing optimum quantization methods is greatly needed for efficient implementation of neural network algorithms.

Direct quantization of deep neural networks usually does not show satisfactory performance with very low precision weights. However, when the quantized weights are optimized by retraining, the fixed-point performance improves dramatically. Even ternary valued weights (+1, 0, and -1) for a DNN have yielded satisfactory performance [3, 4]. Recently, several improved fixed-point optimization methods are developed by employing retraining based fine tuning [5, 6]. Also, VLSI and FPGA based deep neural networks have been implemented using fixed-point weights [7, 8, 9, 10, 11].

This work was supported in part by the Brain Korea 21 Plus Project and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2015R1A2A1A10056051).

In this work, an improved retraining algorithm is developed for fixed-point optimization of deep neural networks. The previous works decide the optimum quantization step size based on the distribution of floating-point weights and freezes the step-size during the retraining period [4, 5]. The proposed algorithm adaptively determines the step-size at the re-quantization step during retraining. Since the weight values change much at the beginning of retraining, this approach is especially effective when applied at initial retraining epochs. In order to change the weight values less abruptly, we also propose and evaluate the gradual quantization method. In this schemes, floating-point weights are converted to, for example, 6-bit weights, which are then converted to 4-bit weights, and so on. We evaluate the proposed schemes in three different networks: feed-forward deep neural networks (FFDNNs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs). The proposed methods yielded better results compared to the previous retrain-based quantization schemes.

The rest of this paper is organized as follows. Section 2 presents the proposed quantization with step size adaptation during the retraining procedure. The gradual quantization scheme is also explained. Experimental results on FFDNN, CNN, and RNN applications are shown in Section 3. Concluding remarks follow in Section 4.

## 2. STEP SIZE ADAPTATION AND GRADUAL QUANTIZATION FOR RETRAINING OF DEEP NEURAL NETWORKS

In this section, we explain the conventional retrain based fixed-point optimization algorithm, and present adaptive step size retraining and gradual quantization methods.

### 2.1. Retrain-based fixed-point quantization review

The original retrain based fixed-point optimization algorithm can be represented briefly as shown in Figure 1. Note that, conventional algorithms [3, 4, 12] do not compute $\Delta_{new}$ at the 'weights update' stage. In this figure, after obtaining the floating-point weights by training, the quantization step size, $\Delta$, is determined by minimizing the L2 error between the floating-point and fixed-point weights. For the convenience of arithmetic, uniform quantization is assumed. Two algorithms

**- Quantization step size determining:**

$$\Delta = QStep(\boldsymbol{w}) = \underset{\Delta}{\text{argmin}}\ \frac{1}{2}\sum_{i=1}^{N}\big(Q(w_i,\Delta) - w_i\big)^2$$

**- Quantized weights:**

$$\boldsymbol{w}^{(q)} = Q(\boldsymbol{w},\Delta) = sgn(\boldsymbol{w})\cdot\Delta\cdot min\left(\left\lfloor\frac{|\boldsymbol{w}|}{\Delta}+0.5\right\rfloor, \frac{M-1}{2}\right)$$

**- Forward:**

$$net_i = \sum_{j\in A_i} w_{ij}^{(q)} y_j$$

$$y_i = \phi_i(net_i)$$

**- Backward:**

$$\delta_j = \phi_j^{'}(net_j)\sum_{i\in P_j}\delta_i w_{ij}^{(q)}$$

**- Gradient calculation:**

$$\frac{\partial E}{\partial w_{ij}} = -\delta_i y_j$$

**- Weights update:**

$$w_{ij,new} = w_{ij} - \alpha\frac{\partial E}{\partial w_{ij}}$$

$$\Delta_{new} = QStep(w_{ij,new})$$ **(Proposed scheme)**

$$w_{ij,new}^{(q)} = Q_{ij}(w_{ij,new},\Delta_{new})$$

**Fig. 1**: Overall fixed-point retraining algorithm with step size adaptation scheme, where $\Delta$ is the quantization step size, $\boldsymbol{w}$ is the weight groups, $net_i$ is the summed input value of unit $i$, $\delta_i$ is the error signal of unit $i$, $M$ is quantization points (2-bit quantization = 3 points, 3-bit quantization = 7 points), $\alpha$ is the learning rate, $N$ is the number of the weights in each layer, $A_i$ and $P_j$ represent the activation of next and previous layer, $\phi(\cdot)$ is the activation function, $E$ is the output error, and superscript $(q)$ means the value is quantized.

have been developed for the quantization step size optimization. One is an exhaustive search, which decides the initial quantization step size $\Delta_{initial}$ by considering the weight distribution, and then searches the best performing step size between $\Delta_{initial}/2$ and $2\cdot\Delta_{initial}$ by testing the quantized network with the evaluation set [4]. The second approach is deciding the quantization step size by measuring the mean and the variance of the floating-point weights [5]. Then, in the second stage of Figure 1, floating-point weights are rounded to fixed-point values by using the determined quantization step size. The third stage is the inferencing or the forward stage with the quantized network, $w^{(q)}$. The error signal is calculated and used for backward propagation. The gradient is calculated and weight update is conducted. Note that the floating-point weights, instead of the fixed-point values, are updated because the amount of weight update is usually much smaller than the quantization step size. Then, the fixed-point weight update, yielding $w_{ij,new}^{(q)}$, is accomplished by quantizing the updated floating-point weights. Note that determining $\Delta_{new}$ is not performed in the conventional method, and the same quantization step size is used at every iteration.

## 2.2. Step-size adaptation during retraining

As described in Section 2.1, the conventional method freezes the step size during the retraining. However, in many cases, the weight values change much by retraining. Note that the amount of change decreases as the retraining iteration progresses. Thus, it is advantageous for improving the performance to adjust the quantization step size during the retraining. Especially, the need of step size adaptation is greater at the beginning of retraining. The proposed scheme adds the determination of $\Delta_{new}$ at the weight update stage of Figure 1.

We do not perform 'exhaustive search' anymore but update the quantization step size during retraining by using the L2 error minimization between the floating-point and fixed-point weights. We consider two different quantization step size update timing. The first one is 'epoch-level update', and the other is '1 epoch update & fix'. The 'epoch-level update' changes the step size at every epoch. The '1 epoch update & fix' updates the step size only during one or two epochs and freezes it for the remaining epochs. In our empirical evaluation, the first scheme is good for FFDNNs, but the second one shows better results for CNNs and RNNs. The specific results will be given in Section 3.

## 2.3. Gradual quantization scheme

We also propose another step size adaptation approach which is similar to the curriculum learning. The curriculum learning is a training strategy to move the goal from an easy level to more complex one gradually [13]. One of the important points in curriculum learning is how to organize the tasks from easy to complex ones. We consider that the fixed-point optimization with a small number of bits is a more difficult problem than that with a large one.

In the proposed scheme, we begin fixed-point optimization with a fairly high precision, such as 6 bits, and then keep lowering the word-length by one bit with retraining for each precision. At each retraining process with a given precision, we also combine the proposed quantization step size adaptation scheme. The experiments are conducted for FFDNNs.

## 3. EXPERIMENTAL RESULTS

The proposed step size adaptation is evaluated for three applications. We employ FFDNNs for phoneme recognition, CNNs for house number recognition, and RNNs for language modeling. To analyze the effect of step size adaptation, we change the size of each network and their word lengths.

## 3.1. Phoneme recognition using feed-forward deep neural networks

The FFDNN is trained with the TIMIT corpus [14], and the detailed experimental condition for the data preprocessing is the same with [15]. We construct 11 consecutive frames as the network input. The output layer supports 61 labels, and the labels are merged into 39 classes for the final evaluation. For performance evaluation, the number of units in each layer increases from 64 to 1024. We train the floating-point networks using

**Table 1**: Frame-level phoneme error rate (%) on the test set with the TIMIT phoneme recognition examples. Note that 'conventional' is the baseline [4] and 'adaptive' is the proposed scheme.

| | | Without BN | | | | | With BN | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Size of each layer | 64 | 128 | 256 | 512 | 1024 | 64 | 128 | 256 | 512 | 1024 |
| | Floating result | **34.38** | **31.63** | **30.17** | **29.61** | **29.53** | **33.82** | **30.81** | **29.79** | **29.77** | **29.59** |
| 2-bit (3 point) | Direct | 80.25 | 84.12 | 81.92 | 83.30 | 75.05 | 89.82 | 88.79 | 87.57 | 85.73 | 86.10 |
| | Conventional | 43.73 | 37.80 | 33.70 | 31.43 | 29.99 | 41.81 | 35.88 | 33.12 | 31.21 | 30.22 |
| | Adaptive | **42.06** | **36.88** | **32.61** | **30.61** | **29.49** | **37.87** | **33.46** | **31.48** | **30.73** | **30.09** |
| 3-bit (7 point) | Direct | 68.13 | 63.65 | 60.33 | 51.46 | 48.61 | 80.41 | 69.55 | 69.42 | 81.60 | 64.55 |
| | Conventional | 40.63 | 34.73 | 31.41 | 30.49 | **29.33** | 36.88 | 32.58 | 30.53 | 30.14 | 29.76 |
| | Adaptive | **37.89** | **33.80** | **30.74** | **29.83** | 29.40 | **35.29** | **31.94** | **30.32** | **30.10** | **29.65** |
| 4-bit (15 point) | Direct | 58.90 | 50.58 | 42.15 | 38.05 | 36.53 | 65.63 | 50.43 | 46.46 | 43.80 | 39.77 |
| | Conventional | 36.51 | 32.65 | 30.79 | 29.95 | 29.44 | 34.17 | 31.34 | 29.86 | **29.81** | 29.70 |
| | Adaptive | **35.50** | **32.09** | **30.50** | **29.54** | **29.29** | **33.91** | **30.86** | **29.47** | 29.87 | **29.52** |

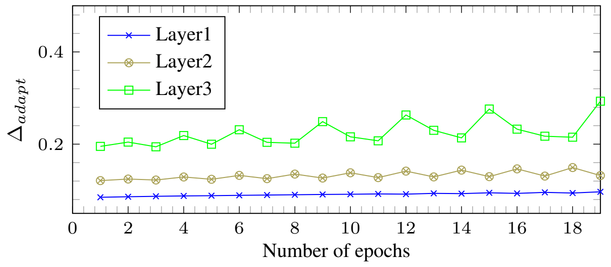

**Fig. 2**: Training curves in terms of $\Delta_{adapt}$ with the 256 size of the FFDNN.

the stochastic gradient descent (SGD) with Nesterov momentum [16]. The learning rate decreases from 2e-3 to 3.90625e-6 with a factor of 2 when the development set does not show improvements for 4 consecutive evaluations. For fixed-point networks training, all other conditions are the same with the floating-point case but the initial learning rate is 5e-4.

The results of fixed-point optimization for FFDNNs with and without the step size adaptation are reported in Table 1. The experiments also show the results with batch normalization (BN) [17]. The step size is updated using the 'epoch-level update' until the end of the retraining. Table 1 shows that the floating-point network performance saturates at 512 units size when BN is applied, and at 256 units when BN is not used. When the unit size in each layer is 512 or smaller, the proposed algorithm yields better performance in both cases. For example, if the 512 units size network is quantized in 2-bit without BN, the differences between the floating-point and the fixed-point networks are 1.82% and 1% for 'conventional' and 'adaptive' schemes, respectively. In addition, the phoneme error rate of the 3-bit network optimized with the 'adaptive' scheme (29.83%) is lower than that of the 4-bit quantized network with the 'conventional' scheme (29.95%).

BN improves the performance of both floating-point and fixed-point networks. Applying the 'adaptive' method improves the performance. For example, if the layer unit size is 128 and 2-bit quantization is used, BN brings the performance gain of 3.42% when 'adaptive' scheme is used. Therefore, the proposed

**Table 2**: Evaluation of the proposed quantization strategies on TIMIT phoneme recognition task. The network is FFDNN with two 512 size hidden layers, and the floating-point result is 29.61%. 'Conventional' is general retraining based quantization, 'adaptive' conducts proposed step size adaptation, 'gradual' is curriculum learning style quantization scheme, and 'adaptive & gradual' represents mixed approach using both techniques.

| | Conventional | Adaptive | Gradual | Adaptive & Gradual |
|---|---|---|---|---|
| 6-bit | **29.32** | **29.32** | **29.32** | **29.32** |
| 4-bit | 29.95 | 29.54 | 29.53 | **29.49** |
| 3-bit | 30.49 | 29.83 | 29.90 | **29.61** |
| 2-bit | 31.43 | **30.61** | 30.69 | 30.62 |

'adaptive' method can be efficiently used with BN.

When the unit size is large enough, the quantization scheme does not affect the performance much because a larger size network has a better resiliency to quantization [18]. Even the 4-bit quantized 512 units size network without BN shows the performance almost comparable to the floating-point 1024 units size network. When the network is trained with BN, it shows a similar trend.

Figure 2 shows the step size $\Delta$ of the proposed adaptive scheme as the retraining progresses. Note that the step size is renewed at each epoch during retraining. As shown in this figure, the step size of the last layer varies much, while that of the first layer is almost constant. The step size adaptation is much needed for the last layer.

We also evaluate the performance of the gradual quantization scheme. The results are reported in Table 2. The floating point results show 29.61% error rate on the test set. The 6-bit word length shows slightly better accuracy than the floating point. Thus, we define the easiest task as the 6-bit quantization. In Table 2, the 'gradual' scheme yields better performance than the 'conventional' strategy, but shows worse or similar results compared to the 'adaptive' quantization. The combined strategy of the 'adaptive' and 'gradual' shows slightly better ac-

**Table 3**: Miss classification rate on the test set with the SVHN house number recognition example. The alphabets 'L', 'C', and 'V' represent specific structure of the CNN. The 'L' is the most smallest network and the 'V' is the biggest network. Please refer Section 3.2 for details.

| | Type of network | L | C | V |
|---|---|---|---|---|
| | Floating result | ***6.45*** | ***5.65*** | ***4.50*** |
| 2-bit (3 point) | Direct | 45.68 | 23.17 | 73.55 |
| | Conventional | 8.37 | 7.10 | 5.24 |
| | Adaptive | **8.01** | **6.65** | **5.02** |
| 3-bit (7 point) | Direct | 10.14 | 7.88 | 6.73 |
| | Conventional | 7.04 | 5.97 | 4.57 |
| | Adaptive | **6.92** | **5.91** | **4.53** |
| 4-bit (15 point) | Direct | 7.85 | 6.03 | 4.79 |
| | Conventional | 6.60 | **5.76** | 4.74 |
| | Adaptive | **6.46** | 5.86 | **4.60** |

**Table 4**: Bit per character (BPC) on the test set with the English Wikipedia language model.

| | Size of each layer | 64 | 128 | 256 |
|---|---|---|---|---|
| | Floating result | ***2.07*** | ***1.81*** | ***1.65*** |
| 2-bit (3 point) | Direct | 8.46 | 9.53 | 7.26 |
| | Conventional | 2.48 | 2.49 | 1.89 |
| | Adaptive | **2.42** | **2.16** | **1.86** |
| 3-bit (7 point) | Direct | 7.176 | 6.84 | 4.35 |
| | Conventional | 2.52 | 2.10 | 1.91 |
| | Adaptive | **2.35** | **2.06** | **1.82** |
| 4-bit (15 point) | Direct | 4.49 | 5.50 | 2.59 |
| | Conventional | 2.43 | 2.04 | **1.83** |
| | Adaptive | **2.32** | **1.95** | 1.86 |
| 6-bit (63 point) | Direct | 2.56 | 3.73 | 1.73 |
| | Conventional | **2.11** | **1.87** | **1.67** |
| | Adaptive | **2.11** | 1.89 | 1.68 |

curacy than the 'adaptive' strategy in 4- and 3-bit quantization, but it is worse than the 'adaptive' scheme in 2-bit quantization. Since there is no performance difference between the 'adaptive' and 'adaptive & gradual' scheme, we only employ the 'adaptive' scheme for CNN and RNN experiments.

### 3.2. Image classification using convolutional neural networks

Image classification experiments are performed on the SVHN dataset [19]. The dataset includes 600,000 labeled 32x32 three channel images from real world house numbers. For the data preprocessing, we employ the same method with [20]. The output label has ten units which represent the numbers from 0 to 9. For evaluation of the proposed scheme, we employ three different structures. We name the networks as 'L', 'C', and 'V' which have the trainable parameters 60k, 84k, and 435k, respectively. The 'L' network is Lenet5 [21], 'C' network is from [22], and 'V' network is constructed as VGG style which is from [23]. We train the floating-point networks using SGD with Nesterov momentum. The learning rate is decreased from 2-e2 to 3.125e-4 with a factor of 2 when the development set does not show improvement for 4 consecutive evaluations. For the fixed-point network training, the initial learning rate was 5e-4. The effects of step size adaptation in the CNNs are examined in Table 3. The step size is updated using the '1 epoch update & fix' strategy. Our algorithm works well for 'L' and 'V' networks regardless of the weight precision, 2, 3, or 4 bits. However, the 'C' networks with the conventional retraining show a better result when the weight precision is 4bits. Overall, the proposed method yields improved performances.

### 3.3. Language modeling using recurrent neural networks

Character-level language modeling predicts the next character, and is used for speech recognition and text generation. Since the input and output layers consider only alphabets, the input and output complexity are much lower than the word level language model. We adopt English Wikipedia dataset for training the character level language modeling. The dataset contains 100 MB English Wikipedia text. The input and output layers are composed of 256 units for one-hot encoded ASCII code. The RNN consists of three Long Short-Term Memory (LSTM) layers with a different number of memory cells ranging from 64 to 256 [24]. We train the RNNs using AdaDelta based SGD with 64 parallel input streams. The networks are unrolled 256 times and weights update is performed for128 forward steps. The learning rate starts from 5e-4 and decreases until 5e-8. For the step size adaptation, '1 epoch update & fix' strategy is employed. The fixed-point optimization results are reported in Table 4. As with our previous FFDNN and CNN results, it shows much improved performances on low-precision weights or small size networks.

## 4. CONCLUDING REMARKS

We have developed improved fixed-point weight optimization methods for deep neural networks. The first one adaptively determines the quantization step size by measuring the weight distribution during the retraining procedure. The second one is a curriculum style fixed-point optimization technique, which conducts fixed-point optimization from high- to low-precision gradually. The proposed work yields better quantization results in FFDNN, CNN, and RNN experiments. Especially the effectiveness of the proposed techniques increases when the number of quantization levels is small and the network size is not large enough.

## 5. REFERENCES

[1] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, et al., "Deep speech 2: End-to-end speech recognition in english and mandarin," in *ICML 2016: 33rd International Conf. Machine Learning*, 2016.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, “Deep residual learning for image recognition,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[3] Sungho Shin, Kyuyeon Hwang, and Wonyong Sung, “Fixed-point performance analysis of recurrent neural networks,” in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 976–980.

[4] Kyuyeon Hwang and Wonyong Sung, “Fixed-point feedforward deep neural network design using weights +1, 0, and -1,” in *2014 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2014, pp. 1–6.

[5] Darryl D Lin, Sachin S Talathi, and V Sreekanth Annapureddy, “Fixed point quantization of deep convolutional networks,” in *ICML 2016: 33rd International Conf. Machine Learning*, 2016.

[6] Song Han, Huizi Mao, and William J Dally, “Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding,” *CoRR, abs/1510.00149*, vol. 2, 2015.

[7] Jonghong Kim, Kyuyeon Hwang, and Wonyong Sung, “X1000 real-time phoneme recognition vlsi using feedforward deep neural networks,” in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 7510–7514.

[8] Jinhwan Park and Wonyong Sung, “FPGA based implementation of deep neural networks using on-chip memory only,” in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 1011–1015.

[9] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally, “EIE: efficient inference engine on compressed deep neural network,” in *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*, 2016, pp. 243–254.

[10] Minjae Lee, Kyuyeon Hwang, Jinhwan Park, Choi Sungwook, Shin Sungho, and Wonyong Sung, “FPGA-based low-power speech recognition with recurrent neural networks,” in *2016 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2016.

[11] Nicholas J Fraser, Yaman Umuroglu, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers, “Scaling binarized neural networks on reconfigurable logic,” in *To appear in the PARMA-DITAM workshop at HiPEAC*, 2017, vol. 2017.

[12] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung, “Fixed point optimization of deep convolutional neural networks for object recognition,” in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 1131–1135.

[13] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston, “Curriculum learning,” in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 41–48.

[14] John S Garofolo, Lori F Lamel, William M Fisher, Jonathon G Fiscus, and David S Pallett, “DARPA TIMIT acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1,” *NASA STI/Recon technical report n*, vol. 93, 1993.

[15] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton, “Speech recognition with deep recurrent neural networks,” in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[16] Ilya Sutskever, James Martens, George E Dahl, and Geoffrey E Hinton, “On the importance of initialization and momentum in deep learning.,” *ICML (3)*, vol. 28, pp. 1139–1147, 2013.

[17] Sergey Ioffe and Christian Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *ICML*, 2015, pp. 448–456.

[18] Wonyong Sung, Sungho Shin, and Kyuyeon Hwang, “Resiliency of deep neural networks under quantization,” *arXiv preprint arXiv:1511.06488*, 2015.

[19] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng, “Reading digits in natural images with unsupervised feature learning,” 2011.

[20] Pierre Sermanet, Soumith Chintala, and Yann LeCun, “Convolutional neural networks applied to house numbers digit classification,” in *Pattern Recognition (ICPR), 2012 21st International Conference on*. IEEE, 2012, pp. 3288–3291.

[21] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[22] Alex Krizhevsky, “cuda-convnet: High-performance c++/cuda implementation of convolutional neural networks,” 2012.

[23] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David, “Binaryconnect: Training deep neural networks with binary weights during propagations,” in *Advances in Neural Information Processing Systems*, 2015, pp. 3123–3131.

[24] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber, “Learning precise timing with lstm recurrent networks,” *Journal of machine learning research*, vol. 3, no. Aug, pp. 115–143, 2002.