Checkpoint Classifier for CNN Image Classification*

Jackson H. Paul[1] and Andy D. Digh[2]
[1]Electrical and Computer Engineering Department
jackson.h.paul@live.mercer.edu
[2]Computer Science Department
Mercer University
Macon, GA 32107
digh_ad@mercer.edu

Abstract

Given a large convolutional neural network (CNN) with hundreds of layers, when can the input data be correctly classified? How many layers does each image require? We propose an architecture with a mid-network classifier to classify certain images at earlier points in the model. When the network is very confident about an image, having high activations, then that individual image will be classified early. The number of computations and the average number of convolutions will be reduced if certain images can be classified earlier. In addition, the mid-network classification task is more difficult because less features have been extracted at earlier points in the network. Thus, the output and mid-network classifier will work together to correctly classify each image as fast as possible while preserving the accuracy. This proposed method has been implemented into well known computer vision architectures, like ResNet and GoogLeNet Inception. We have achieved large runtime improvements while limiting the accuracy degradation.

1 INTRODUCTION

Convolutional neural networks (CNN) have vast amounts of parameters in order to learn hundreds of feature maps for each layer [1, 2]. Although the number

*Copyright ©2023 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

of parameters can be large, convolutional models can perform extremely well and have progressed from accurate handwritten digit classification to real-time, efficient object detection [3, 4]. This paper [4] shows the general trend of how the size of the model increases as the accuracy increases. The winning submissions of the famous ImageNet Large Scale Visual Recognition Challenge have increased in size as well, producing much larger networks each year to achieve better performance [5, 6].

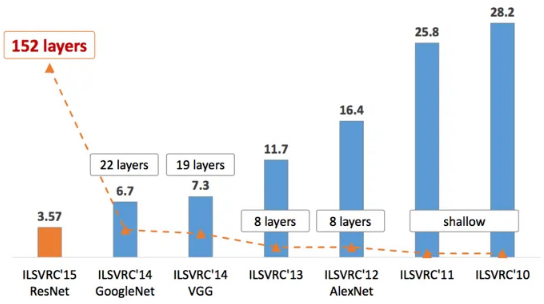

Figure 1: Number of layers (dotted line) and test error (bars) for each winning model of ImageNet Large Scale Visual Recognition Challenge from 2010 to 2015 [6].

The motivation of this paper comes from numerical analysis; we have iterative techniques to solve all types of complex mathematical operations, including integration and differential equations [7]. Iterative techniques are great for modern computers because they can iterate forever. When using these numerical algorithms, thresholds are used to determine when the answer is 'good enough'. Once the answer is below a certain threshold of precision, the algorithm is finished and the answer is returned. Using these concepts, the same question was asked in the context of CNNs; "When is the input data processed enough to be classified correctly?"

We propose a CNN architecture with a checkpoint in order to attempt to classify the input data earlier in the model. If the input data can be classified, then the model will stop, and output the results. If not, the model will continue and use the remaining layers as intended. The network will naturally be more and less confident about certain images, so only the most confident images will be classified early. In addition, less features have been extracted at earlier points in the network, so the checkpoint classifier should perform worse than the overall output classifier. Thus, the output classifier and the checkpoint classifier will work together to classify images faster while retaining accuracy. Theoretically, if certain inputs could be classified earlier in the network, then the total average number of convolutional layers would be reduced, and the runtime would be improved while minimizing accuracy degradation.

2 RELATED WORKS WITH CNNs

Starting with LeNet, with some exceptions, CNNs usually have a basic, predictable structure [8]. First, a set of convolutional layers to extract features, then fully connected layers to classify the input based on the features, in that specific order [2]. This proposed architecture would differ by having a variable number of layers and a second fully connected classifier network that interrupts the series of convolutional layers.

In [9], they describe using two independent classifiers to have redundancy in case one classifier was incorrect. They both perform the classification after the convolutional layers using the same data, but utilize two different approaches to protect against an incorrect classification method. In addition, one of the most famous papers, "Going Deeper with Convolutions," describes the GoogLeNet Inception network [10]. This network uses two mid-network auxiliary classifiers to protect against vanishing or exploding gradients and provide some regularization during training [11]. The network has a traditional set of fully connected layers for classification but aggregates all output losses to perform backpropagation. This proposed architecture has additional sets of fully connected layers but they perform a separate classification that does not combine or contribute to the original output classifier.

3 METHODS

Overview — First, the general context and overview will be discussed and then each individual step will be described. Python and PyTorch were used with the built in CIFAR10 and CIFAR100 datasets. These image classification datasets have ten and 100 output classes, respectively, and have 50,000 training images and 10,000 test images [12]. For the following sections, the base model refers to the unchanged, published architecture, whereas the custom model is the published base model with the extra mid-network classification layer added.

Published Architectures — The first step is training the base model on the given dataset and saving that model to be used for testing. PyTorch has several trained architectures built in, but saves models as a dictionary. So, the models need to be trained from scratch in order to allow new layers to be added after training and for layers to be added to specific places in the sequential order.

Looking at Table 1, the accuracy of the published networks were not particularly optimized using learning rate, batch size, or other training hyperparameters. Firstly, the graphics card used had memory limitations, so the larger models needed a small batch size to fit on the card. This prevented the larger models from generalizing the classification and resulted in overfitting and high losses on the testset. In addition, the parameters learned during the

Table 1: Base model testset accuracies with both datasets, CIFAR10/100.

Model	CIFAR10	CIFAR100
AlexNet	82.47%	48.24%
VGG-16	92.28%	69.34%
ResNet34	82.25%	52.47%
ResNet50	84.46%	56.78%
InceptionV1	86.00%	61.62%

base model training are used for both models because the base model layers are unchanged; only a new layer with the mid-network classification is added to the custom model. Therefore, any accuracy and runtime changes are relative to the base model and that difference is the important metric. Base models could have been trained better, but all base model parameters are shared with the custom model so the results will still show the validity of this proposed method.

Mid-network Classification Model — This model was used as a separate classifier and trained using mid-network activations extracted from the implementation layer. After the base model training finished, a custom layer was added to the base model to write the mid-network activations and correct labels to a binary file. Multiple layers were tested for each base model and dataset so activations from each specific layer were collected and stored for training. The layers were chosen to be roughly 40% to 60% of the overall network such that some features have been extracted for a successful classification, but enough layers remain to improve the runtime.

This model was trained on all 50,000 trainset inputs and used a $90\% - 5\% - 5\%$ train, development, and testset split, respectively. The network used two fully connected layers, size 2,048 and 1,024, and each layer used a dropout of 70% and batch normalization [19, 20, 21, 11]. The second layer outputs ten or 100 classes depending on which dataset was being used [22]. For the input, this architecture was used for all base models, so the feature map sizes will change accordingly. Thus, average pooling was used to downsize the feature maps, choosing a pooling size to have about 3,000 to 4,000 values after flattening the feature maps. Using about 3,000 to 4,000 input values was enough information to classify the inputs but still retain a fairly small overall network. Although, there were 1 or 2 cases that used 4,608 as the input size because due to the number of feature maps, $\sim$4,600 was the closest to the desired range. Although marginally better classification could be achieved with individual optimization for each base model and dataset combination, this architecture performed well and we decided to keep it constant for more consistent results.

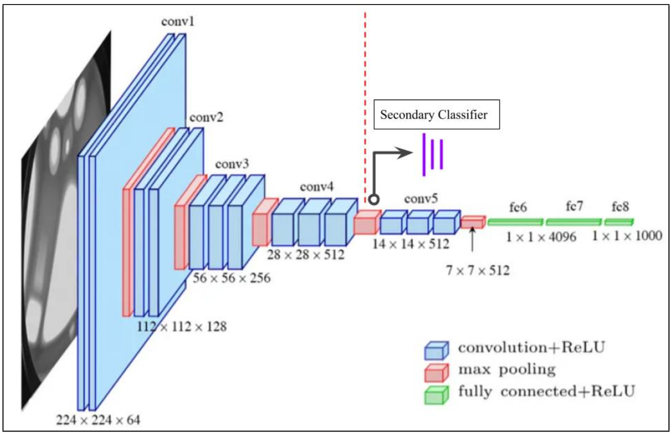

Figure 2: Mid-network classification model implementation into VGG-16 base architecture as checkpoint classifier [13]. For the functions, the activation function was Rectified linear unit (ReLU); Adam optimizer was used with a weight decay of $4 * 10^{-5}$; the loss function was Cross Entropy Loss [14, 15, 16]. Using Adam optimizer helped the model converge faster than traditional stochastic gradient descent [17]. The final output layer did not use a softmax function because the unrestricted activation values were used to find the threshold value [18].

Parameter Search — A maximum value threshold is used to determine if the mid-network classification will be kept or not. The mid-network classification model will attempt to classify every image, but only activations above the max threshold will be classified early. In addition to the max threshold, the number of inputs let in and processed early was also optimized as a threshold, this percentage is the let in threshold. The best max value threshold is found for each let in threshold for the given layer and base model. At a given let in threshold, the “best" max value was found by iterating through values, at a specified step, and finding the percentage of images above the max threshold and the percentage correct and above the max threshold. Those values were divided to find the relative accuracy of inputs above the let in threshold.

Although, in order for a max threshold to be optimized for each let in threshold, the percentage of images above the max threshold is constrained between the let in threshold plus 10% and the let in threshold. Thus, as less images are let in, mid-network classification model can be more selective so relative accuracy increases. The network chooses the highest activations, and only classify images it is the most confident about. Conversely, as more images are let in, the network becomes less confident and the relative accuracy decreases. This is a helpful property because the let in threshold will converge to letting in less images, thus optimizing down to the let in threshold for each constrained range.

Timed Testing — For the final results, the base and custom models were tested against each other to observe the accuracy and runtime changes. The custom layer added includes the mid-network classification network and max value threshold to determine if the given image will be classified early. The models were tested using the CIFAR10/100 testset and ran alternating sets of epochs to ensure the computer switches tasks. A batch size of one was used for all testing to simulate real world scenarios. Also, two warm up epochs of each model are run before testing to warm up the gpu and system. The data was collected using two sets of five epochs, alternating between each set. Some of the smaller models were tested with three sets of five epochs because the faster runtimes were less consistent.

Implementation — At the specified layer, the image tensor is first average pooled down to the desired image size, passed through the classification model, and the max activation is checked against the max value threshold. The custom layer will return the input image tensor, the output of the classification model, and a boolean value. If the max activation is higher than the threshold, the boolean value will toggle to true. In the forward function of the model, the boolean is checked and if true, the model will return the output. If not, the input image tensor is unchanged and is passed into the next layer like usual.

4 RESULTS AND DISCUSSION

In this section, the runtime and accuracy changes will be shown from all models to display the validity of the proposed method. Multiple layers were tested for each model on both datasets, ranging from various points in the networks to find the best position for a classification checkpoint. Sections of this data will be presented and an online appendix of all layers and models is available by request. The accuracy changes were measured by subtracting from the base model accuracy to observe difference and runtime changes were measured by dividing the base model and custom model time to observe percent change. CIFAR10/100 results, shown in table two below, for an individual layer implemented in AlexNet, VGG-16, and ResNet50 [22, 23, 24]. The progression of runtime and accuracy changes can be observed for the series of let in thresholds. Looking at AlexNet, the accuracy drops off as more images are let in on CIFAR10, but the accuracy is very stable on CIFAR100 with slight accuracy improvements. The runtime did not improve very much because it is a very small model, it takes more than 80% of images to be classified early to provide a benefit. Despite being a small model, only five convolutional layers, the mid-network classification performed well with limited accuracy degradation on both datasets.

Table 2: Runtime and accuracy changes with AlexNet, VGG-16 and ResNet50 on both datasets on select layers. For accuracy changes, a positive value is an increase in overall accuracy, whereas a negative value is a decrease.

	Layer 14			**Layer 20**			**Layer 26**		
Dataset	Let-In Thres.	Accuracy Change	Runtime Change	Let-In Thres.	Accuracy Change	Runtime Change	Let-In Thres.	Accuracy Change	Runtime Change
CIFAR10	0.600	0.480	-4.19%	0.200	-0.180	-0.60%	0.102	-0.020	-0.36%
	0.700	0.310	-0.91%	0.300	-0.580	**3.46%**	0.201	-0.190	**5.54%**
	0.800	0.190	**1.70%**	0.400	-1.010	**6.90%**	0.300	-0.590	**10.66%**
	0.900	-0.060	**4.63%**	0.502	-1.710	**11.16%**	0.400	-1.010	**15.80%**
	1.000	-1.080	**7.84%**	0.601	-2.800	**15.36%**	0.501	-1.760	**20.68%**
CIFAR100	0.600	0.880	-6.75%	0.200	-1.610	-1.04%	0.102	-1.070	-0.71%
	0.700	1.040	-3.56%	0.300	-2.890	**2.76%**	0.200	-2.830	**4.72%**
	0.800	1.220	-0.32%	0.400	-4.520	**6.60%**	0.300	-5.110	**10.22%**
	0.901	0.920	**3.50%**	0.500	-6.640	**10.88%**	0.401	-7.680	**15.23%**
	1.000	0.240	**7.60%**	0.600	-9.110	**15.54%**	0.501	-10.81	**20.74%**

Comparatively, VGG-16 and ResNet50 are larger networks, thirteen and 49 convolutional layers respectively. Thus, there is a greater potential for runtime improvements, both models needing only 20–30% of images to observe runtime improvements. A general trend can be seen, as more images are classified early, the overall accuracy decreases. This is expected because the output classifier is larger and more features have been extracted after more convolutional layers.

With VGG-16, the runtime has much larger improvements but the accuracy degrades much faster compared to AlexNet. The accuracy degradation is very steep in CIFAR100, almost 10% accuracy loss with only 60% classified early. Although, larger runtime improvements can also be observed; VGG-16 achieving 15% faster results compared to the base model. Overall, VGG-16 did not perform well due to the large accuracy degradation, but there are large potential runtime improvements.

Similar to VGG-16, ResNet50 displayed very promising runtime improvements with over 20% faster on both datasets with only 50% classified early. Although, the classification was not very accurate and produced significant amounts of accuracy degradation. On the CIFAR10 dataset the degradation was much less, but became very high on CIFAR100. For VGG-16 and ResNet50, due to graphics card limitations, small batch sizes were used: 64 & 100, respectively. Thus, base models did not train very well and negatively impacted mid-network classification as a result. Nevertheless, both still demonstrate potentially large runtime improvements of a mid-network classifier.

With 33 convolutional layers, ResNet34 provides a deep enough network to accurately train and provide sufficient depth to improve runtime [24]. The mid-network classification was implemented after the fourteenth, twentieth, and 26th convolutional layers. Table 3 above shows the full results of each layer

Table 3: Runtime and accuracy changes with AlexNet, VGG-16 and ResNet50 on both datasets on select layers. For accuracy changes, a positive value is an increase in overall accuracy, whereas a negative value is a decrease.

ResNet34 CIFAR100 Results Per Layer

	Layer 14			Layer 20			Layer 26	
Let-In Thres.	Accuracy Change	Runtime Change	Let-In Thres.	Accuracy Change	Runtime Change	Let-In Thres.	Accuracy Change	Runtime Change
0.000	0.000	-7.76%	0.000	0.000	-7.12%	0.000	0.000	-8.99%
0.136	0.070	-3.87%	0.135	-0.040	-4.02%	0.102	-0.010	-6.27%
0.218	0.180	-1.02%	0.203	-0.100	-2.42%	0.255	0.030	-6.00%
0.301	0.160	**2.38%**	0.303	-0.060	-1.22%	0.348	0.050	-4.49%
0.416	0.170	**5.783%**	0.408	0.000	**0.059%**	0.415	0.020	-3.94%
0.501	0.300	**7.21%**	0.505	-0.060	**2.43%**	0.514	0.020	-2.00%
0.604	0.430	**11.46%**	0.601	-0.070	**3.57%**	0.612	0.060	-2.18%
0.701	0.480	**14.75%**	0.701	-0.130	**3.79%**	0.701	0.010	-0.09%
0.801	0.740	**18.83%**	0.800	-0.080	**7.44%**	0.801	0.160	**0.02%**
0.901	0.640	**24.33%**	0.900	-0.070	**9.54%**	0.901	0.180	**1.93%**
1.000	-0.680	**36.52%**	1.000	-0.380	**21.60%**	1.000	-0.290	**10.43%**

tested at each ~10% let in threshold interval. The sequence of the runtimes can be seen, each layer requiring more images in order to overcome the classification cost. Moreover, each layer has progressively faster runtimes as the secondary clarifier is implemented earlier in the network.

Starting from the right, layer 26 exhibited adequate classification with no accuracy loss. Although, it was too far into the network to produce major runtime improvements, needing about 80% of early classification to offset the cost of the secondary classifier. Layer 20 addressed these issues with negligible accuracy loss while providing much larger runtime improvements. With a let in threshold of 90%, layer 20 had a 0.07% loss of accuracy and a 9.5% faster runtime. Lastly, layer 14 was still able to perform an excellent classification while greatly increasing the runtime. In addition, the accuracy did not degrade until the let in threshold was 100%; with all images being classified early, the network is 36% faster and 0.68% lower accuracy.

Similar to ResNet34, the InceptionV1 network performed exceptionally well providing large runtime improvements and very small accuracy degradation [10]. Three separate points were tested, after the 4a inception block, 4c inception block, and 4e inception block. Figure 3 above presents the results for each layer. For the runtime, the results show a progressively greater improvement as the images are classified at earlier points in the network. Each block produces a faster runtime then the later blocks because there are more convolutions to be skipped. In addition, the runtime decreases as more images are let in because the average number of convolutions is reduced; all lines trend upwards as the let in threshold increases. As seen on the other models, the cost of adding a mid-network classification to a model is about 5% of the runtime and about

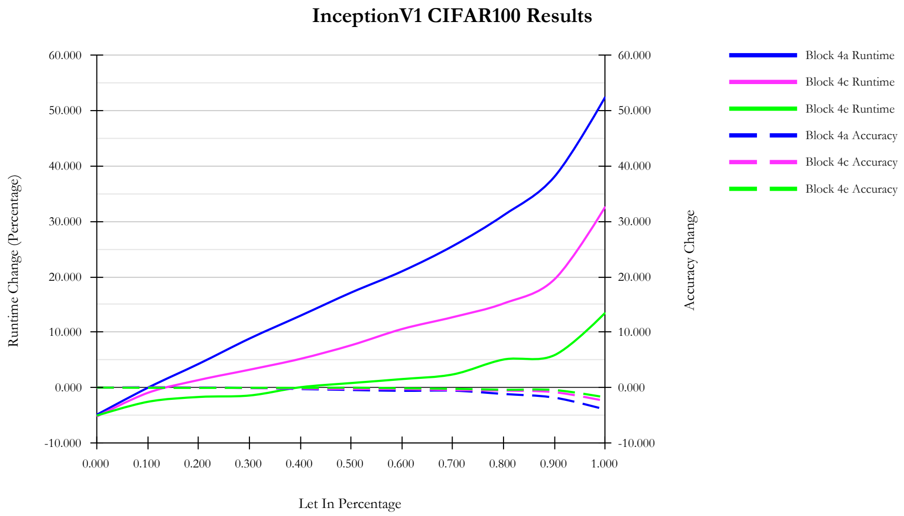

Figure 3: InceptionV1 results on CIFAR100 dataset after the 4a, 4c, and 4e inception blocks. The InceptionV1 network utilized blocks with multiple convolutions known as an inception block. The base architecture has a total of nine inception blocks; reference paper for architecture diagram [10].

ten to 40% of images need to be classified early to overcome this cost.

For the accuracy, the trends are also as expected in which each block's accuracy degrades as more images are let in and the later blocks degrade less than the earlier blocks. InceptionV1 performed very well overall with a maximum of 15% to 55% faster runtimes and less than 4% accuracy degradation at the highest let in threshold. Like AlexNet and ResNet34, the mid-network classification was very accurate and the accuracy loss remained negligible, or even slightly positive, until very high let in thresholds. At around 0.5% of accuracy loss, block 4a's runtime improved by 25%, block 4c by 15%, and block 4e by 6%, at 70%, 80%, and 90% let in threshold respectively.

5 CONCLUSION

We have investigated the use of an independent, secondary classifier implemented into published architectures for faster classification on CIFAR10/100 datasets. These findings indicate that mid-network classification can give a marginal increase in accuracy, while at the same time, significantly reducing the overall runtime of the model. The specific implementation in the network does need to be surveyed to produce the best results, providing enough information for the classification and enough layers to reduce convolutions. Further investigation should be conducted to assess which types of images or classes benefit from the secondary classifier, if there is any pattern or predictability to the classification improvement. Although these concepts were not verified using the expansive ImageNet dataset, there is sufficient evidence to suggest similar results will be yielded. Therefore, additional experiments should be carried out to investigate the utilization of multiple independent classifiers.

References

[1] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*, 2015.

[2] Matthew D. Zeiler and Rob Fergus. *Visualizing and Understanding Convolutional Networks*. 2013.

[3] Y. LeCun, K. Kavukcuoglu, and C. Farabet. Nano-bio circuit fabrics and systems. *IEEE International Symposium on Circuits and Systems*, pages 253—-256, 2010.

[4] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. 2017.

[5] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. *ImageNet Large Scale Visual Recognition Challenge*. 2015.

[6] S. Das. *CNN architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and More*. Medium, 2019.

[7] R. Burden and J. Faires. *Numerical Analysis*. Cengage Learning, 2016.

[8] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. *Proceedings of the 25th International Conference on Neural Information Processing Systems*, 1:1223–1231, 2012.

[9] S. Gehlot, A. Gupta, and R. Gupta. SDCT-AuxNet$^{\theta}$: DCT augmented stain deconvolutional CNN with auxiliary classifier for cancer diagnosis. 2020.

[10] C. Szegedy. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[11] Jan Kukačka, Vladimir Golkov, and Daniel Cremers. *Regularization for Deep Learning: A Taxonomy*. 2017.

[12] A. Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. 2009.

[13] K. Le. *An overview of VGG16 and NiN models*. Medium, 2021.

[14] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pages 807—-814, 2010.

[15] Abien Fred Agarap. *Deep Learning using Rectified Linear Units (ReLU)*. 2019.

[16] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017.

[17] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2017.

[18] Tim Pearce, Alexandra Brintrup, and Jun Zhu. *Understanding Softmax Confidence and Uncertainty*. 2021.

[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929—-1958, 2014.

[20] N. Srivastava. *Improving Neural Networks with Dropout*. 2013.

[21] S. Ioffe and C. Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60:84–90, 2017.

[23] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2015.

[24] K. He, S Zhang, and J. Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.