FAST AND ACCURATE IMAGE RECOGNITION USING DEEPLY-FUSED BRANCHY NETWORKS

Mou-Yue Huang[†⋆] *Ching-Hao Lai*[⋆] *Sin-Horng Chen*[†]

[†] Institute of Electrical and Computer Engineering, National Chiao Tung University, Hsinchu, Taiwan
[⋆]Industrial Technology Research Institute, Hsinchu, Taiwan

ABSTRACT

In order to achieve higher accuracy of image recognition, deeper and wider networks have been used. However, when the network size gets bigger, its forward inference time also takes longer. To address this problem, we propose Deeply-Fused Branchy Network (DFB-Net) by adding small but complete side branches to the target baseline main branch. DFB-Net allows easy-to-discriminate samples to be classified faster. For hard-to-discriminate samples, DFB-Net makes probability fusion by averaging softmax probabilities to make collaborative predictions. Extensive experiments on the two CIFAR datasets show that DFB-Net achieves state-of-the-art results to obtain an error rate of 3.07% on CIFAR-10 and 16.01% on CIFAR-100. Meanwhile, the forward inference time (with a batch size of 1 and averaged among all test samples) only takes 10.4 ms on CIFAR-10, 18.8 ms on CIFAR-100, using GTX 1080 GPU with cuDNN 5.1.

Index Terms— Deep learning, convolutional neural network, image recognition, classification, inference time.

1. INTRODUCTION

CNNs have demonstrated its power in many image recognition tasks [1, 2, 3, 4, 5, 6, 7, 8, 9]. However, to achieve remarkable accuracy of image recognition by increasing network size, including depth and width, comes at the expense of much more latency for forward evaluations. E.g., benchmarks for popular CNN models[1] on ImageNet dataset [10] show that the latency at test time has been increased from 7.0 milliseconds (ms) (AlexNet [1]), to 109.32 ms (ResNet [5]) although top-1 error has been reduced from 42.90%, to 22.16%.

To address this issue, BranchyNet [11] introduced early-exit branchy networks to speed up the forward inference. In their design, side branches are added into the original baseline models, e.g., LeNet [12], AlexNet [1], and ResNet [5], for fast inference. But their results on CIFAR-10 dataset reflect two points worth thinking: (1) better design for side branches, and (2) synergy of exit-point probabilities for better prediction.

Thanks to ITRI, Hsinchu, Taiwan (Grant no. G301AR9H00) for funding.

[1]https://github.com/jcjohnson/cnn-benchmarks

Inspired by the research works [11, 13, 14, 15], we propose Deeply-Fused Branchy Network (DFB-Net) to address these two points. Figure 1 demonstrates an example of DFB-Net. Notice that BranchyNet is different from ours in two aspects: (1) the side branches added in BranchyNet are not equipped with "small but complete" structures (see Section 2.1 for details). We argue that such types of side branches are crucial to allow much more data points to exit earlier and thus result in higher speedup gain when compared with the baseline model. (2) BranchyNet doesn't make probability fusion for collaborative decision. But we believe that collaborative decision by making probability fusion should be able to provide better prediction for those hard-to-discriminate samples. Experiments on the two CIFAR datasets show that our DFB-Net achieves state-of-the-art results and yet only takes low latency for forward inference.

Major contributions of this paper are threefold: (1) we propose to use small but complete side branches to strongly encourage much more test samples to exit earlier and thus get higher speedup gain for fast inference. (2) We make probability fusion to provide better prediction collaboratively and thus obtain much lower error rates. (3) DFB-Net provides an intuitive, probability-based, exit-threshold setting for a flexible trade-off between inference time and accuracy, and thus greatly facilitates application deployment.

2. DEEPLY-FUSED BRANCHY NETWORKS

Before proceeding, we recall two terms given in [11]: (1) *baseline model*, and *main branch*, denote the original network before any side branches are added, and (2) a *branch*, or *side branch*, consists of contiguous layers which do not overlap with other branches, and ends with an exit point. Besides, we define a *complete path* of DFB-Net as the entire path going from input layer to softmax layer of a branch.

2.1. Architecture Design

A baseline model is developed for the evaluation of gains brought by DFB-Net. In designing the main branch, we utilize the network topology used in [5], which consists of five convolutional groups followed by a global average pooling

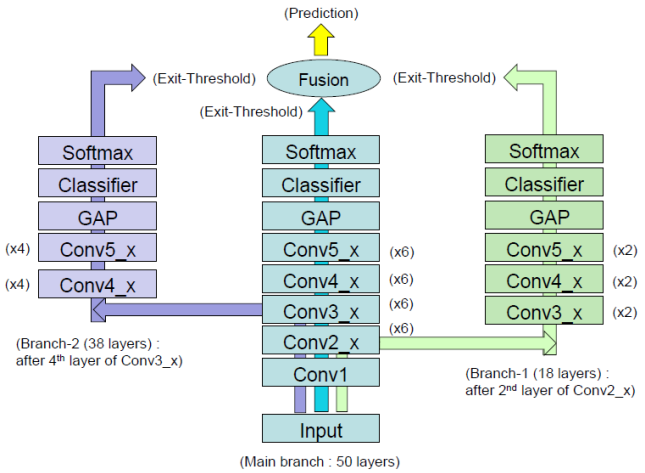

Fig. 1. An example of DFB-Net for forward inference. Here two "small but complete" side branches are added to the central main branch. Each branch has its own exit threshold for early-exit decision. The sidenotes ($\times 2$), ($\times 4$), and ($\times 6$) denote the values of N defined in Table 1.

Group Name	Output Size	Block Type
Conv1	56×56	$[3 \times 3, \quad 16]$
Conv2_x	56×56	$\begin{bmatrix} 3 \times 3, & 16 \times k \\ 3 \times 3, & 16 \times k \end{bmatrix} \times N$
Conv3_x	28×28	$\begin{bmatrix} 3 \times 3, & 32 \times k \\ 3 \times 3, & 32 \times k \end{bmatrix} \times N$
Conv4_x	14×14	$\begin{bmatrix} 3 \times 3, & 64 \times k \\ 3 \times 3, & 64 \times k \end{bmatrix} \times N$
Conv5_x	7×7	$\begin{bmatrix} 3 \times 3, & 128 \times k \\ 3 \times 3, & 128 \times k \end{bmatrix} \times N$
Global-Ave-Pool	1×1	$[7 \times 7]$

Table 1. An example of the network topology for DFB-Net main branch. Follow the definitions given in [7], N is a number of blocks in a convolutional group, and k is a widening factor to increase the number of filter size. We set $N = 6$, and $k = 6$ in our experiments on the two CIFAR datasets. At test time, an image is resized to 56×56 for forward inference, however, at training stage, a crop of 48×48 is sampled.

(GAP) layer and ends with a classification layer with softmax, as shown in Table 1. But the building blocks are composed of wide ResNet structures [7], not bottleneck designs. The total number of weighted layers is equal to $8 \times N + 2$. E.g., in Figure 1, we adopt $N = 6$, $k = 6$ to result in a baseline CNN with 50 layers, denoted by WRN-50-N6-k6.

While adding side branches to form a complete DFB-Net, we make side branches be *small but complete*: *"Small"* implies that side branches are equipped with fewer number of layers and smaller filter size for faster forward inference. *"Complete"* means that a branchy sub-network still has a complete form analogous to the main branch. We argue that the methodology of adding small but complete side branches is critical to DFB-Net design because: (1) for forward inference, this design should significantly increase the ratio of exit samples and thus would bring major gains in speedup. (2) Such branchy sub-network should be able to obtain good enough classification accuracy so that the whole DFB-Net could sustain or even outperform the performance achieved by the original main branch.

As illustrated in Figure 1, side branch 1 consists of 18 layers (counting from input to softmax layers) and provides the first exit point. Its additional groups Conv3_x to Conv5_x are designed by $N = 2$ and $k = 2$. The branchy point is located after the second layer of Conv2_x of the main branch. As the number of parameters and computation cost are quadratic in the widening factor k, for forward inference, branch 1 brings major gains in speedup (compared with the main branch). To add side branch 2, which provides the second exit point, we adopt the wide ResNet structures with $N = 4$, $k = 4$ for Conv4_x to Conv5_x and joins to the main branch directly after the fourth layer of Conv3_x.

2.2. Network Training

For brevity, we number all exit points of DFB-Net starting from 1,..., M, where M is for the exit point of the main branch, and 1 is for the exit point of the earliest side branch, and so on. Since there are more than one exit points in DFB-Net, training is done by solving a joint optimization problem on a weighted sum of loss $L_s(W_s)$ associated with each exit point, where $s = 1,\ldots, M$, and W_s denotes the sets of parameters contained in the specified complete path respectively.

Given a training example $\boldsymbol{x}$, we treat each complete path as a feature extractor $f(\cdot)$. Then the output $\boldsymbol{z}$ of a complete path just before the softmax layer is given by

$$\boldsymbol{z} = f(\boldsymbol{x}; W_s).$$

Suppose that the number of all possible labels is K, therefore, the predicted probability y_c, $c = 1,\ldots, K$, for label c produced by the softmax layer can be written as

$$y_c = \frac{\exp(z_c)}{\sum_{k=1}^{K} \exp(z_k)},$$

and we define

$$\boldsymbol{y} = \{y_c\}_{c=1}^{K} = \text{softmax}(\boldsymbol{z}).$$

Thus the loss function $L_s(W_s)$ for a complete path can be formulated in the following form

$$L_s(W_s) = -\sum_{k=1}^{K} t_k \ln y_k,$$

where t_k denotes the corresponding ground truth label for sample $\boldsymbol{x}$ and uses the 1-of-K coding scheme. The combined

Algorithm 1: DFB-Net Forward Inference

Input: A test image $\boldsymbol{x}$, exit thresholds $\{p_s\}$
Output: The predicted label of test image $\boldsymbol{x}$

procedure DFB-Net($\boldsymbol{x}$, $\{p_s\}$)
  Initialize $\bar{\boldsymbol{y}} = 0$
  for $s = 1, \ldots, M$ **do**
    $\boldsymbol{z} = f(\boldsymbol{x}; W_s)$
    $\boldsymbol{y} = \text{softmax}(\boldsymbol{z})$
    if $\max\{\boldsymbol{y}\} > p_s$ **then**
      return $argmax\{\boldsymbol{y}\}$
    else
      $\bar{\boldsymbol{y}} = \bar{\boldsymbol{y}} + \boldsymbol{y}$
  $\bar{\boldsymbol{y}} = \bar{\boldsymbol{y}} / M$
  return $argmax\{\bar{\boldsymbol{y}}\}$

loss function for the whole DFB-Net can be expressed by

$$L_{total} = \sum_{s=1}^{M} \alpha_s \, L_s(W_s),$$

where α_s is a loss weight associated with each branchy loss function $L_s(W_s)$.

2.3. Forward Inference

To perform forward inference on an already-trained DFB-Net, we formulate the procedures in the Algorithm 1. Firstly, for each exit point s, we need to assign an exit threshold $p_s \in [0, 1]$ as a confidence measure. Given a test image $\boldsymbol{x}$, its softmax probability $\boldsymbol{y}$ generated by an exit point s is used for early-exit decision: if $\max\{\boldsymbol{y}\} > p_s$, then return argmax$\{\boldsymbol{y}\}$ as the predicted label from this exit point and stop further computation; otherwise, continue the forward evaluations in following layers of the next branch. If $\max\{\boldsymbol{y}\} \leq p_s$, for all s, then we make probability fusion by averaging softmax outputs of all exit points to obtain the average $\bar{\boldsymbol{y}}$ and then return argmax$\{\bar{\boldsymbol{y}}\}$ as the predicted label.

3. EXPERIMENTS

The DFB-Net used in our experiments is illustrated in Figure 1. While network training, exit thresholds are replaced by loss weights to form a weighted sum of losses as an output of the fusion unit. We use SGD with momentum and weight decay to train the baseline model from scratch. Once trained, we initialize the main branch of DFB-Net with the weights of the already-trained baseline model. For side branches, their weights are initialized by the method [16], and then we train the whole DFB-Net. For baseline model training, the learning rate starts from 0.1 and is dropped by 0.2 every 60 epochs, and the models are trained for total 300 epochs. We set the weight decay to 0.0001, momentum to 0.9, and mini-batch size to 50.

We evaluate DFB-Net on CIFAR-10 and CIFAR-100 [17], and compare our results with state-of-the-art methods. The two CIFAR datasets consist of 32×32 color images drawn from 10 and 100 classes respectively, and each contains 50,000 images for train set and 10,000 images for test set. We apply scale and aspect ratio data augmentation by randomly choosing two values h, $w \in [48, 64]$ and then resize an image to $h \times w$. Then a 48×48 crop is randomly sampled from the resized image or its horizontal flip, with the per-pixel mean subtracted.

At test time, an image is resized to 56×56 without any crop for forward inference with a batch size of 1, and the runtime reported in this paper is the average among all test samples over three trials running on NVIDIA GeForce GTX 1080 (8GB) GPU with CUDA 8.0 and cuDNN 5.1 installed. Our implementation is based on the framework Caffe [18].

3.1. CIFAR Dataset Classification

To train the whole DFB-Net on CIFAR-10, the learning rate starts from 0.004, and the total number of epochs is 180. Contrary to the practice in [3, 14, 15], we give more loss weight to earlier exit branches to encourage more discriminative feature learning in side branches [11]. We attach loss weight 2.5 to both side branches and 0.25 to main branch. We don't use dropout [19] in the whole DFB-Net training. Table 2 shows the forward inference results of DFB-Net. We see that DFB-Net outperforms its baseline model when exit thresholds are set to (0.99, 0.975, 0.75) and gains 3x speedup. When we raise exit thresholds to (0.99, 0.99, 0.75), DFB-Net achieves state-of-the-art result with an error rate of 3.07% and still gains 2.85x speedup. Compared with B-ResNet [11], our DFB-Net is distinctly superior in three measures: (1) accuracy (79.19% vs. 96.93%), (2) speedup gains (1.9x vs. 2.85x), and (3) ratio of exit samples at the shortest branch (41.5% vs. 80.0%).

To train the whole DFB-Net on CIFAR-100, we apply dropout within each building block follows the practice in [7], and the learning rate starts from 0.025 for total 200 epochs. We place loss weight 3.75 on the first exit branch, 2.5 on the second exit branch, and 0.25 on the main branch. Notice that down-sampling is implemented by 2×2 average pooling with stride 2, then followed by the 1×1 and 3×3 convolutions with stride 1, not the strategy used in [5, 7, 6]. Table 3 shows the forward inference results and establishes the same fact that DFB-Net outperforms its baseline model again. As we can see that DFB-Net achieves lower error rates than the main branch does, when exit thresholds are set to (0.8, 0.75, 0.75), and gains 2.75x speedup. When we raise exit thresholds to (0.99, 0.99, 0.75), DFB-Net achieves state-of-the-art result with the error rate 16.01% and still gains 1.56x speedup.

Performance Results : CIFAR-10						
Network Topology	Exit Thresholds (Exit-1, Exit-2, Exit-3)	Error (%)	Time (ms)	Gain (x)	Exit Ratio (%) (Exit-1, Exit-2, Exit-3, Fused)	Error (%) within Each Branch (Exit-1, Exit-2, Exit-3, Fused)
(Baseline) WRN-50-N6-k6	N/A	3.23	29.67	1.00	N/A	N/A
DFB-Net : (Exit-1) Branch-1, 18 layers (Exit-2) Branch-2, 38 layers (Exit-3) Baseline, 50 layers	0.900, 0.900, 0.00	3.72	7.39	4.01	90.48, 5.98, 3.54	1.90, 15.72, 29.94
	0.900, 0.900, 0.75	3.63	7.43	3.99	90.48, 5.98, 2.83, 0.71	1.90, 15.72, 23.32, 43.66
	0.950, 0.950, 0.00	3.54	8.21	3.61	87.50, 7.05, 5.45	1.37, 11.21, 28.44
	0.950, 0.950, 0.75	3.39	8.22	3.61	87.50, 7.05, 4.50, 0.95	1.37, 11.21, 22.67, 40.00
	0.975, 0.975, 0.00	3.46	9.09	3.26	84.33, 8.27, 7.40	1.01, 7.86, 26.49
	0.975, 0.975, 0.75	3.29	9.14	3.25	84.33, 8.27, 6.30, 1.10	1.01, 7.86, 21.59, 39.09
	0.990, 0.975, 0.00	3.36	9.85	3.01	80.03, 11.53, 8.44	0.65, 5.98, 25.48
	0.990, 0.975, 0.75	**3.15**	9.89	**3.00**	80.03, 11.53, 7.22, 1.22	0.65, 5.98, 20.50, 37.70
	0.990, 0.990, 0.00	3.29	10.35	2.87	80.03, 9.48, 10.49	0.65, 4.11, 22.69
	0.990, 0.990, 0.75	**3.07**	10.41	**2.85**	80.03, 9.48, 9.19, 1.30	0.65, 4.11, 18.06, 38.46

Table 2. DFB-Net performance results on CIFAR-10 dataset (best view in color).

Performance Results : CIFAR-100						
Network Topology	Exit Thresholds (Exit-1, Exit-2, Exit-3)	Error (%)	Time (ms)	Gain (x)	Exit Ratio (%) (Exit-1, Exit-2, Exit-3, Fused)	Error (%) within Each Branch (Exit-1, Exit-2, Exit-3, Fused)
(Baseline) WRN-50-N6-k6	N/A	17.74	29.39	1.00	N/A	N/A
DFB-Net : (Exit-1) Branch-1, 18 layers (Exit-2) Branch-2, 38 layers (Exit-3) Baseline, 50 layers	0.75, 0.75, 0.00	18.06	10.01	2.94	78.73, 11.47, 9.80	10.91, 34.70, 56.02
	0.75, 0.75, 0.75	17.89	10.02	2.93	78.73, 11.47, 4.38, 5.42	10.91, 34.70, 38.58, 66.97
	0.80, 0.75, 0.00	17.78	10.62	2.77	75.83, 13.43, 10.74	9.75, 33.43, 54.93
	0.80, 0.75, 0.75	**17.55**	10.67	**2.75**	75.83, 13.43, 4.93, 5.81	9.75, 33.43, 37.93, 65.40
	0.85, 0.80, 0.00	17.34	11.51	2.55	72.62, 14.32, 13.06	8.39, 29.19, 54.13
	0.85, 0.80, 0.75	17.09	11.52	2.55	72.62, 14.32, 6.18, 6.88	8.39, 29.19, 37.70, 65.26
	0.90, 0.90, 0.00	16.94	13.04	2.25	68.64, 13.50, 17.86	6.98, 23.11, 50.56
	0.90, 0.90, 0.75	16.64	13.06	2.25	68.64, 13.50, 9.25, 8.61	6.98, 23.11, 35.35, 63.41
	0.95, 0.85, 0.00	16.64	13.77	2.13	62.61, 19.73, 17.66	4.87, 22.76, 51.53
	0.95, 0.85, 0.75	16.42	13.81	2.13	62.61, 19.73, 9.06, 8.60	4.87, 22.76, 36.53, 64.77
	0.99, 0.99, 0.00	16.60	18.81	1.56	50.79, 14.68, 34.53	2.30, 8.92, 40.89
	0.99, 0.99, 0.75	**16.01**	18.83	**1.56**	50.79, 14.68, 21.91, 12.62	2.30, 8.92, 27.89, 58.80

Table 3. DFB-Net performance results on CIFAR-100 dataset (best view in color).

3.2. Comparison with State-of-the-art Methods

In Table 4, we compare the error rates with state-of-the-art methods. Notice that we apply both scale and aspect ratio data augmentation, while other methods listed in this table use common data augmentation (random crops and/or horizontal flip). DFB-Net outperforms existing state-of-the-art methods on both datasets with error rates of 3.07% on CIFAR-10 and 16.01% on CIFAR-100. Even better, running on GTX 1080 with a batch size of 1, DFB-Net takes less than 10.5 ms, on average, to finish the forward inference on CIFAR-10, and less than 19 ms on CIFAR-100.

Model	Depth	Parameters	CIFAR-10	CIFAR-100
(pre-act) ResNet [6]	1001	10.2M	4.62	22.71
Wide ResNet [7]	28	36.5M	3.89	18.85
DenseNet-BC (k = 24) [8]	250	15.3M	3.62	17.60
DenseNet-BC (k = 40) [8]	190	25.6M	3.46	**17.18**
DFB-Net Baseline (ours)	50	81.1M	**3.23**	17.74
DFB-Net (ours)	18 / 38 / 50	106.2M	**3.07**	**16.01**

Table 4: Error rates (%) on CIFAR datasets compared with state-of-the-art methods.

4. DISCUSSION AND CONCLUSIONS

Several points are worth noting: (1) DFB-Net provides an intuitive, probability-based, exit-threshold setting for a flexible trade-off between inference time and accuracy. If we set an exit threshold of 0 to Exit-1, it only takes less than 5.5 ms for each forward inference, and still measures low error rates (5.74% on CIFAR-10 , 21.61% on CIFAR-100). (2) Small but complete side branches strongly encourage a large portion of test samples to exit earlier and thus get a high speedup gain for fast inference. (3) If a test sample failed to exceed any of the exit thresholds, making probability fusion provides a better collaborative prediction.

To sum up, DFB-Nets nicely exploit the strengths of early exit branches, probability fusion and deep supervision. DFB-Nets not only achieved state-of-the-art results on both CIFAR datasets but were equipped with the capability for fast forward inference. We are hopeful that future research will provide more experiment results on large-scale datasets to explore more nice properties of DFB-Nets.

5. REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, November 2015.

[3] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich, "Going deeper with convolutions," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[4] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," *arXiv preprint arXiv:1602.07261*, 2016.

[5] 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.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Identity mappings in deep residual networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 630–645.

[7] Sergey Zagoruyko and Nikos Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[8] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten, "Densely connected convolutional networks," *arXiv preprint arXiv:1608.06993*, 2016.

[9] Peng Tang, Xinggang Wang, Bin Feng, and Wenyu Liu, "Learning multi-instance deep discriminative patterns for image classification," *IEEE Transactions on Image Processing*, 2016.

[10] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al., "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[11] Surat Teerapittayanon, Bradley McDanel, and H Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *ICPR*, 2016.

[12] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[13] Jingdong Wang, Zhen Wei, Ting Zhang, and Wenjun Zeng, "Deeply-fused nets," *arXiv preprint arXiv:1605.07716*, 2016.

[14] Chen-Yu Lee, Saining Xie, Patrick W Gallagher, Zhengyou Zhang, and Zhuowen Tu, "Deeply-supervised nets.," in *AISTATS*, 2015, vol. 2, p. 5.

[15] Liwei Wang, Chen-Yu Lee, Zhuowen Tu, and Svetlana Lazebnik, "Training deeper convolutional networks with deep supervision," *arXiv preprint arXiv:1505.02496*, 2015.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[17] Alex Krizhevsky and Geoffrey Hinton, "Learning multiple layers of features from tiny images," *Master thesis, Computer Science Department, University of Toronto*, 2009.

[18] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.

[19] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting.," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.