Mini-Project 1: Residual Network Design

Anant Singh
New York University
anant.singh@nyu.edu

Shantanu Kumar
New York University
sk9698@nyu.edu

Stuti Biyani
New York University
sb7580@nyu.edu

Abstract

Convolutional Neural Networks are expected to perform better when there are more number of layers. But when the network is too deep, it results in the vanishing gradient problem. This is because the gradient function shrinks to zero after the chain rule is applied repeatedly. Therefore, the weights never update values and no learning is performed. This problem is solved by using ResNets. In this paper, we design a ResNet model to classify images in the CIFAR-10 dataset. After tuning the hyper-parameters and applying data augmentation techniques, our model performs with an accuracy of 94.8%. The code can be found at https://github.com/95anantsingh/NYU-ResNet-On-Steroids.

1 Introduction

To solve a complex problem, we add layers to a Deep Neural Network which results in improved performance and accuracy. But there is a maximum threshold for depth with traditional CNNs. After the threshold is reached, the model stops learning and the performance starts degrading. This problem has been solved with Residual Networks or ResNets.

ResNets consist of N Residual layers which contain one or more residual blocks. Residual blocks contain two convolutional layers with a skip connection. In figure 1, the connection that skips some layers of the model is the skip connection.

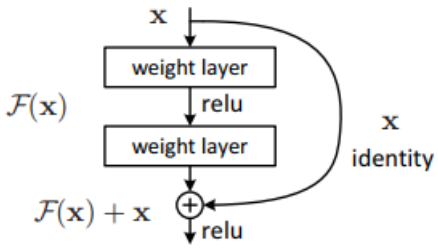


Figure 1
Residual Block
Source: Deep Residual Learning for Image Recognition [1]

The output of the model after using skip connection is:

$$H(x) = f(x) + x$$

Unlike the layers in a traditional network, instead of learning the true output *H(x)*, the layers in a residual block are learning the residual *f(x)*.

Each residual layer is preceded by a fixed convolutional block, and followed by an average pooling layer, and a fully connected layer. Average Pooling is basically a pooling operation that calculates the average value for patches of a feature map, and uses it to create a down-sampled feature map [6].

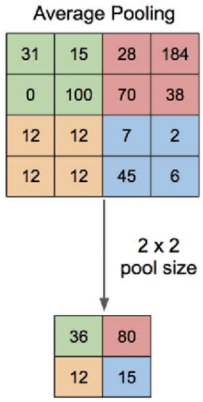


Figure 2
Average Pooling
Source: Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail [6]

The accuracy of any model depends on the careful selection of hyper-parameters. Our goal is to choose the hyper-parameters such that the model has the best accuracy, and less than $5M$ trainable parameters. The hyper-parameters that we vary to get the most accurate Resnet model:

N := Number of layers
B_i := Number of residual blocks in Residual layer *i*
C_i := Number of channels in Residual layer *i*
F_i := Convolutional kernel size in Residual layer *i*
K_i := Skip connection kernel size in Residual layer *i*
P := Average pool kernel size

In our work, we train different ResNet models on the CIFAR-10 dataset to classify images. We get the best accuracy for **ResNet-24** which has $\mathbf{4,935,242}$ number of parameters. The learning rate we used is $\mathbf{0.1}$, loss function is **Cross Entropy Loss**, and the optimizer used is **adadelta**. We performed data augmentation using **mixup**. We also use learning rate schedulers and cosine annealing. Our final ResNet model has the following hyper-parameters:

$$N = 4 \qquad B = 3, 3, 2, 3$$
$$C = 64, 128, 128, 256 \qquad F = 3, 3, 3, 3$$
$$K = 1, 1, 1, 1 \qquad P = 4$$

We found that this configuration of hyper-parameters gives us the best results. This model could achieve a test accuracy of 94.8% on the CIFAR-10 dataset. The techniques we used to get the best configuration of hyper-parameters is discussed in detail in section 2. The final architecture of the model and the results are discussed in section 3.

2 Methodology

To develop and train our models, we started by loading the CIFAR-10 dataset and split it into train, test, and validation sets. We applied different transforms and data augmentation techniques on these sets. Our code is modular and can be re-used to generate any ResNet model. We have also used model checkpoints. So whenever training any model, we save the best state of the model and resume training from that state later. We will now discuss in detail what techniques and methods we used to get the most efficient ResNet.

2.1 ResNet Hyper-parameters

Number of Layers *N* and Residual Blocks *B*: We started by building a basic ResNet model with 3 layers with 1, 2, and 1 residual blocks in each layer respectively. We then tested this model for different number of residual blocks and recorded their accuracy. We found that for any configuration, **4** layers in the network give the best results. After finalizing on the number of layers, we changed the number of residual blocks in each layer to get the most efficient configuration. We monitored the test error and accuracy and increased the number of residual blocks in the network accordingly.

Kernel Size *(F, K, P)*: As we do not consider bottleneck blocks, the number of feature planes is kept the same across all the blocks. So the convolutional kernel size **F** we decided upon is 3, the skip kernel size **K** is 1, and the average pool kernel size is **P**. These values are adapted from Zaguruyko et al.[2]

Table 1 summarizes different values of the hyper-parameters we tuned to build our model.

Model	# of Params	N	B	C	F	K	P
ResNet-10-1	312,490	3	1,2,1	64,128,256	3,3,3	1,1,1	4
ResNet-10-2	4,903,242	4	1,1,1,1	64,128,256,512	3,3,3,3	1,1,1,1	4
ResNet-12	4,977,226	4	2,1,1,1	64,128,256,512	3,3,3,3	1,1,1,1	4
ResNet-16	4,909,642	4	2,1,2,2	64,128,256,256	3,3,3,3	1,1,1,1	4
ResNet-20	4,335,434	3	3,3,3	64,128,256	3,3,3	1,1,1	4
ResNet-24	4,935,242	4	3,3,2,3	64,128,128,256	3,3,3,3	1,1,1,1	4
ResNet-32	4,754,218	4	4,4,4,3	32,64,128,256	3,3,3,3	1,1,1,1	4
ResNet-48	4,994,378	4	8,5,5,5	64,128,128,128	3,3,3,3	1,1,1,1	4
ResNet-86	4,994,378	4	10,8,10,14	16,32,64,128	3,3,3,3	1,1,1,1	4

Table 1
ResNet Hyper-parameters

2.2 Other Hyper-parameters

Data Loading Workers: We started with finding the optimal number of data workers to load the CIFAR-10 dataset. We recorded the time taken by different number of workers to load data for five epochs. The results are as shown in Table 2. Based on these results, we decided that the optimal number of data loading workers is 2 for any ResNet configuration as it takes the least amount of time to load the data.

# of Workers	Time(sec)
0	63.96
2	2.16
4	2.80
8	4.18
12	5.61
16	7.00
20	8.24

Table 2
Time taken to load data for different # of workers

Data Augmentation: Even after tuning the parameters for different number of layers, our model did not show a significant increase in the validation accuracy. This is because, as we train on only a certain part of data, the model is likely to over fit on the 'seen' part of the distribution. So the more data we have for training, the model will cover a better distribution of the data. We perform data augmentation - the process of increasing the number of data points by altering the existing data. We then feed these 'new' altered samples into the network for training.

We also performed data augmentation using **transformers**. These transforms are applied to every batch, hence keeping the original dataset unchanged. The different transformations we applied to our dataset are: Random Cropping - we add sufficient padding to the original image and then randomly crop to create a new image, Random Flipping - the original image is randomly flipped to produce new image, and Normalization.

(a) Data *before* transformation

(b) Data *after* transformation

Figure 3
Data Augmentation

Even after applying transformers, we did not observe significant increase in accuracy. So we additionaly applied another data augmentation technique: **Mixup**. Mixup simply averages out two images and their labels correspondingly as new data. Zhang et al. [3] had experimented with multiple datasets and architectures, including CIFAR-10 and observed a significant increase in performance.

When we tested our model before using mixup, our network had resulted in approximately 91.09% accuracy on the test set, while **with mixup**, the accuracy was boosted to around 93.98% with the same configuration!

Optimizers and Learning Rate: We tested our models on five different optimizers - adadelta, adagrad, adam, sgd, and sgdn. The accuracies that we achieved are as shown in 5a. These accuracies are the best accuracies for a given optimizer with varying learning rates.

We start with a learning rate that decays as the number of epochs increases. After every 100 epochs, the learning rate decays by 0.1. This is called **Learning Rate Scheduler** [4]. Learning Rate Scheduler helps converge faster with higher accuracy. There are two types of Learning Rate Schedules - Step-wise Decay, and Reduce on Loss Plateau Decay. We use Step-Wise Decay for our model. We have also used **Cosine Annealing** technique.

Cosine Annealing a type of learning rate schedule that has the effect of starting with a large learning rate that is relatively rapidly decreased to a minimum value before being increased rapidly again [5]. The resetting of the learning rate acts like a simulated restart of the learning process and the re-use of good weights as the starting point of the restart is referred to as a "warm restart" in contrast to a "cold restart" where a new set of small random numbers may be used as a starting point. The following equation refers as to how the learning rate is updated at every epoch T_{cur}.

$$\eta_t = \eta^i_{min} + \tfrac{1}{2}(\eta^i_{max} - \eta^i_{min})(1 + cos(\frac{T_{cur}}{T_i}\pi))$$

Figure 5b shows the accuracies of different ResNet models when the learning rate is varied.

For all our networks, the **momentum** is set to 0.9, **weight decay** is 0.0005, **train batch size** is 128, and **test batch size** is 100. We have logged all our results at this link to Google Sheet.

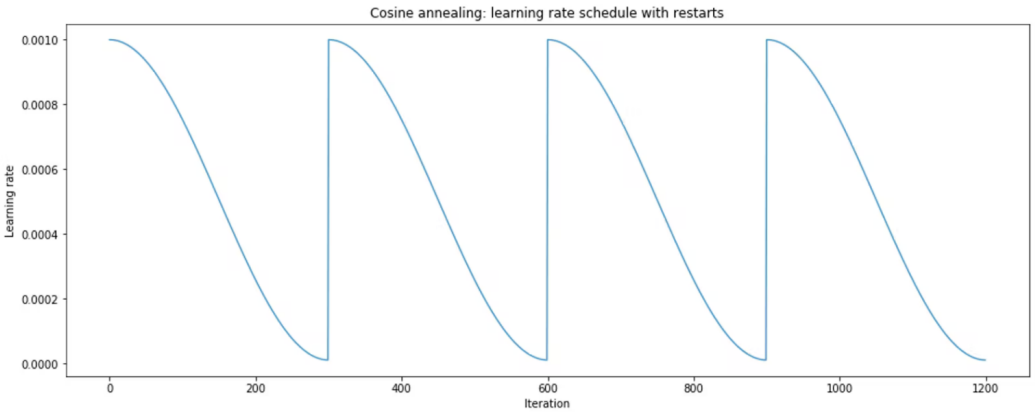
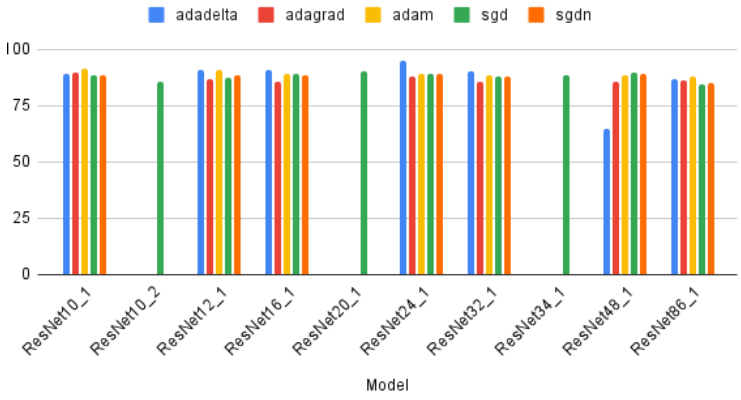
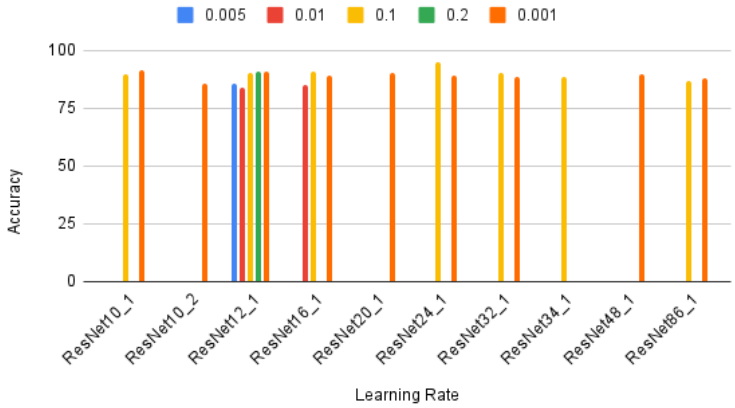


Figure 4
Cosine Annealing
Source: A brief history of learning rate schedulers and adaptive optimizers [7]



(a) Optimizers vs Accuracy for different ResNet Models



(b) Learning Rate vs Accuracy for our best performing model ResNet-24

Figure 5
Optimizers and Learning Rate vs Accuracy

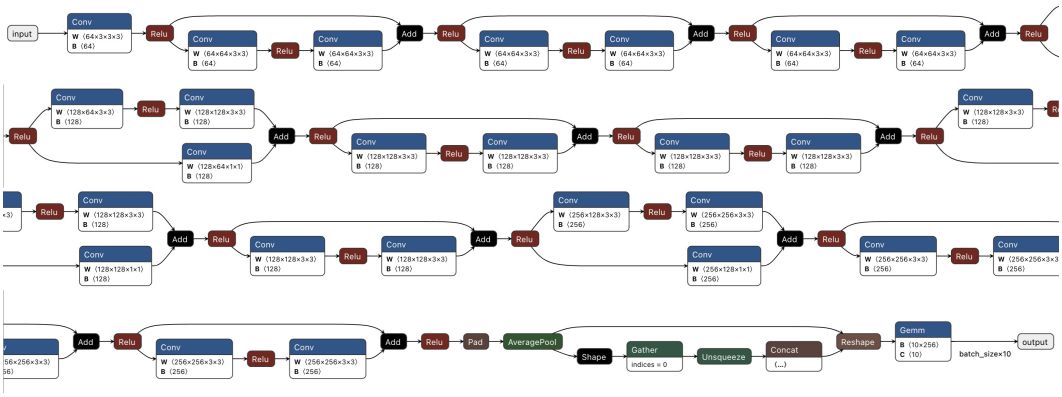


Figure 6
ResNet-24 Architecture

3 Results

After testing our model for different hyperparameters, the best accuracy we achieved is of 94.8%. Figure 6 is the visualization of our ResNet-24 model. The summary of this model is as shown in 7

Total Trainable params :	$4,935,242$
Train Batch Size :	128
Test Batch Size :	256
Validation Batch Size :	256
Optimizer :	adadelta
Learning Rate :	0.1
Momentum :	0.9
Weight Decay :	0.0005
N :	4
B :	$3, 3, 2, 3$
C :	$64, 128, 128, 256$
F :	$3, 3, 3, 3$
K :	$1, 1, 1, 1$
P :	4

Figure 7
Final ResNet Architecture

When we tested CIFAR-10 dataset on different ResNet architectures, this particular architecture gave us the best results. Training the model for just 50 epochs, without using learning rate scheduling, and without applying mixup, we could achieve an accuracy of 91.09% which was better than all the other model performances. Then we decided to choose ResNet-24 as our final model and applied techniques to boost its accuracy.

We first enabled learning rate scheduler and trained the model for 200 epochs. Then we applied Mixup data augmentation strategy which boosted the models test accuracy to 93.2%. We saved the best model state as our checkpoint and then trained it again for 200 epochs more. We then achieved an accuracy of 94.8%. After this, we could not observe any significant increase in the performance of the model.

The final accuracy we achieved is $\mathbf{94.8}\%$ and the final test error achieved is $\mathbf{0.359}$.

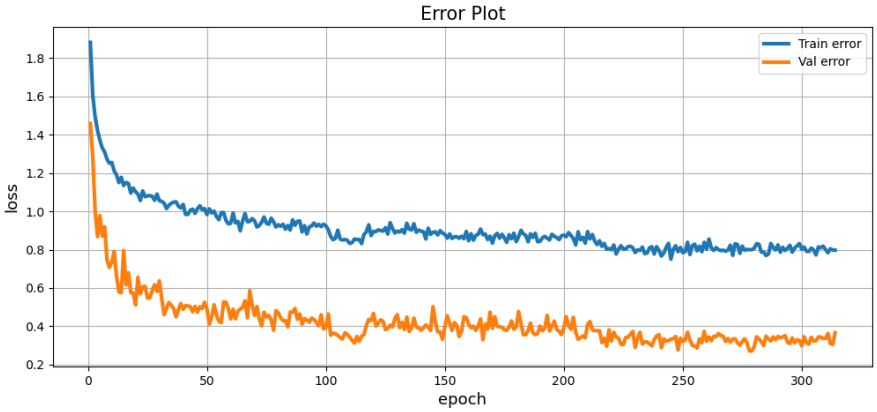


Figure 8
Test and Validation Error

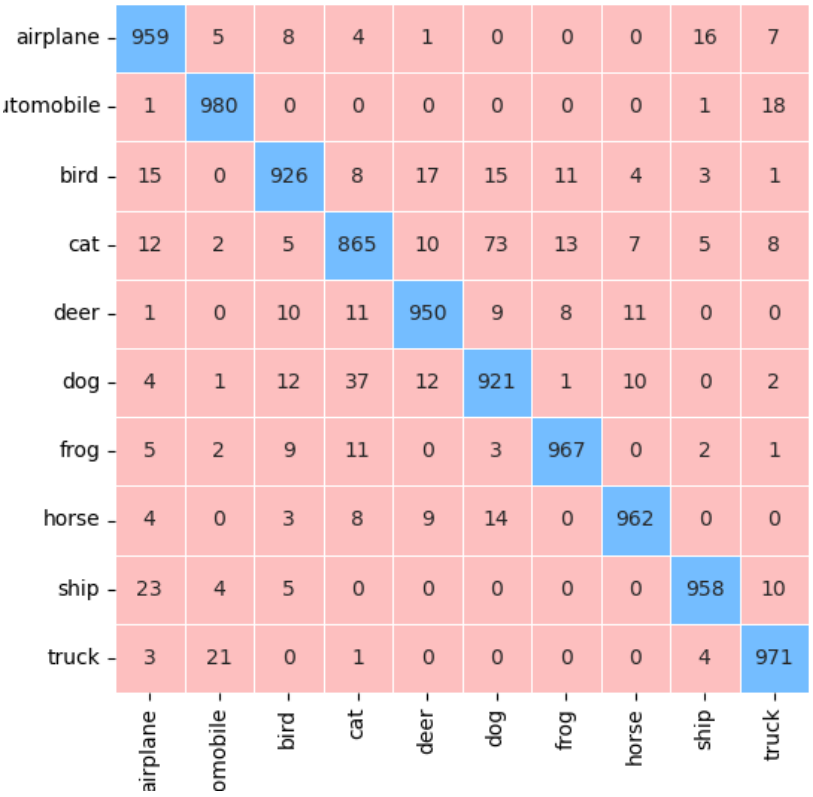


Figure 9
Confusion Matrix

9 shows the confusion matrix which summarizes the prediction results on all the different classes of the CIFAR-10 dataset.

4 Conclusion

In this paper we experiment with different hyper-parameters for ResNet architecture and apply data augmentation techniques to classify images in the CIFAR-10 dataset. We achieved an accuracy of 94.8% after training the dataset for over 300 epochs using adadelta optimizer and learning rate scheduler starting at 0.1. Our model has 4 convolutional layers with $3, 3, 2, 3$ residual blocks in each layer respectively. As restricted our experiment to just five optimizers, we believe that using more advanced optimizers and data augmentation techniques, we can achieve an even better accuracy.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. Deep Residual Learning for Image Recognition

[2] Sergey Zagoruyko, & Nikos Komodakis. Wide Residual Networks

[3] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, & David Lopez-Paz. mixup: Beyond Empirical Risk Minimization

[4] Chiheon Kim, Saehoon Kim, Jongmin Kim, Donghoon Lee, & Sungwoong Kim. Automated Learning Rate Scheduler for Large-batch Training

[5] Ilya Loshchilov, & Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts

[6] Yani, Muhamad & Irawan, S, & Setianingsih, Casi. Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail

[7] Aleksey Bilogur. A brief history of learning rate schedulers and adaptive optimizers