Convolutional Neural Network Optimization Using Modified NSGA-II

Zhidong Su, Weihua Sheng* and Senlin Zhang

Abstract— **Convolutional Neural Networks (CNN) are becoming deeper and deeper. It is challenging to deploy the networks directly to embedded devices because they may have different computational capacities. When deploying CNNs, the trade-off between the two objectives: accuracy and inference speed, should be considered. NSGA-II (Non-dominated Sorting Genetic Algorithm II) algorithm is a multi-objective optimization algorithm with good performance. The network architecture has a significant influence on the accuracy and inference time. In this paper, we proposed a convolutional neural network optimization method using a modified NSGA-II algorithm to optimize the network architecture. The NSGA-II algorithm is employed to generate the Pareto front set for a specific convolutional neural network, which can be utilized as a guideline for the deployment of the network in embedded devices. The modified NSGA-II algorithm can help speed up the training process. The experimental results show that the modified NSGA-II algorithm can achieve similar results as the original NSGA-II algorithm with respect to our specific task and saves 46.20% of the original training time.**

I. Introduction

With the development of the computing capability of the hardware devices, neural networks are becoming deeper and deeper and have achieved good performance in image classification [1], object detection [2], speech recognition [3], etc. CNNs are widely used in the computer vision area and other tasks as a feaure extractor. The convolutional layers are concatenated together to make the network deeper and deeper to obtain good performance.

However, different application scenarios have different requirements for the inference speed and different embedded devices have different computational capacities. Fig. 1 shows the training time obtained from different GPU devices for 1 epoch using MINIST dataset [4]. We can observe that GeForce 1070 GPU only needs 8 seconds to run 1 epoch. It is 4 times faster than NVIDIA NX which is almost 4 times faster than NVIDIA NANO. If the deep neural network that runs fast in a high end device is deployed in a lower end device, the inference time will get longer with the same accuracy and may not be practical for real world applications. Therefore, the trade-off between the test accuracy and inference time of a specific network architecture should be considered when deploying the network to different embedded devices.

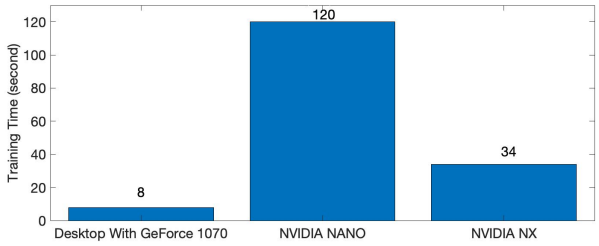

Fig. 1: Training a 3-layer CNN network with different devices for 1 epoch using MNIST dataset.

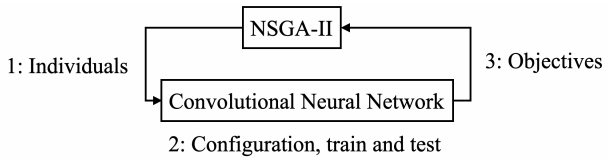

Fig. 2: The framework of the optimization process.

The network architecture has a significant impact on the accuracy and inference time. Therefore, a convolutional neural network optimization method using modified NSGA-II algorithm [5] is proposed. This method can generate an optimal network set for the trade-off between the test accuracy and inference time and speed up the training process. Fig. 2 shows the framework of the optimization method. There are three steps in this optimization process. Firstly, the

Weihua Sheng is the corresponding author and with the School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK, 74078, USA, weihua.sheng@okstate.edu. Zhidong Su is with the School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK, 74078, USA. Senlin Zhang is with the State Key Laboratory of Industrial Control Technology, Zhejiang University, Hangzhou 310027, China.

NSGA-II algorithm generates individuals for each generation. The individuals are the hyperparameters that are used to configure the network. Secondly, the network is constructed using the individuals. The network is trained and tested to obtain the final accuracy. The accuracy and number of parameters are the two objectives of the multi-objectives optimization algorithm. Finally, the two objectives are returned to NSGA-II algorithm to generate new individuals. The original NSGA-II algorithm requires to train the network for N (N is the number of individuals in each generation) times in order to obtain the second and following N individuals generation even when keeping the records of the history individuals, which makes it time-consuming when it comes to train the neural networks especially the deep neural networks. Therefore, we modified the NSGA-II algorithm to accelerate the optimization process.

II. Related Work

A. Multi-objective Optimization

Multi-objective optimization problems (MOOPs) are important in real-world application. Many real-world optimization problems can be modeled using multiple conflicting objectives. This research area has been explored by many researchers. In order to solve multi-objective optimization problems, Ines *et. al* [6] proposed a generic algorithm based on ant colony optimization. The number of pheromone trails and the number of ant colonies are utilized to parameterize the proposed algorithm. In order to solve the problem that multi-objective evolutionary algorithms (MOEAs) tend to have a computational complexity, use non-elitism approach and specify a sharing parameter, Kalyanmoy *et. al* [5] suggested a non-dominated sorting-based multi-objective evolutionary algorithms, called NSGA-II. In their algorithm, a fast non-dominated sorting approach with $O(MN^2)$ computational complexity is presented. In NSGA-II, the parent and children are combined and a selection operator is utilized to select the best N solutions. Fulya *et. al* [7] applied the genetic algorithms-based method to solve the problem of the multi-objective supply chain network, where they tried to find the set of Pareto-optimal solutions. They used the weight-based approaches in their algorithm. Ozan *et. al* [8] cast multi-task learning as multi-objective optimization to find a Pareto optimal solution. They used algorithms developed in the gradient-based multi-objective optimization literature to carry out multi-task learning.

B. Deep Neural Network Architecture Optimization

There are many ways to design the CNN architecture automatically. Designing a CNN algorithm consumes very significant time and computational resources. The most successful deep neural networks were handcrafted from scratch. It also takes the problem domain knowledge into consideration. Considering above information, Francisco *et. al* [9] proposed an algorithm to find the useful convolutional neural networks (CNNs). Their algorithm was proposed based on particle swarm optimization and able to achieve a fast convergence than other evolutionary methods. Francisco *et. al* [10] introduced a two-phase algorithm to create compact DNNs. They firstly enlarged the model size until overfitting the given dataset. After that, a pruning method is used to reduce the model size according to the user preference. Existing methods for neural network architecture optimization are mostly based on the discrete space searching. Luo *et. al* [11] proposed a continuous optimization method to improve the searching efficiency. They used an encoder to convert the architectures into a continuous space and a decoder to convert the architectures' embedding into network architectures. A predictor was used to accept the continuous representation and generate the corresponding accuracy.

III. Methodology

In this paper, we developed a modified NSGA-II algorithm to optimize the CNN network. This section will firstly introduce the CNN network that is optimized. Then the NSGA-II algorithm will be covered. After that the CNN network optimization problem is formulated. The modified NSGA-II algorithm is proposed at the end.

A. Convolutional Neural Network

The DenseNet [12] is employed as the basic network architecture to be optimized. It achieved the state-of-the-art performance in the CIFAR-10 [13] and CIFAR-100 dataset [14]. The DenseNet is composed of dense blocks. Inside the dense blocks, all layers are connected with each other directly. Fig. 3 [12] shows a three dense blocks' DenseNet. The feature of DenseNet architecture makes it easier to configure. We can vary the number of dense blocks, dense block layers and growth rate to generate new architecture. Therefore, we use DenseNet as the CNN network to be optimized.

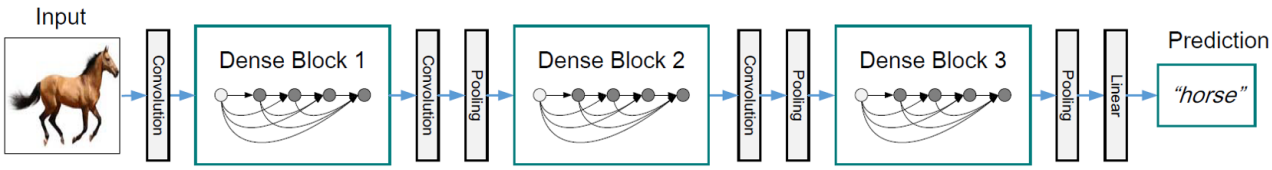

Fig. 3: A deep DenseNet with three dense blocks.

B. Multi-objective Optimization Algorithm: NSGA-II

The recognition accuracy and inference time of the CNN network are the two objectives of the optimization in this work. Table I shows the pseudo codes of the original NSGA-II [5]. We can see that

TABLE I: NSGA-II Pseudo Codes.

Step 1: Combine parent and offspring population: $R_i = P_i \cup Q_i$.
Step 2: Perform a non-dominated sorting to R_i to get F_j: F_j, j = 1, 2, ..., *etc*. all non-dominated fronts of R_i
Step 3: Set a variable $P_{i+1} = \emptyset$ to contain a new population. Set a counter j = 1. Until $\|P_{i+1}\| + \|F_j\| <$ N+1, perform $P_{i+1} = P_{i+1} \cup F_j$ and $j = j$ +1.
Step 4: Carry out a crowding-sorting operation to include (N - $\|P_{i+1}\|$) widely spread solutions in the sorted F_j to P_{i+1}.
Step 5: Generate a sub-population Q_{i+1} from P_{i+1} by using the crowded tournament selection, crossover and mutation operators. Return to Step 1 until the last generation.

NSGA-II can preserve the elite individuals and keep the diversity. After generating an offspring Q_i from parent P_i, Q_i and P_i are combined to generate R_i, where a non dominated sorting is carried out. Then the new population of size N is generated from the different non dominated fronts of R_i.

C. Problem Formulation

The goal of this paper is to find a set of optimal configurations of CNN network to maximize the accuracy and minimize the inference time. From the popular CNN networks like VGG16 [15], ResNet [16] and GoogleLeNet [17], we can observe that the number of convolutional layer, filter size, activation functions, optimizer, etc. have different degree of effect to the network performance. Therefore, they can be used as the decision variables to decide the network performance. The following part will explain the definition of the multi-objective optimization problem.

1) Objectives: There are two objectives in our task. The first objective is to maximize the accuracy or minimize the error obtained from the test dataset. The second objective is to minimize the inference time. The number of parameters can be regarded as an indicator of the network complexity and the inference time is related to the network architectural complexity. Therefore, we use the number of parameters to replace the inference time as the second objective.

2) Decision Variables: DenseNet has many hyperparameters that will affect the network performance. We chose five main hyperparameters. They are shown in Table II. In order to save the training time, the number of layers in dense block, number of dense blocks and dense block growth rate are not set to be large. According to the five decision variables, there are totally 7200 combinations. It is difficult to traverse every combination to find the optimal network set. Therefore, the NSGA-II algorithm can be used in the situation to find the Pareto front.

3) Crossover and Mutation: The individuals of each generation can be generated from the decision variables. Each individual has its genotype. The genotype has five positions that correspond the five hyperparemeters. There are two basic operations, crossover and mutation, utilized to generate new individuals from the existing individuals using the decision variables. As can be seen in Fig. 4, the first operation is crossover. In the crossover operation, two individuals are selected as the parents. The number in the genetype is corresponding to the specific value of the hyperparameter. For example, in the father genotype, the value of the second position (number of dense blocks) is 4. The digital number 4 is related to the fifth value of the number of dense blocks in Table II. It means the number of dense blocks is 5. In order to generate child 1 and child 2, firstly we choose a position randomly. Then we switch the two segments of the parents to generate the offspring. For the mutation operation, we choose a position randomly and change the value of the selected position randomly according to the variable value.

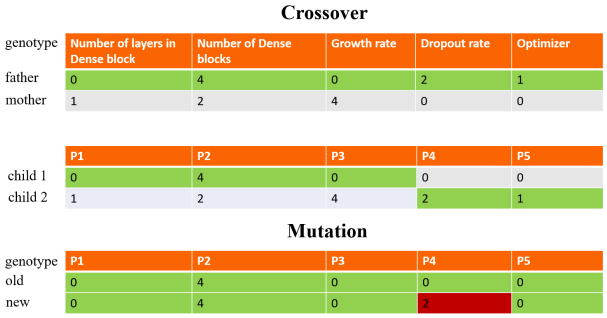

Fig. 4: Crossover and mutation.

TABLE II: Decision variables.

Variables	Values
Number of layers in Dense block	2, 3, 4, 5, 6, 7
Number of Dense blocks	1, 2, 3, 4, 5
Dense Block growth rate	4, 6, 8, 12, 14, 16,18, 20
Dropout rate	0.0, 0.1, 0.2, 0.25, 0.3, 0.5
Optimizer	adam, sgd, adagrad adamax, nadam

D. Proposed Algorithm

1) Modified NSGA-II: From Table I, we can observe that when applying NSGA-II to optimize the CNN network, in order to generate the next generation, the population size of R_i is $2N$. It means that the CNN network needs to be trained for $2N$ times for each generation without keeping the training history. This is very time-consuming when training neural networks. In order to save the training time, the genotype of each individual and the corresponding accuracy and number of parameters can be recorded. If the newly generated individuals have the same genotypes as the previous individuals, we can use the recorded history directly. By using this method, the CNN network need to be trained for N times for each generation. However, when the neural networks are very deep and the input image size are lager like $224\times224\times3$, it still takes a long time. Besides, if we want to obtain a better result, we have to train the model for many epochs. The original NSGA-II method is time-consuming. Therefore, we modified the NSGA-II algorithm to speed up the training process. Table III shows the modified the NSGA-II algorithm. Different from the original NSGA-II algorithm, when generating new individuals, the modified algorithm just perform a non-dominated sorting to P_i without generating Q_i. Keep all the individuals in the first front (e.g. n individuals), which is used to preserve the elite individuals in each generation. In order to increase the variation of the reserved individuals in the first front, the mutation operation is utilized. The first half of the remaining (N - n) individuals are generated randomly and the second half are generated from the current population. The modified NSGA-II algorithm will reduce the number of training times from N to (N - n) for each generation because we only need to train the networks for the remaining individuals to generate the next N individuals generation. When the number of individuals in the first front becomes larger in the future generations, the number of the required new individuals will become smaller and the training time will be further reduced.

TABLE III: Modified NSGA-II.

Input: The population of the current generation P_i.
Output: The population of the next generation P_{i+1}.

Step 1: Perform a non-dominated sorting to P_i and identify different fronts:
F_i, i = 1, 2, ..., etc;
Step 2: Set new population $P_{i+1} = \emptyset$. Keep all individuals in the first front to preserve the elite individuals:
$P_{i+1} = F_1$;
Step 3: Mutate the elite individuals in P_{i+1} at the probability of p.
for k in range(0, length(P_{i+1})):
if random.uniform(0, 1) $< p$:
$P_{i+1}[k]$ = mutate($P_{i+1}[k]$);
Step 4: Set m = (length(P_i) - length(P_{i+1}))/2. Generate m new individuals randomly:
while length(P_{i+1}) $<$ length(P_i) - m:
Randomly generate an individual I_a;
$P_{i+1} = P_{i+1} + I_a$;
Step 5: Generate another m new individuals from P_i:
while length(P_{i+1}) $<$ length(P_i):
Use crowded tournament selection to select two parents pr_1 and pr_2 from P_i;
I_a, I_b = crossover(pr_1, pr_2);
I_a= mutate(I_a);
I_b= mutate(I_b);
$P_{i+1} = P_{i+1} + I_a + I_b$;
Step 6: Return P_{i+1}.

2) The Whole System: Fig. 5 shows the whole system that we proposed to optimize the CNN architecture. 1) The modified NSGA-II algorithm provides the genotype of the individuals. 2) By using the hyperparameters, the DenseNet configures the network architecture and train the model. After the training procedure, the network will be tested on the test dataset to get the accuracy. 3) Return the two objec-

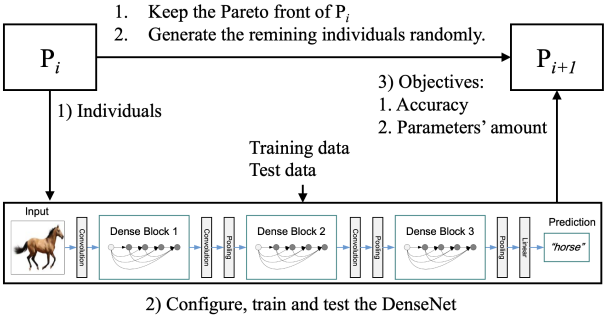

Fig. 5: Overall architecture of the proposed method.

tives: test accuracy and the amount of the parameters with respect to the genotype. The modified NSGA-II algorithm will utilize the two objectives to perform non-dominated sorting to generate a new generation.

IV. Experimental Evaluation

We implemented the proposed algorithm using Python programming language and TensorFlow deep learning framework. This section presents the experiments and evaluations of this system.

A. Experimental Setup

1) Training and test Dataset: The CIFAR-10 dataset is used to train and test the network. This dataset has 50,000 images for training and 10,000 images for test purpose. The images are 32×32 RGB images in 10 classes.

2) Parameter Setting and Training Hardware: For the DenseNet, the batch size is set to 64. For each individual, in order to save the training time, the training epoch is set to 15. For the modified NSGA-II algorithm, the group size is set to 30 and we run the program for 5 generations. We use the Google Cloud GPU NVIDIA Tesla P100 to speed up the training process.

3) Results and Analysis: Fig. 6 shows the obtained results of the individuals of different generations. The red dots are the dots in the Pareto front. From Generation 1 to Generation 5, we can observe that the individuals are moving towards the left-bottom side and the left part of the Pareto front goes up gradually. The best accuracy obtained from the Pareto front of generation 5 is 83.13%. The corresponding number of parameters is 1.94 million. Its genotype is as follows. Number of layers in Dense block: 7, Number of Dense blocks: 4, Growth rate: 20, Dropout rate: 0.1, Optimizer: adam. The best result obtained from the DenseNet paper is 94.23%. It has 7 million parameters. The accuracy is reduced by 11.78% but

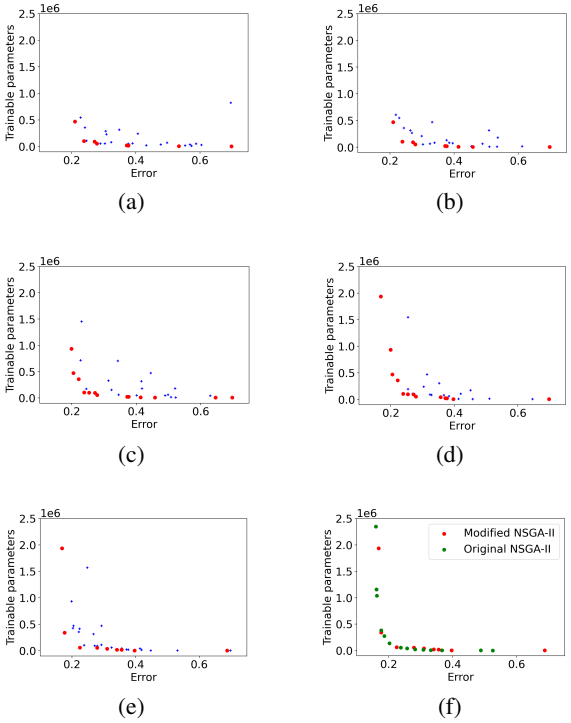

Fig. 6: Pareto front of DenseNet trained from CIFAR-10: (a) – Generation 1; (b) - Generation 2; (c) - Generation 3; (d) - Generation 4; (e) - Generation 5; (f) - Comparison with Generation 5 obtained from original NSGA-II.

the speed is 3.61 times faster. The decision variables are not set to be too large because of the limitation of the computational resource. It limits the size of the network which reduces the test result.

Fig. 6 (f) shows the comparison of the Pareto front of Generation 5 between the modified and original NSGA-II. The original NSGA-II algorithm is used to train the DenseNet to get the Pareto front. The dataset is CIFAR-10. It also utilizes 30 individuals for each generation and the algorithm is trained for 5 generations. It uses the same GPU. We can see that they have similar results. Especially the two Pareto fronts overlap each other when the number of parameters is less than 0.5 million. When the number of parameters is larger than 1.0 million, the modified NSGA-II result is sparser than the original NSGA-II, which has 3 individuals. The smallest error of the modified NSGA-II is 16.87% and the corresponding number of parameter is 1.94 million. The smallest error of the original NSGA-II is 16.01% and the corresponding number of parameter is 2.35 million, where the accuracy is improved by 0.86% but the number of parameter is 1.2 times larger than that of

the modified NSGA-II.

For the training time, by using the same dataset, hyperparameters and hardware, the original NSGA-II algorithm takes 17 hours and 32 minutes to run for 5 generations, while the modified NSGA-II algorithm takes 9 hours and 26 minutes, which saves 46.20% of the original training time.

The proposed algorithm can speed up the training process and provide the Pareto front for different devices with respect to the trade-off between the accuracy and the inference time. However, there are still some drawbacks in the current work. In order to speed up the training process, the proposed algorithm sacrificed part of the ability to explore the sample space. It is also not fully evaluated in different dataset and neural networks.

V. Conclusion and Future Work

In this paper, we utilize NSGA-II algorithm to optimize DenseNet architecture to obtain the Pareto front. It can be used for the trade-off between the accuracy and the inference time of a specific network architecture when deploying the network to different embedded devices. In order to reduce the training time, we modified the NSGA-II algorithm. The test results show that the modified NSGA-II algorithm achieved similar results as the original NSGA-II algorithm with respect to our specific task and saves 46.20% of the training time. In the future work, we will evaluate the proposed algorithm in different datasets and neural networks and train the algorithm using real world data like people detection datasets and design the application strategy to deploy the algorithm on different embedded devices guided by the Pareto front.

Acknowledgement

This project is supported by the National Science Foundation (NSF) Grants CISE/IIS 1910993, EHR/DUE 1928711, the Joint Fund of Chinese Ministry of Education for Pre-research of Equipment under Grant 6141A02022371, and the Open Research Project of the State Key Laboratory of Industrial Control Technology, Zhejiang University, China (No. ICT2021B14).

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 84–90, May 2017.

[2] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[3] D. Amodei, S. Ananthanarayanan, and R. Anubhai, "Deep speech 2 : End-to-end speech recognition in english and mandarin," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 173–182. [Online]. Available: http://proceedings.mlr.press/v48/amodei16.html

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[5] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[6] I. Alaya, C. Solnon, and K. Ghedira, "Ant colony optimization for multi-objective optimization problems," in *19th IEEE International Conference on Tools with Artificial Intelligence(ICTAI 2007)*, vol. 1, 2007, pp. 450–457.

[7] "A genetic algorithm approach for multi-objective optimization of supply chain networks," *Computers and Industrial Engineering*, vol. 51, no. 1, pp. 196 – 215, 2006.

[8] O. Sener and V. Koltun, "Multi-task learning as multi-objective optimization," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018, pp. 527–538.

[9] "Particle swarm optimization of deep neural networks architectures for image classification," *Swarm and Evolutionary Computation*, vol. 49, pp. 62 – 74, 2019.

[10] F. E. Fernandes and G. G. Yen, "Automatic searching and pruning of deep neural networks for medical imaging diagnostic," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–11, 2020.

[11] R. Luo, F. Tian, T. Qin, and T. Liu, "Neural architecture optimization," *CoRR*, vol. abs/1808.07233, 2018. [Online]. Available: http://arxiv.org/abs/1808.07233

[12] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016. [Online]. Available: http://arxiv.org/abs/1608.06993

[13] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar, "Do CIFAR-10 classifiers generalize to cifar-10?" *CoRR*, vol. abs/1806.00451, 2018. [Online]. Available: http://arxiv.org/abs/1806.00451

[14] A. Krizhevsky, "Learning multiple layers of features from tiny images," Master's thesis, Department of Computer Science, University of Toronto, Toronto, Canada, 2009.

[15] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1409.1556

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Computer Vision and Pattern Recognition (CVPR)*, 2015. [Online]. Available: http://arxiv.org/abs/1409.4842