Mini-Project 1: Residual Network Design

Kartikay Kaushik
Dept. of ECE
NYU Tandon
kk4332@nyu.edu

Navya S. Jammalamadaka
Dept. of ECE
NYU Tandon
nsj9072@nyu.edu

Aadrit Ghimire
Dept. of ECE
NYU Tandon
ag8520@nyu.edu

Abstract

As part of the project, we have designed, developed and trained a ResNet architecture on the CIFAR-10 dataset. The objective of the model is to maximize the classification accuracy with a constraint on the trainable parameters. We observed the impact of various hyperparameters on the accuracy and total parameters of the model. We have also experimented by increasing the diversity of our training dataset using data augmentation and normalization techniques. The implemented architecture is computationally and memory-wise efficient.

1 Introduction

Deep convolutional neural networks have proven to be highly efficient in image classification [1]. The classifiers based on deep networks work in a multi-layered manner where the number of features can be accommodated by the depth of the network[2]. Though sufficient evidence proves the importance of the depth of network[3,4], it has also been observed that the increase in depth of network saturates the overall accuracy and then causes degradation[1,5]. This problem is addressed by deep residual learning frameworks like Residual Networks (ResNets). Most ResNet models are implemented using skip connections through convolutional layers, with batch normalization and non-linearities like ReLU or sigmoid activation in between. The skip connections efficiently address the vanishing gradients that have a high impact on convergence.[6][7]

In this project, we have designed a network model based on the ResNet architecture. The model was used for classification of CIFAR-10 image dataset under constraints of 5 million trainable parameters. We observed a maximum accuracy of 91.62% in approximately 2.9 million parameters. The model was trained for 60 epochs. The model was developed using PyTorch (version 1.10.0+cu113) and trained and tested on the NYU HPC platform. The project report aims to discuss the architecture search process, the procedure for selecting data augmentation methods and the impact of changes of various hyperparameters on the accuracy of the model. The constraints on the model size is a reflection of a problem statement where a light-weight image classification software is needed to be run on a device with storage constraints like mobile, IoT and edge computing devices.

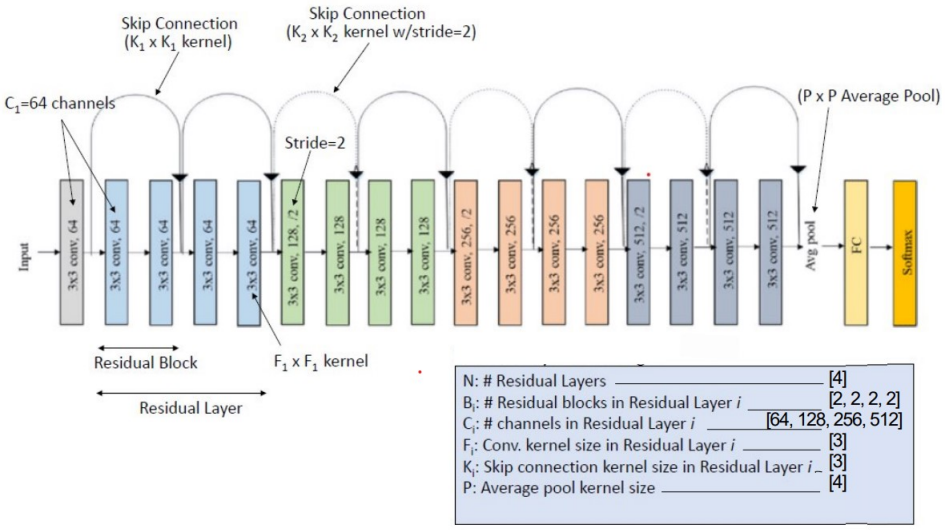
2 Methodology

We started the project by understanding the problem statement, the constraints and the available resources. After carrying out the necessary installations on NYU HPC, we then comprehended our dataset (Section 2.1) and the provided skeleton ResNet architecture (Section 2.2) and its components. Then, we proceeded to develop our own architecture on top of the base ResNet by analyzing the impact of each parameter that can be tweaked and other techniques to pre-process data (Section 2.3 to Section 2.9).

2.1 Understanding the dataset

We have used the CIFAR-10 dataset to train and test our model. The CIFAR-10 dataset contains 60000 coloured images of size 32x32. Each image is labeled and belongs to either of the 10 classes. Each class has 6000 images. We split the dataset into training and testing data in a 5:1 ratio (50000 training images and 10000 test images). The training data was loaded with the shuffle parameter set to true while the testing data was loaded with the shuffle parameter set to false. We experimented with different batch sizes of 64, 128, 256 and 400. We observed that the overall computational time taken to run the complete code with training and testing was higher for larger batch sizes.

2.2 Understanding the provided skeleton ResNet



The skeleton ResNet architecture first consists of a convolutional layer of kernel size 3, padding 1, and stride 1. This layer maps the 32 X 32 X 3 (RGB)channel input into a feature map of the same image dimensions and 64 channels. The output of the convolution is batch normalized and passed through a ReLU nonlinear activation function. This is followed by 4 successive residual layers, each consisting of 2 residual blocks. Excluding the first residual layer, the first convolution across each residual block results in halving the dimensions of the image (32, 16, 8, 4), and doubling the channel width (64, 128, 256, 512). The output of the final residual layer is average-pooled across a dimensionality of 4, resulting in a feature set of 512 channels with (1 X 1) dimension. This feature set is now passed through a fully connected layer and SoftMax function to classify the input image into the 10 distinct CIFAR-10 dataset classes.

2.2.1 Calculating the number of parameters

For each convolutional layer within the architecture, including the skip connections, the number of parameters is given by the following formula:

$$F \times F \times C_{in} \times C_{out}$$

where, $F \times F$ = Kernel Dimensions, C_{in} = Input Channels, C_{out} = Output Channels,

The number of parameters for the fully connected layer is given by the following formula:

$$C_{in} \times K$$

where, C_{in} = Number of input features to FC layer, K = Number of classes

Based on the above formulae, the total number of parameters of the skeleton ResNet model defined in the previous section is approximately 11.2 million, which is significantly higher than our target maximum of 5 million parameters.

Throughout the project, we have used this method to calculate the number of parameters of the model.

2.3 Developing the model

In order to optimize the convergence and maximize the accuracy of our model, the following methodology was used:[8][9]

2.3.1 Loss function

We have used Cross-Entropy Loss as the loss function to calculate the performance of our classification model. We expect our model to provide an output in the form of predicted probability. The predicted probability is then compared to the actual label to determine the penalty to be imposed. Our objective is to minimize the value of the loss function.

2.3.2 Selection of optimizer and learning rate

As mentioned in Section 2.4, in order to improve the performance of our model, we have to minimize the value of the loss function. Optimizers help in modifying the weights, learning rate or training parameters of the network to reduce this penalty in each epoch of training. We experimented with the following set of optimizers on the provided skeleton network to determine which optimizer would give us the best accuracy - **Stochastic Gradient Descent, AdamW, AdaGrad, AdamW, AdaDelta**

We observed that the AdamW optimizer gave us the best results. The observations were also affirmed by [10]. The optimizers were tested with different combinations of learning rate - **0.01 and 0.001**

For N=4, B = [2,2,2,2], C = [64, 128, 256, 512], F = 3, K = 1, P = 4

Optimizer	Learning Rate	Accuracy	Parameters	Iterations
Adagrad	0.001	77.91%	11,164,352	49/50
SGD	0.001	61.24%	11,164,352	49/50
AdamW	0.001	85.01%	11,164,352	48/50
Adam	0.001	83.27%	11,164,352	86/50
Adam	0.01	67.87%	11,164,352	49/50
Adadelta	0.01	84.56%	11,164,352	47/50
AdamW	0.01	82.99%	11,164,352	31/50

2.3.3 Addition of weight decay

Weight decay adds an additional regularization term to our loss function, to control overfitting of the model to the training dataset. The value of weight decay was chosen experimentally to ensure perfect fit; The following values of weight decay were tested for different configurations - **0.0001, 0.001, 0.01, 0.1**

We observed that increasing values of weight decay provide increased regularization. However, beyond a certain value, the model is prone to underfiting. The optimal results were obtained with a value of 0.0001.

For Optimizer = AdamW, L.R. = 0.001, N=2, B = [4,4], C = [64, 128], F = 5, K = 1, P = 4

Weight Decay	Accuracy	Parameters	Iterations
0.0001	91.05%	3,924,672	37/50
0.001	90.32%	3,924,672	47/50
0.01	89.67%	3,924,672	48/50
0.1	88.32%	3,924,672	46/50

2.4 Selection of model configuration

After choosing an optimizer and a learning rate, we then proceeded to improve the accuracy of the model while reducing the number of training parameters. The following hyperparameters were tweaked to observe the impact of changes on accuracy and parameter count -

2.4.1 Number of residual layers in the convolution network - *N*

The following values of N were tested - **4, 3, 2**

N	*B*	*C*	Accuracy	Parameters	Iterations
2	[4,4]	[64, 128]	89.76%	1,431,232	42/50
3	[4,4,4]	[64, 128, 256]	83.56%	5,908,160	47/50
4	[2,4,4,4]	[64, 128, 256, 512]	83.21%	5,889,376	44/50

We observed that upon reducing the number of layers and keeping other hyperparameters intact, the number of trainable parameters significantly decreased. But, the accuracy of the model was severely affected. Hence, we decided to increase the depth of each residual layer by changing the residual blocks in it. This approach was backed by the findings in [3].

2.4.2 Number of residual blocks within each of the N residual layers - B_i

The following values of combinations of B_i were tested for for Optimizer = AdamW, Learning rate = 0.001, F = 3, K = 1, P = 4

N	*B*	*C*	Accuracy	Parameters	Iterations
2	[2,2]	[64, 128]	85.42%	693,952	46/50
2	[4,4]	[64, 128]	89.76%	1,431,232	42/50
2	[6,6]	[64, 128]	90.85%	2,168,512	41/50
2	[8,8]	[64, 128]	91.62%	2,905,792	52/60
2	[10,10]	[64, 128]	90.31%	3,643,072	49/50
2	[12,12]	[64, 128]	90.61%	4,380,352	45/50
3	[4,4,4]	[64, 128, 256]	83.56%	5,908,160	47/50
4	[2,2,2,2]	[64, 128, 256, 512]	81.58%	11,164,352	5/20
4	[3,3,3,3]	[32, 64, 128, 256]	83.78%	4,359,520	40/50
4	[2,4,4,4]	[64, 128, 256, 512]	83.21%	5,889,376	44/50

In the last case, we observed that the parameter count exceeded the limitation. As we decreased the value of *N*, we observed the decrease in parameter count as mentioned above. In the case of *N*=2, we observed that the accuracy of the model improved as we increased the value of B_i , but the value peaked at B_i=8. Nevertheless, we experimented with the other combinations too for different channel widths and kernel sizes, but with a greater confidence in *N*=2 and B_i=8 combinations.

2.4.3 The channel width of the first layer of the network - C_i

The following values of C_i were tested on top of the changes made in 2.6.1 - 2.6.3 - **16, 32, 64**

For *N*=4, *B* = [2,2,2,2], F = 3, K = 1, P = 4

C	Accuracy	Parameters	Iterations
[64, 128, 256, 512]	83.87%	11,164,352	40/50
[32, 64, 128, 256]	82.42%	2,792,800	24/50
[16, 32, 64, 128]	81.35%	699,056	37/50

We observed that the values of accuracies were slightly better with the C_i = 64. But in order to increase the channel width, we had to ensure that the values of N and Bi were such that the parameter count remained in check. This approach was backed by the findings in [11]

2.4.4 The filter/kernel size applied across each residual block across the convolutional network - F_i

The following values of F_i were tested for different configurations - **3, 5, 7**

For learning rate = 0.001, *N*=2, *P* = 16, *C* = [64, 128, 256, 512]

Optimizer	*B*	*F*	*K*	Accuracy	Parameters	Iterations
Adam	[2,2]	3	1	86.06%	674,752	16/50
AdamW	[4,4]	5	1	87.85%	3,971,008	49/50
AdamW	[2,2]	7	3	84.94%	3,631,552	48/50

We observed that the values of accuracies with respect to kernel size across each residual block varied depending on the depth of each residual layer (number of residual blocks). When the number of residual blocks per layer was low, a higher kernel size (F_i =5) gave us better accuracies and vice versa for (F_i =3).

In order to keep the dimensions of the output consistent, we modified the padding accordingly. For Fi =3, 5 and 7 the padding was 1,2 and 3 respectively.

2.4.5 The Kernel size applied across the skip connections of the residual blocks - K_i

The following values of K_i were tested for different configurations - **1, 3** For Optimizer = AdamW, learning rate = 0.001, *N*=2, *P* = 16, *C* = [64, 128, 256, 512]

B	*F*	*K*	Accuracy	Parameters	Iterations
[4,4]	5	1	87.85%	3,971,008	49/50
[2,2]	7	3	84.94%	3,631,552	48/50

We observed that the values of accuracies with respect to kernel size across each skip connections were slightly better with a lower kernel size on skip connection. Hence, we proceeded to test most of our configurations with a K_i=1.

2.4.6 The average pooling applied between the convolutional layer and the fully connected layer - *P*

In order to decrease the computational costs, we tried to reduce our dimensions. We achieved this by using an average pooling layer. In average pooling, we iterated through our image and calculated averages for a fixed size of image. This was performed right before our fully connected layer.

The following values of P were tested for different configurations - **2, 4, 16**

For Optimizer = Adam, learning rate = 0.001, *N*=2, *K*=1, *C* = [64, 128, 256, 512]

B	*F*	*P*	Accuracy	Parameters	Iterations
[2,2]	3	2	76.61%	755,392	37/50
[4,4]	5	4	87.32%	3,924,672	41/50
[2,2]	3	16	86.06%	674,752	16/50

Upon obtaining the corresponding accuracies, we observed that the average pooling size of 2 had least accuracy for different combinations of above discussed hyperparameters. When our image size after the convolutional layers were large, we observed better outputs with an *P*=16 and vice versa with *P*=4

2.5 Addition of dropouts

Dropout is a regularization technique that involves random disconnection of nodes in each iteration during training of the mode with a predetermined probability factor. As a result, the individual nodes in the network don't overfit to the training dataset.

Our model was trained with various dropout probabilities of **0.3, 0.5 and 0.7**. Dropout probabilities of 0.3 and 0.5 resulted in the best performance of the model, with insignificant difference in between the two. For greater regularization, we proceeded forward with 0.5.

Dropout was also tested on the fully connected layer. However, this resulted in a significant dip in the performance and was consequently abandoned. This dip can be attributed to the fewer number of nodes in the shallow fully connected layer[12][13].

For N=4, B = [2,2,2,2], C = [64, 128, 256, 512], F = 3, K = 1, P = 4, Normalized with mean [0,0,0] and Standard deviation [1,1,1]

Optimizer	LR	Accuracy	Parameters	Iterations	Dropout
Adam	0.001	84.56%	11,164,352	86/100	0
Adam	0.001	85.37%	11,164,352	46/50	0.3
Adam	0.01	85.07%	11,164,352	47/50	0.5
Adam	0.01	79.34%	11,164,352	45/50	0.7

2.6 Pre-processing data

Data Normalization is a pre-processing technique that standardizes the data around a predetermined mean and standard deviation (usually 0 and 1 respectively). This strategy optimizes the convergence of the learning rate by scaling down the distribution of the inputs. Both training and testing data were normalized before feeding into the skeleton network, comparing between 2 different mean and standard deviations of **[0,0,0], [1,1,1]** and of CIFAR-10 data **[0.491, 0.482, 0.446], [0.247, 0.243, 0.261]**.[14][15]

In addition, Data Augmentation was used to increase the training dataset. This was done by taking a copy of the CIFAR-10 training dataset, and transforming it with the following techniques: *Random Horizontal flip* with probability **0.5**, *Padding* with **4** and *randomly cropping* a **(32 x 32)** portion of the image. The resultant dataset was then concatenated with the original training dataset, and used to train the network, essentially doubling out training dataset from 50,000 to 100,000 [16][17][18].

Significant improvement in accuracy (Approx 3.5%) was observed by applying Normalization and Data Augmentation to the input dataset. *Gaussian blur* was also tested upon in the augmentation phase to introduce noise to the dataset, but it resulted in a negative impact on the test accuracy.

For Optimizer = Adam, Learning rate = 0.001, N = 4, B = [2,2,2,2], F_i = 3, C = [64,128,256,512], K = 1, P = 4, learning rate = 0.001

Characteristics	Accuracy	Parameters	Iterations
No Normalization	84.56%	11,164,352	86/100
Normalized with mean 0 and std dev. 1	85.84%	11,164,352	49/50
Normalized with actual mean and std dev. of CIFAR-10	84.25%	11,164,352	34/50

Learning rate = 0.001, N = 2, B = [4,4], F_i = 3, C_i = 64, K = 1, P = 16, Drop Out = 0.5

Optimizer	Characteristics	Accuracy	Parameters	Iterations
AdamW	Normalized true mean, std dev., Data Aug	90.54%	3,905,472	57/60
Adam	Normalized true mean, std dev., Data Aug	90.02%	3,905,472	42/60
AdamW	Normalized mean 0, std dev. 1, Data Aug	91.04%	3,905,472	87/100
AdamW	Gausian Blur	84.48%	3,924,672	42/50

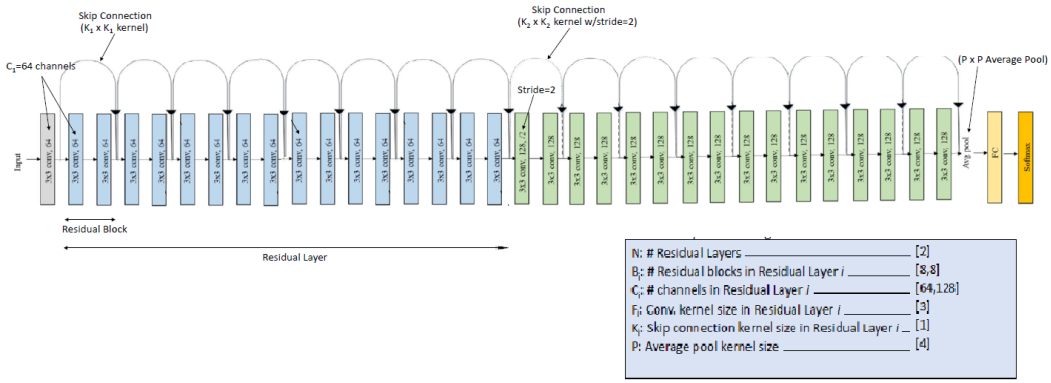
3 Results

The code and the corresponding steps to run it is provided on the following GitHub Link

This section contains the results of the above experimentation. The detailed information about the experiments can be found on this LINK. The drive folder contains the tracker maintained to note the results for changes in hyper-parameters and a folder containing the source codes corresponding to the same.

Following the methodology used above, we arrived at the following ResNet model:

N	*Bi*	*C*	Fi	Ki	P	Parameters
2	[8,8]	[64, 128]	3x3	1x1	4x4	2,905,792



How we trained the model

The model was trained using the following techniques:

- Training data vs Testing data ratio split = 10:1
- Optimizer: AdamW (Learning Rate: 0.001, Weight Decay: 0.0001)
- Batch Size: 128
- Data Augmentation:
  - Random Horizontal Flip: Probability = 0.5,
  - Random Cropping: Size = 32, Padding = 4
- Input Normalization with Mean = [0, 0, 0], Standard deviation = [1, 1, 1]
- Number of Iterations = 60
- Dropout = 0.5

Training results and graphs

Obtained in Figure 1 and Figure 2

4 Conclusion

Hence, we were able to construct a model with the above mentioned hyperparameters and classify CIFAR-10 dataset with a maximum accuracy of 91.62% under 5 million trainable parameters. The model satisifies all the constraints of the given problem statement.

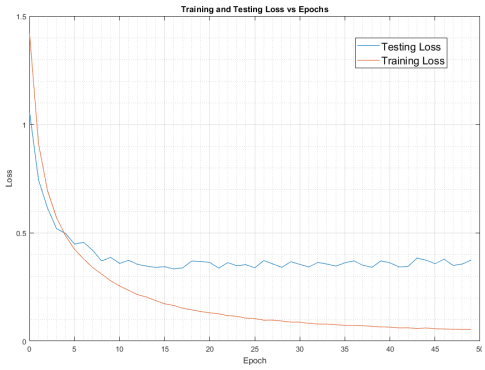


Figure 1: Loss vs Epoch

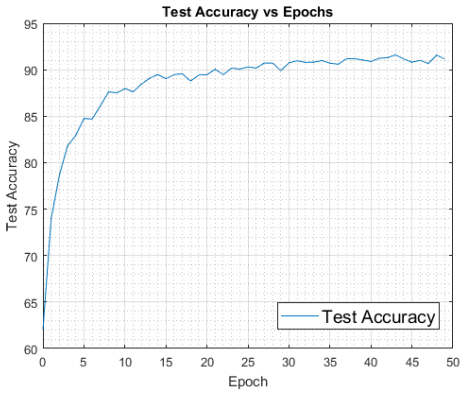


Figure 2: Test Accuracy vs Epoch

5 References

[1] He, Kaiming & Zhang, Xiangyu & Ren, Shaoqing & Sun, Jian. (2016). Deep Residual Learning for Image Recognition. 770-778. 10.1109/CVPR.2016.90.

[2] Shelhamer, Evan & Long, Jonathon & Darrell, Trevor. (2016). Fully Convolutional Networks for Semantic Segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence. 39. 1-1. 10.1109/TPAMI.2016.2572683.

[3] Bello, Irwan & Fedus, William & Du, Xianzhi & Cubuk, Ekin & Srinivas, Aravind & Lin, Tsung-Yi & Shlens, Jonathon & Zoph, Barret. (2021). Revisiting ResNets: Improved Training and Scaling Strategies.

[4] Simonyan, Karen & Zisserman, Andrew. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv 1409.1556.

[5] Szegedy, Christian & Liu, Wei & Jia, Yangqing & Sermanet, Pierre & Reed, Scott & Anguelov, Dragomir & Erhan, Dumitru & Vanhoucke, Vincent & Rabinovich, Andrew. (2015). Going deeper with convolutions. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 1-9. 10.1109/CVPR.2015.7298594.

[6] Farag, Hania & Said, Lamiaa & Rizk, Mohamed & Ahmed, Magdy. (2021). Hyperparameters optimization for ResNet and Xception in the purpose of diagnosing COVID-19. Journal of Intelligent & Fuzzy Systems. 41. 10.3233/JIFS-210925.

[7] Dai, Zhenwei & Heckel, Reinhard. (2019). Channel Normalization in Convolutional Neural Network avoids Vanishing Gradients.

[8] towardsdatascience.com - Understanding and visualizing ResNets

[9] brandonmorris.dev - Building a World-Class CIFAR-10 Model From Scratch

[10] analyticsvidhya.com - A Comprehensive Guide on Deep Learning Optimizers

[11] Piotrowski, Adam & Napiórkowski, Jarosław & Piotrowska, Agnieszka. (2019). Impact of deep learning-based dropout on shallow neural networks applied to stream temperature modelling. Earth-Science Reviews. 201. 103076. 10.1016/j.earscirev.2019.103076.

[12] towardsdatascience.com - Dropout on convolutional layers is weird

[13] stackoverflow.com - Implement dropout to fully connected layer in PyTorch

[14] Kim, Byeonggeun & Chang, Simyung & Lee, Jinkyu & Sung, Dooyong. (2021). Broadcasted Residual Learning for Efficient Keyword Spotting. 4538-4542. 10.21437/Interspeech.2021-383.

[15] Ioffe, Sergey & Szegedy, Christian. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.

[16] nanonets.com - Data Augmentation — How to use Deep Learning when you have Limited Data

[17] medium.com - How Data Augmentation Improves your CNN performance?

[18] medium.com - Normalization Techniques in Deep Neural Networks