# Modified ResNet
# A comparison between three proposed ResNet model

**Written by Karan Vora, [1] Parisima Abdali , [2] Ajay Tibrewal [3]**

New York University
kv5124@nyu.edu, pa2297@nyu.edu, at5632@nyu.edu
github.com/modifiedResNet

**Abstract**

In this project, we introduce three modified versions of the ResNet model with the primary goal of achieving high accuracy while adhering to a constraint of less than 5 million trainable parameters. To achieve this objective, we conducted extensive experiments, leveraging different parameters and techniques, including active learning rate, efficient optimizer, layer augmentation, and dropouts. We meticulously analyzed the results of our experiments, presenting the findings in this paper. The outcomes of our work may have implications for various applications where model size and computational resources are critical factors.

## Introduction

ResNets, or residual neural networks, are a type of deep learning architecture that have been widely used in computer vision tasks such as image classification, object detection, and segmentation. They introduce residual connections that allow information to bypass layers and flow directly to the next layer, which has shown to improve the performance of deep networks. In this project, we developed and trained a ResNet architecture on the CIFAR 10 dataset (Krizhevsky 2009) with the goal of achieving high accuracy while adhering to a constraint of less than 5 million trainable parameters. To understand the effect of parameter scaling, we experimented with three different architectures, ranging from a small ResNet with 78 thousand parameters to a larger architecture with 4.9 million parameters. We utilized data visualization techniques to analyze the results of scaling. Batch normalization was integrated to improve the training of deep networks. Various optimizers were compared, and it was determined that Stochastic Gradient Descent with momentum, including Nesterov acceleration to regulate the momentum, was the most effective approach, as reported by (Liu, Gao, and Yin 2020). Additionally, we investigated the impact of dropout layers. The final model met the constraint and demonstrated computational efficiency.

## ResNet Model

ResNet is a deep convolutional neural network (CNN) architecture proposed by Kaiming He et al. in 2015 (He et al. 2015) to address the challenge of training very deep neural networks. It introduces residual blocks with skip connections, allowing gradients to bypass layers and learn residual mappings, making optimization easier. ResNet typically consists of convolutional, batch normalization, activation, and pooling layers, followed by global average pooling and fully connected output layers. It has multiple variants with increasing depth and capacity, such as ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152. ResNet has achieved state-of-the-art performance in various computer vision tasks and is widely used in deep learning applications.

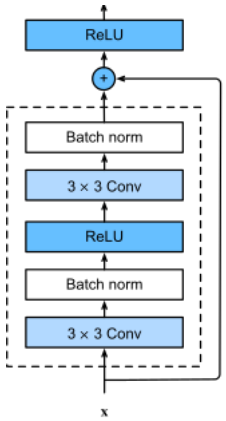


Figure 1: ResNet Block

## Network Architecture

The three models, ResNetSmall, ResNetMedium, and ResNetLarge, are variations of the ResNet architecture. The main differences between these three models are the number of layers, the number of filters (or channels) in each layer, and the size of the final fully connected layer.

### Small Network Layer Summary and Total Parameters

This model has 3 layers (referred to as ”layer1”, ”layer2”, and ”layer3”) and a final fully connected layer with 64 units (or neurons). The number of filters in the convolutional lay-

ers gradually increases from 16 to 64 as we go deeper into the network.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| Conv2d-1 | [-1, 16, 32, 32] | 432 |
| BatchNorm2d-2 | [-1, 16, 32, 32] | 32 |
| Conv2d-3 | [-1, 16, 32, 32] | 2,304 |
| BatchNorm2d-4 | [-1, 16, 32, 32] | 32 |
| Conv2d-5 | [-1, 16, 32, 32] | 2,304 |
| BatchNorm2d-6 | [-1, 16, 32, 32] | 32 |
| BasicBlock-7 | [-1, 16, 32, 32] | 0 |
| Conv2d-8 | [-1, 32, 16, 16] | 4,608 |
| BatchNorm2d-9 | [-1, 32, 16, 16] | 64 |
| Conv2d-10 | [-1, 32, 16, 16] | 9,216 |
| BatchNorm2d-11 | [-1, 32, 16, 16] | 64 |
| Conv2d-12 | [-1, 32, 16, 16] | 512 |
| BatchNorm2d-13 | [-1, 32, 16, 16] | 64 |
| BasicBlock-14 | [-1, 32, 16, 16] | 0 |
| Conv2d-15 | [-1, 64, 8, 8] | 18,432 |
| BatchNorm2d-16 | [-1, 64, 8, 8] | 128 |
| Conv2d-17 | [-1, 64, 8, 8] | 36,864 |
| BatchNorm2d-18 | [-1, 64, 8, 8] | 128 |
| Conv2d-19 | [-1, 64, 8, 8] | 2,048 |
| BatchNorm2d-20 | [-1, 64, 8, 8] | 128 |
| BasicBlock-21 | [-1, 64, 8, 8] | 0 |
| AdaptiveAvgPool2d-22 | [-1, 64, 1, 1] | 0 |
| Linear-23 | [-1, 10] | 650 |

Total params: 78,042
Trainable params: 78,042
Non-trainable params: 0

Input size (MB): 0.01
Forward/backward pass size (MB): 1.53
Params size (MB): 0.30
Estimated Total Size (MB): 1.84

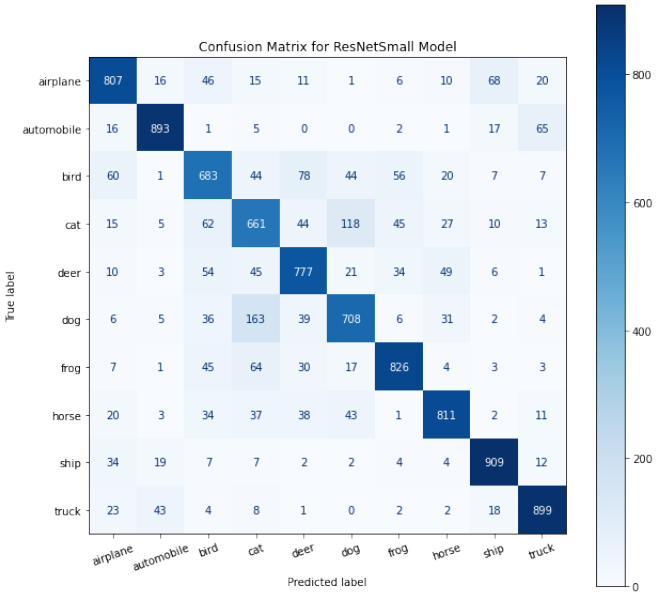


Figure 2: Confusion Matrix SmallModel

## Medium Network Layer Summary and Total Parameters

This model has 4 layers (referred to as ”layer1”, ”layer2”, ”layer3”, and ”layer4”) and a final fully connected layer with 256 units. The number of filters in the convolutional layers gradually increases from 32 to 512 as we go deeper into the network.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| Conv2d-1 | [-1, 32, 32, 32] | 864 |
| BatchNorm2d-2 | [-1, 32, 32, 32] | 64 |
| Conv2d-3 | [-1, 32, 32, 32] | 9,216 |
| BatchNorm2d-4 | [-1, 32, 32, 32] | 64 |
| Conv2d-5 | [-1, 32, 32, 32] | 9,216 |
| BatchNorm2d-6 | [-1, 32, 32, 32] | 64 |
| BasicBlock-7 | [-1, 32, 32, 32] | 0 |
| Conv2d-8 | [-1, 64, 16, 16] | 18,432 |
| BatchNorm2d-9 | [-1, 64, 16, 16] | 128 |
| Conv2d-10 | [-1, 64, 16, 16] | 36,864 |
| BatchNorm2d-11 | [-1, 64, 16, 16] | 128 |
| Conv2d-12 | [-1, 64, 16, 16] | 2,048 |
| BatchNorm2d-13 | [-1, 64, 16, 16] | 128 |
| BasicBlock-14 | [-1, 64, 16, 16] | 0 |
| Conv2d-15 | [-1, 128, 8, 8] | 73,728 |
| BatchNorm2d-16 | [-1, 128, 8, 8] | 256 |
| Conv2d-17 | [-1, 128, 8, 8] | 147,456 |
| BatchNorm2d-18 | [-1, 128, 8, 8] | 256 |
| Conv2d-19 | [-1, 128, 8, 8] | 8,192 |
| BatchNorm2d-20 | [-1, 128, 8, 8] | 256 |
| BasicBlock-21 | [-1, 128, 8, 8] | 0 |
| Conv2d-22 | [-1, 256, 4, 4] | 294,912 |
| BatchNorm2d-23 | [-1, 256, 4, 4] | 512 |
| Conv2d-24 | [-1, 256, 4, 4] | 589,824 |
| BatchNorm2d-25 | [-1, 256, 4, 4] | 512 |
| Conv2d-26 | [-1, 256, 4, 4] | 32,768 |
| BatchNorm2d-27 | [-1, 256, 4, 4] | 512 |
| BasicBlock-28 | [-1, 256, 4, 4] | 0 |
| AdaptiveAvgPool2d-29 | [-1, 256, 1, 1] | 0 |
| Linear-30 | [-1, 10] | 2,570 |

Total params: 1,228,970
Trainable params: 1,228,970
Non-trainable params: 0

Input size (MB): 0.01
Forward/backward pass size (MB): 3.28
Params size (MB): 4.69
Estimated Total Size (MB): 7.98

## Large Network Layer Summary and Total Parameters

This model has 4 layers (referred to as ”layer1”, ”layer2”, ”layer3”, and ”layer4”) and a final fully connected layer with 512 units. The number of filters in the convolutional layers gradually increases from 64 to 512 as we go deeper into the network.

| Layer (type) | Output Shape | Param # |
|---|---|---|

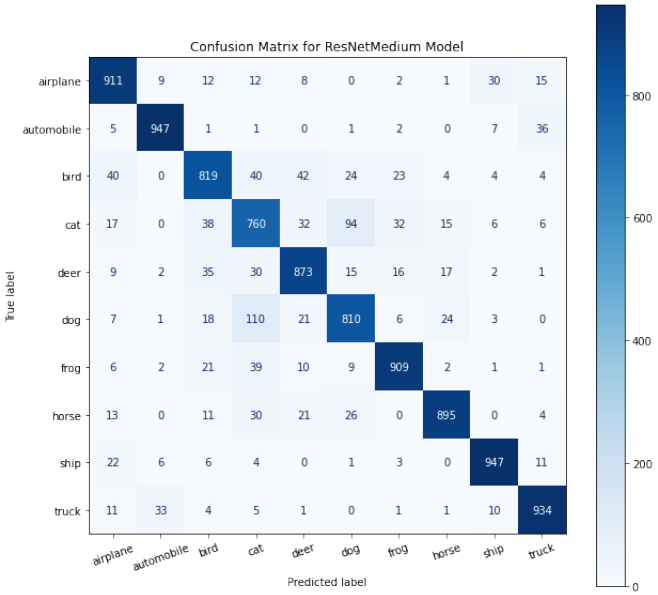


Figure 3: Confusion Matrix Medium Model

| | | |
|---|---|---|
| Conv2d-1 | [-1, 64, 32, 32] | 1,728 |
| BatchNorm2d-2 | [-1, 64, 32, 32] | 128 |
| Conv2d-3 | [-1, 64, 32, 32] | 36,864 |
| BatchNorm2d-4 | [-1, 64, 32, 32] | 128 |
| Conv2d-5 | [-1, 64, 32, 32] | 36,864 |
| BatchNorm2d-6 | [-1, 64, 32, 32] | 128 |
| BasicBlock-7 | [-1, 64, 32, 32] | 0 |
| Conv2d-8 | [-1, 128, 16, 16] | 73,728 |
| BatchNorm2d-9 | [-1, 128, 16, 16] | 256 |
| Conv2d-10 | [-1, 128, 16, 16] | 147,456 |
| BatchNorm2d-11 | [-1, 128, 16, 16] | 256 |
| Conv2d-12 | [-1, 128, 16, 16] | 8,192 |
| BatchNorm2d-13 | [-1, 128, 16, 16] | 256 |
| BasicBlock-14 | [-1, 128, 16, 16] | 0 |
| Conv2d-15 | [-1, 256, 8, 8] | 294,912 |
| BatchNorm2d-16 | [-1, 256, 8, 8] | 512 |
| Conv2d-17 | [-1, 256, 8, 8] | 589,824 |
| BatchNorm2d-18 | [-1, 256, 8, 8] | 512 |
| Conv2d-19 | [-1, 256, 8, 8] | 32,768 |
| BatchNorm2d-20 | [-1, 256, 8, 8] | 512 |
| BasicBlock-21 | [-1, 256, 8, 8] | 0 |
| Conv2d-22 | [-1, 512, 4, 4] | 1,179,648 |
| BatchNorm2d-23 | [-1, 512, 4, 4] | 1,024 |
| Conv2d-24 | [-1, 512, 4, 4] | 2,359,296 |
| BatchNorm2d-25 | [-1, 512, 4, 4] | 1,024 |
| Conv2d-26 | [-1, 512, 4, 4] | 131,072 |
| BatchNorm2d-27 | [-1, 512, 4, 4] | 1,024 |
| BasicBlock-28 | [-1, 512, 4, 4] | 0 |
| AdaptiveAvgPool2d-29 | [-1, 512, 1, 1] | 0 |
| Linear-30 | [-1, 10] | 5,130 |

```
==============================================================
Total params: 4,903,242
Trainable params: 4,903,242
Non-trainable params: 0
--------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 6.57
Params size (MB): 18.70
Estimated Total Size (MB): 25.28
--------------------------------------------------------------
```

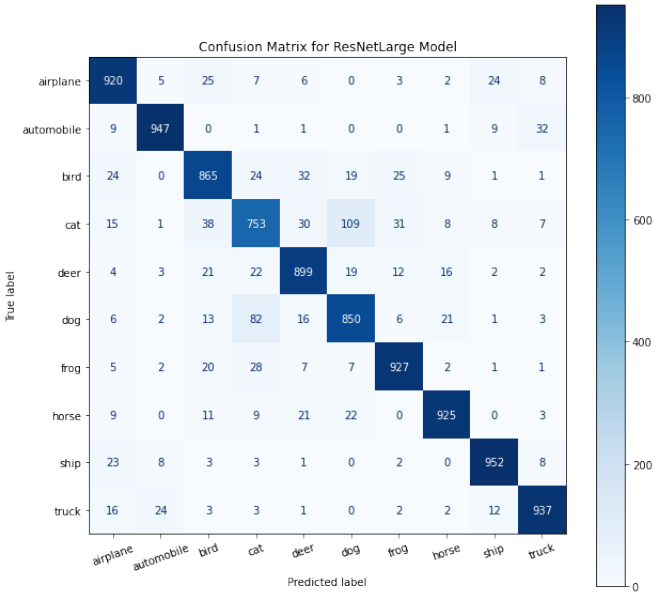


Figure 4: Confusion Matrix LargeModel

**Basic block** The BasicBlock class in PyTorch is a building block for the ResNet neural network architecture. It consists of two convolutional layers, batch normalization, and ReLU activation. The first convolutional layer applies a 3x3 convolution and batch normalization, followed by a second convolutional layer with shortcut connections. The output is passed through ReLU activation and returned as the output of the BasicBlock layer.

**Skip Connection in basic block** The skip connection in the BasicBlock is implemented through a shortcut layer as we can see in Figure 1, which directly connects the input to the output of the block through element-wise addition. This helps the network learn identity mappings, which can improve gradient propagation in deep networks. The shortcut connection is only applied when input and output have different dimensions, and it consists of Conv2d and BatchNorm2d layers to adjust dimensions and channels. If input and output have the same dimensions, the shortcut layer is an identity mapping with no additional computation.

**Optimizer** After experimenting with various optimizers for training our model, we concluded that using Stochastic Gradient Descent (SGD) with Momentum and Nesterov Accelerated Gradient (NAG) resulted in the highest accuracy. The optimizer configuration with the highest accuracy with $learningrate = 0.01, momentum = 0.9$ achieved an accuracy of above 94% on the large model. Other optimizers such as Adam achieved an accuracy of about 85%.

**Hyper-parameters** In the project, we utilized several hyperparameters to configure the learning rate (LR) during the

training process. The LR was set to a minimum value of 1e-6 and a maximum value of 1e-2, spanning a wide range to enable effective model optimization. We trained the model for a total of 30 epochs, which represents the number of complete passes through the entire training dataset. To dynamically adjust the LR during training, we employed a cyclic learning rate (CLR) strategy. The CLR was implemented using the CyclicLR scheduler from the PyTorch optimizer module.

**Criterion** In our project, we used the cross-entropy loss (denoted as nn.CrossEntropyLoss() in PyTorch) as the objective function, or criterion, for training our model. The cross-entropy loss is a common choice for multi-class classification tasks, such as image classification, where the goal is to classify input data into multiple mutually exclusive classes. It measures the dissimilarity between the predicted class probabilities and the true class labels and provides a scalar value that reflects the model's performance. During training, the model's parameters are updated based on the gradient of the cross-entropy loss, with the aim of minimizing the loss and improving the model's accuracy in predicting the correct class labels.

**Data Augmentation** In our project, we used data augmentation techniques to enhance the training dataset and improve the model's ability to generalize to unseen data. We used the torchvision.transforms module in PyTorch to apply various image transformations to the input images before feeding them into the model for training.
For the training dataset, we applied the following transformations using the transforms.Compose() function:
Randomly rotating the image by a maximum of 5 degrees in a counter-clockwise or clockwise direction, which introduces diversity in the orientation of the images in the training set.
Horizontally flipping the image with a probability of 0.5, introduces diversity in the horizontal orientation of the images in the training set.
Randomly cropping a 32x32 pixel patch from the image with a padding of 2 pixels, introduces diversity in the spatial location of the images in the training set.

## Evaluation

The three proposed models were evaluated using multiple metrics. Cross Entropy Loss was chosen for training purposes, and the models were compared based on their training loss and accuracy across epochs. As shown in Figure 5 and Figure 6, the large model outperformed the small model, exhibiting the lowest loss and highest accuracy throughout the epochs. To further visualize the results, confusion matrices were plotted in Figures 2, 3, and 4. It was observed that certain labels, such as 'airplane' and 'automobile', were classified more accurately by all three models, while the 'cat' label had the highest misclassification rate, often being misclassified as 'dog'. Overall, the larger model demonstrated better performance in terms of both accuracy and loss, with the other models showing similar performance in comparison.

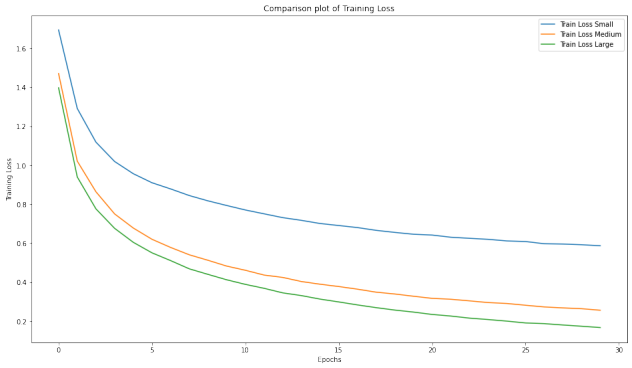


Figure 5: Training loss comparison of the three models

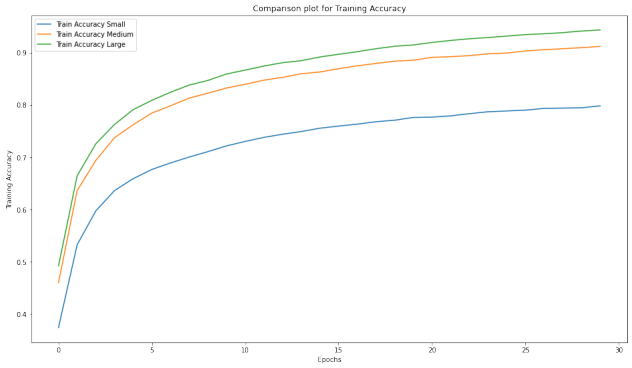


Figure 6: Training accuracy comparison of the three models

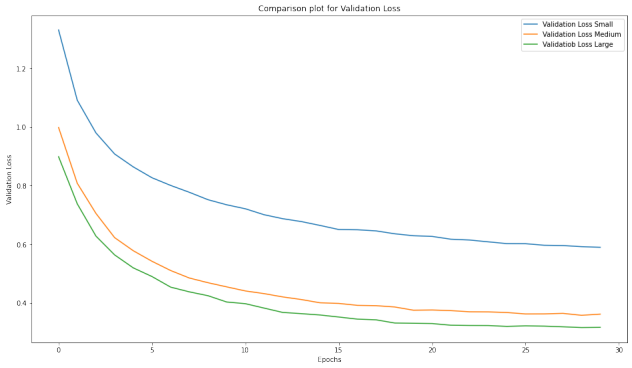


Figure 7: Test loss comparison of the three models

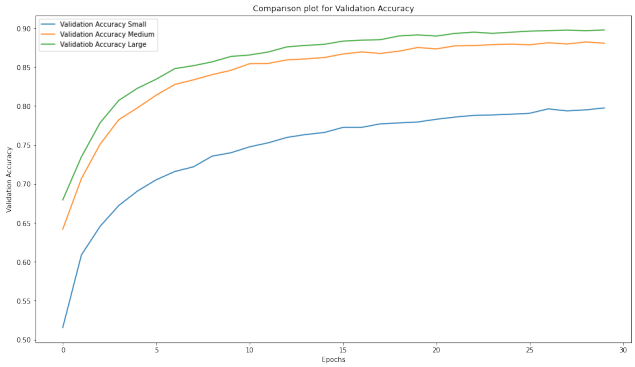


Figure 8: Test accuracy comparison of the three models

Table 1: Finding best $\lambda$ for different noise level

| Models | Parameters | Train Accuracy | Test Accuracy |
|---|---|---|---|
| ResNetSmall | $78,042$ | $24.97$ | $16.88$ |
| ResNetMedium | $1,228,970$ | $24.97$ | $16.88$ |
| ResNetLarge | $4,903,242$ | $28.68$ | $19.53$ |

## System Specification

NYU HPC VM
CPU: 8 Virtualized Cores of Intel Xeon-Platinum 8286
GPU: Nvidia Quadro RTX 8000
System Memory: 96 GB
Python Version: 3.8.6
CUDA version: v11.8
Torch Version: 2.0.0

## Acknowledgment

We would like to acknowledge the use of OpenAI's GPT-3.5 language model, commonly referred to as ChatGPT, for providing assistance with generating content for certain sections of this report.

## References

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. Technical report.

Liu, Y.; Gao, Y.; and Yin, W. 2020. An improved analysis of stochastic gradient descent with momentum. *Advances in Neural Information Processing Systems*, 33: 18261–18271.

.