# Using ResNet Model for Image Classification using CIFAR10 dataset

**Written by Ran Xu, Syed Ahzam Tariq, Safdar Ahmed**

New York University
rx523@nyu.edu, sat10045@nyu.edu, sa8237@nyu.edu
Github link

## Abstract

In this paper, we present modified versions of the ResNet model designed to achieve high accuracy within the constraint of fewer than 5 million trainable parameters. Through extensive experimentation involving various techniques such as learning rates, efficient optimizers, epoch, and dropout strategies, we have fine-tuned these models to optimize performance. Our results are meticulously analyzed and discussed, highlighting the potential implications for applications where model size and computational resource limitations are significant.

## Introduction

The CIFAR-10 dataset(Krizhevsky 2009), a staple in the machine learning community for benchmarking image recognition algorithms, consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. Given the complexity and variability of the dataset, achieving high accuracy requires a model that is both sufficiently complex to capture detailed features and efficient enough to operate under parameter constraints. We use ResNet in this project. ResNet, originally developed by He et al. (He et al. 2015), introduced a revolutionary architectural concept of skip connections that allow gradients to flow through the network directly, mitigating the vanishing gradient problem and enabling the training of very deep networks. Our approach modifies the standard ResNet architecture to tailor it for the CIFAR-10 dataset while optimizing the total number of parameters to stay within the 5 million limit. This involves strategic adjustments in layer configurations, filter sizes, and the incorporation of advanced regularization techniques to enhance learning efficiency and generalization.

## ResNet Model

ResNet introduces the ”residual block,” each of which contains a ”shortcut connection.” This connection allows gradients to pass directly through the block, thereby avoiding the issue of gradient vanishing. Consequently, deep networks can learn residuals through identity mapping, making the network easier to train as its depth increases. ResNet is characterized by its use of residual blocks, each consisting of two or three convolutional layers whose outputs are added to the input of the block via shortcut connections to form residuals. These shortcut connections allow gradients to flow directly through the block, helping to avoid the problem of gradient vanishing. Batch normalization is extensively used in ResNet to accelerate training and stabilize the network. By stacking multiple residual blocks, ResNet constructs deep networks, with the depth being freely adjustable based on the complexity of the task. The classic network architectures of ResNet include: ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152. Among these, ResNet-18 and ResNet-34 share the same basic structure and are considered relatively shallow networks, while the latter three have different structures compared to ResNet-18 and ResNet-34 and are deeper networks.

However, due to a limit of 5 million parameters as part of this project, we have restricted ourselves to smaller and less complex models. We have finalised 2 different custom models which produced high accuracy on unseen test sets - one of which we are referring to as ResNetLarge in our test and the other one as ResNet18. The authors refered to GithUB repository of Kuang Liu to get started on the project (Liu 2023).

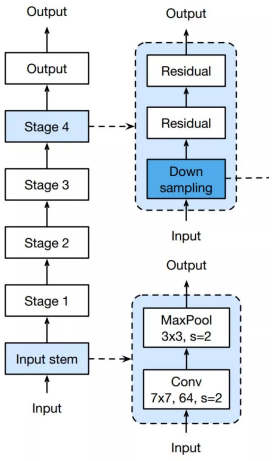


Figure 1: ResNet Model

## Model Specifications - ResNet18

| Layer (type) | Output Shape | Param # |
|---|---|---|
| Conv2d-1 | [-1, 42, 32, 32] | 1,134 |
| BatchNorm2d-2 | [-1, 42, 32, 32] | 84 |
| Conv2d-3 | [-1, 42, 32, 32] | 15,876 |
| BatchNorm2d-4 | [-1, 42, 32, 32] | 84 |
| Dropout-5 | [-1, 42, 32, 32] | 0 |
| Conv2d-6 | [-1, 42, 32, 32] | 15,876 |
| BatchNorm2d-7 | [-1, 42, 32, 32] | 84 |
| BasicBlock-8 | [-1, 42, 32, 32] | 0 |
| Conv2d-9 | [-1, 42, 32, 32] | 15,876 |
| BatchNorm2d-10 | [-1, 42, 32, 32] | 84 |
| Dropout-11 | [-1, 42, 32, 32] | 0 |
| Conv2d-12 | [-1, 42, 32, 32] | 15,876 |
| BatchNorm2d-13 | [-1, 42, 32, 32] | 84 |
| BasicBlock-14 | [-1, 42, 32, 32] | 0 |
| Conv2d-15 | [-1, 84, 16, 16] | 31,752 |
| BatchNorm2d-16 | [-1, 84, 16, 16] | 168 |
| Dropout-17 | [-1, 84, 16, 16] | 0 |
| Conv2d-18 | [-1, 84, 16, 16] | 63,504 |
| BatchNorm2d-19 | [-1, 84, 16, 16] | 168 |
| Conv2d-20 | [-1, 84, 16, 16] | 3,528 |
| BatchNorm2d-21 | [-1, 84, 16, 16] | 168 |
| BasicBlock-22 | [-1, 84, 16, 16] | 0 |
| Conv2d-23 | [-1, 84, 16, 16] | 63,504 |
| BatchNorm2d-24 | [-1, 84, 16, 16] | 168 |
| Dropout-25 | [-1, 84, 16, 16] | 0 |
| Conv2d-26 | [-1, 84, 16, 16] | 63,504 |
| BatchNorm2d-27 | [-1, 84, 16, 16] | 168 |
| BasicBlock-28 | [-1, 84, 16, 16] | 0 |
| Conv2d-29 | [-1, 168, 8, 8] | 127,008 |
| BatchNorm2d-30 | [-1, 168, 8, 8] | 336 |
| Dropout-31 | [-1, 168, 8, 8] | 0 |
| Conv2d-32 | [-1, 168, 8, 8] | 254,016 |
| BatchNorm2d-33 | [-1, 168, 8, 8] | 336 |
| Conv2d-34 | [-1, 168, 8, 8] | 14,112 |
| BatchNorm2d-35 | [-1, 168, 8, 8] | 336 |
| BasicBlock-36 | [-1, 168, 8, 8] | 0 |
| Conv2d-37 | [-1, 168, 8, 8] | 254,016 |
| BatchNorm2d-38 | [-1, 168, 8, 8] | 336 |
| Dropout-39 | [-1, 168, 8, 8] | 0 |
| Conv2d-40 | [-1, 168, 8, 8] | 254,016 |
| BatchNorm2d-41 | [-1, 168, 8, 8] | 336 |
| BasicBlock-42 | [-1, 168, 8, 8] | 0 |
| Conv2d-43 | [-1, 336, 4, 4] | 508,032 |
| BatchNorm2d-44 | [-1, 336, 4, 4] | 672 |
| Dropout-45 | [-1, 336, 4, 4] | 0 |
| Conv2d-46 | [-1, 336, 4, 4] | 1,016,064 |
| BatchNorm2d-47 | [-1, 336, 4, 4] | 672 |
| Conv2d-48 | [-1, 336, 4, 4] | 56,448 |
| BatchNorm2d-49 | [-1, 336, 4, 4] | 672 |
| BasicBlock-50 | [-1, 336, 4, 4] | 0 |
| Conv2d-51 | [-1, 336, 4, 4] | 1,016,064 |
| BatchNorm2d-52 | [-1, 336, 4, 4] | 672 |
| Dropout-53 | [-1, 336, 4, 4] | 0 |
| Conv2d-54 | [-1, 336, 4, 4] | 1,016,064 |
| BatchNorm2d-55 | [-1, 336, 4, 4] | 672 |
| BasicBlock-56 | [-1, 336, 4, 4] | 0 |
| Linear-57 | [-1, 10] | 3,370 |

Total params: 4,815,940
Trainable params: 4,815,940
Non-trainable params: 0

Input size (MB): 0.01
Forward/backward pass size (MB): 8.61
Params size (MB): 18.37
Estimated Total Size (MB): 27.00

Figure 2: ResNet18 Model

Our version of ResNet18 architecture contains 4 layers (42, 84, 168, 336 channels each) and a fully connected layer with 336 units, along with dropout layers added at a a rate of 0.15. We have added Global Average pooling before the final classification layer and used the AdamW optimizer in training.

## ResNetLarge

| Layer (type) | Output Shape | Param # |
|---|---|---|
| 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 3: ResNet Model

Our custom ResNetLarge architecture contains 4 layers (64, 128, 256, 512 channels each) and a fully connected layer with 512 units. The number of filters in the convolutional layers gradually increases from ? to ? as we go deeper into the network. We have added Global Average pooling before the final classification layer and used the SGD optimizer in training.

## Training Approach

As part of our pursuit to achieve more than 80% accuracy on the testing set, we first followed a manual hit and trial method in order to achieve an acceptable accuracy. However, this was not yielding any fruitful outcome and even by trying various methods we could only raise the accuracy by a few fractions of a percent. This led us to take up more powerful approaches first of which was implementing a powerful data augmentation strategy. At the same time, our team was trying several different combinations of various paramters. This lead us to achieve an above 80% accuracy on Kaggle. This was our ResNet18 model.

At the same time, we were working on a slightly different architecture as well. We implemented a Grid Search strategy so that we do not have to manually select hyperparameters and we can automate this process. This led us to build our ResNetLarge model which was able to get an accuracy of 81.3% on Kaggle.

## Data Augmentation

In our project, we've enhanced the data augmentation process for training deep learning models using the CIFAR10 dataset. We applied different transformations to the training and testing datasets to improve model robustness and generalization. For the training set, a series of augmentations were implemented: random rotations up to 10 degrees, horizontal flips with a 50% probability, random cropping with padding, color jittering for brightness, contrast, saturation, and hue adjustments, and random affine transformations for slight translations. These operations help the model learn from a more varied set of images, thereby reducing overfitting and improving its ability to generalize from the training data to unseen data.

The original training images also undergo standard transformations including tensor conversion and normalization using predefined mean and standard deviation values. We combined the transformed and original datasets using PyTorch's `ConcatDataset`, creating a richer, more diverse training dataset. This combined dataset is then loaded for training using a batch size of 64 and shuffling enabled to ensure random selection of images during model training.

For testing, we maintained a simple transformation sequence: conversion to tensor format and normalization, similar to the original images in the training set. This approach ensures that the model evaluations during testing are performed on data that reflect the real-world conditions the model expects to operate within, without the additional variability introduced in the training phase. The test dataset is loaded with the same batch size but without shuffling, ensuring consistent evaluation across different testing sessions. This comprehensive approach to data handling and augmentation significantly aids in developing a robust model capable of performing well on diverse image data.

## Leanings as part of the project

This was an important learning experience where we learned how to push our neural network training to its limit given limited training dataset. Our several iterations were not able to cross the 80% suggested accuracy. This led us to try various other methods and among the successful methods was our data augmentation strategy along with trying out various different optimiser. We started with Adam and switched to Stochastic Gradient Descent and AdamW for ResNetLarge and ResNet18 respectively. As part of the project, we realised that it is necessary to find the best combination of our parameters like: Momentum, Learning Rate, batch sizes, epochs in order to achieve a good accuracy. Therefore, we performed a Grid Search in our ResNetLarge model so that we can avoid manual trial and error.

| **Model ->** | **ResNet18** | **ResNetLarge** |
|---|---|---|
| Epochs | 80 | 120 |
| Optimiser | AdamW | SGD |
| Parameters | 4,815,940 | 4,903,242 |
| Final Test Accuracy | 93.02% | 92.73% |
| Final Train Accuracy | 97.98% | 98.22% |

Table 1: Model Comparison

We also noted that the size of the dataset is also very important if we want to train robust models. We therefore implemented a very powerful data augmentation strategy so that our model can handle very different test cases.

## Results

We have worked on 2 different models which are slightly similar as part of this mini Project. Both of the models have been described earlier and have been summarised in Table 1. The ResNetLarge model with 4.9 million parameters performs slightly better with an accuracy 0f 81.3% on Kaggle. Our ResNet18 model has an accuracy of 81.2 on Kaggle.

Figure 4 and 8 shows the confusion matrix of each of our algorithms for the test case. Also, training & testing loss and accuracy have been plotted for each of our architectures in Figure 6 and 9 to get an idea about how the training is taking place. It helps us visualised that after 40 epochs, very slight improvement in accuracy is taking place.

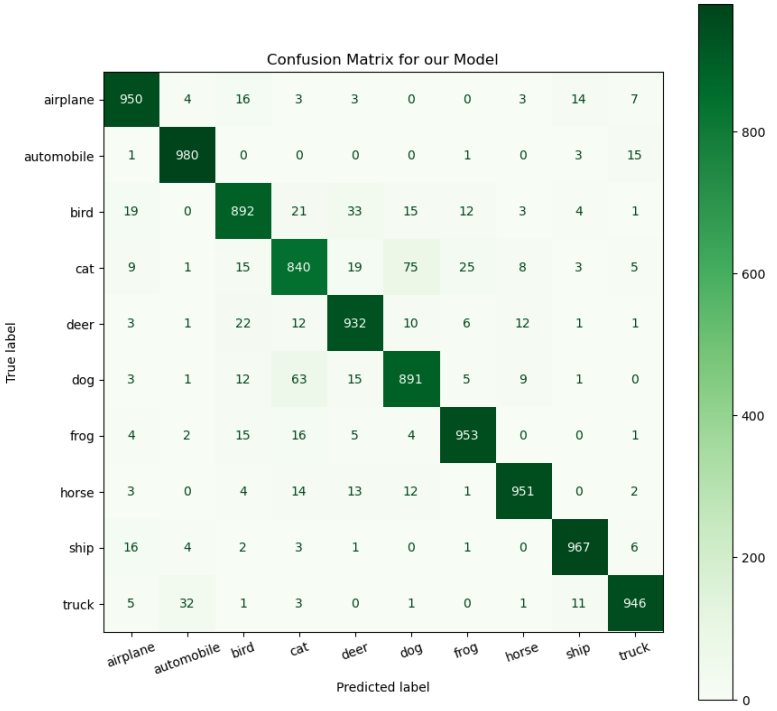


Figure 4: ResNet18 Confusion Matrix

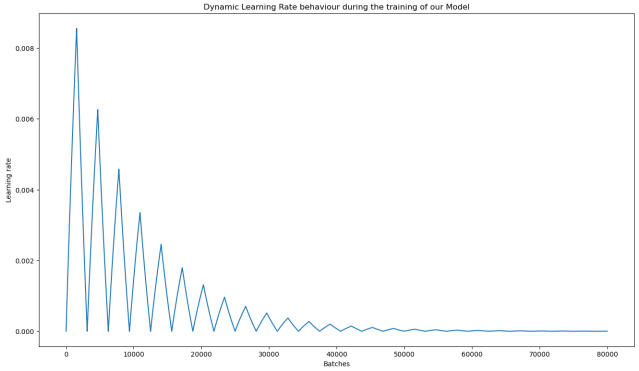


Figure 5: Dynamic Learning Rate behaviour during the training of ResNet18 Model

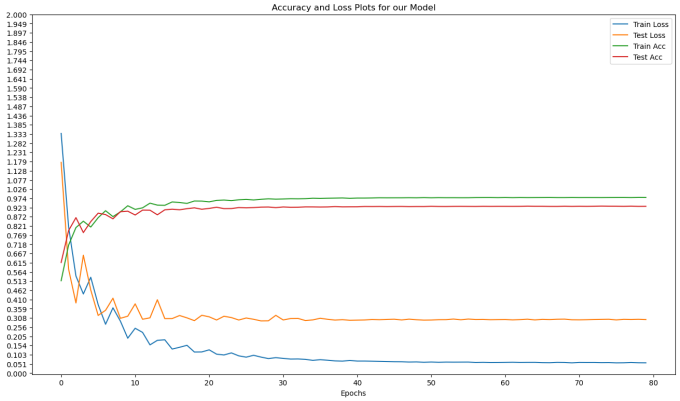


Figure 6: ResNet18 Accuracy & Loss Plots

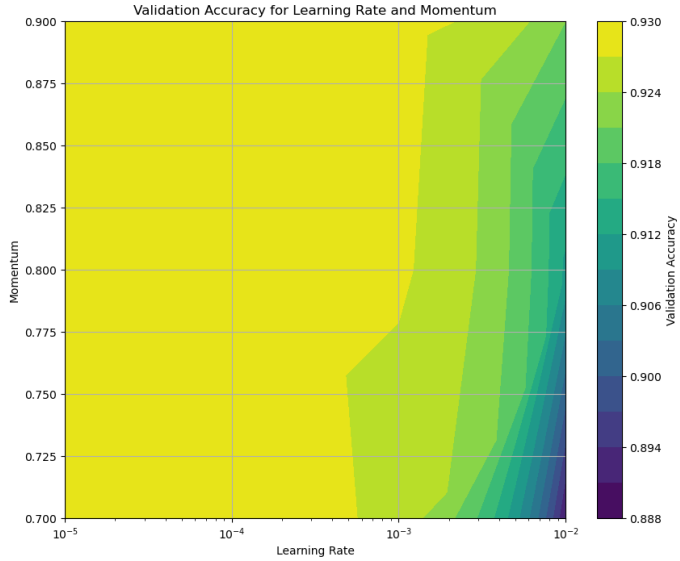


Figure 7: ResNetLarge Validation accuracy for Learning rate & Momentum

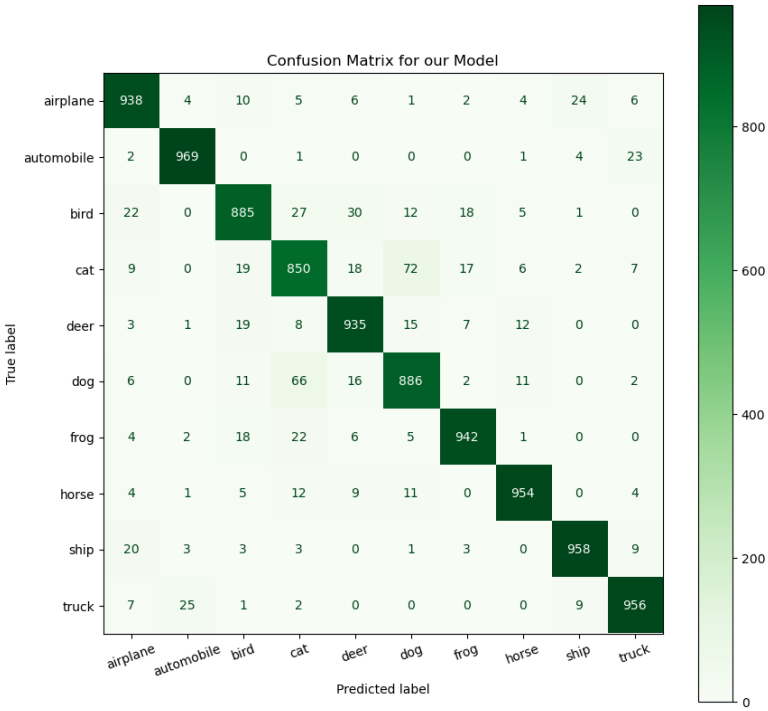


Figure 8: ResNetLarge Confusion Matrix

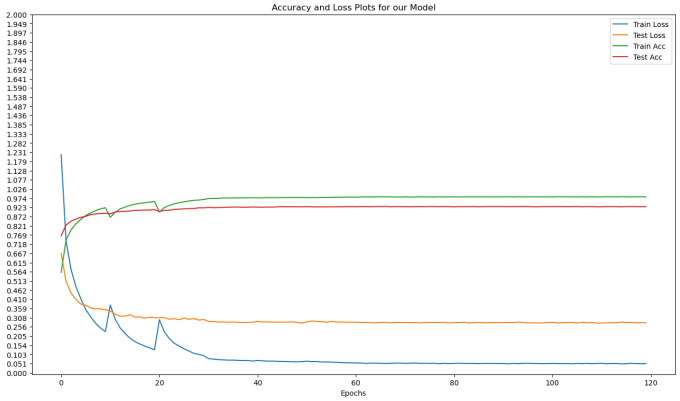


Figure 9: ResNetLarge Accuracy & Loss Plots

## References

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep Residual Learning for Image Recognition. *CoRR*, abs/1512.03385.

Krizhevsky, A. 2009. Learning Multiple Layers of Features from Tiny Images. Technical report. Chapter 3: Dataset and Methodology.

Liu, K. 2023. PyTorch CIFAR. https://github.com/kuangliu/pytorch-cifar. Accessed: 2023-04-12.