NYU Deep Learning Mini Project 1

Şeyda Güzelhan, Kubilay Ülger, Alice Nüz
New York University
sg6896@nyu.edu, ou2007@nyu.edu, an3456@nyu.edu

Abstract

We sought to find the best performing model using less than 5 million parameters on CIFAR-10 test set data with ResNet architecture while modifying 6 architecture specific parameters, doing general hyperparameter tuning, and performing transformations. We first varied the 6 architecture specific parameters one a time keeping other parameters fixed at ResNet18 values. Then based on our understanding of how each parameter affects the final model, we chose a subset of values for each parameter to test in a joint optimization. We ran the best performing parameter assignment from the joint optimization for 500 epochs, achieving a final test set accuracy of 92.5%. For general hyperparameter tuning and transformations, we selected values and transformations successful in the literature. The ResNet architecture parameter assignment was N = 3, $B_i = 2$, $C_i = 85$, $F_i = 3$, P = 8, $K_i = 1$. We used ADAM optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$, a learning rate scheduler where we start with lr=0.001 and decay it 10% every 10 epochs, batch size 64, and various transformations to train our network.

1 Introduction

In many deep learning problems, the final result is strongly dependent on the choice of parameters and hyperparameters. After defining the problem, the metrics, and the loss function (in our case: CIFAR10 classification, classification accuracy, and the cross entropy loss); we generally continue with the model. Model selection includes deciding on the deep learning or machine learning model and is followed by the process of hyper parametrization.

We are required to use ResNet architecture in our mini project. The building blocks of the ResNet model consists of 2 convolutional layers back to back together with a 'skip' connection from input to output. This skip connection is the novelty of the model and it helps the layers to learn identity function more easily and solves the vanishing gradient problem. So, we can train deeper models without losing performance. Figure 1 shows the ResNet-18 model given as an example in the project description. In this project, we are using the same architecture and blocks while we are allowed to tune hyperparameters under a budget constraint of 5 million parameters to achieve the highest test accuracy we can.

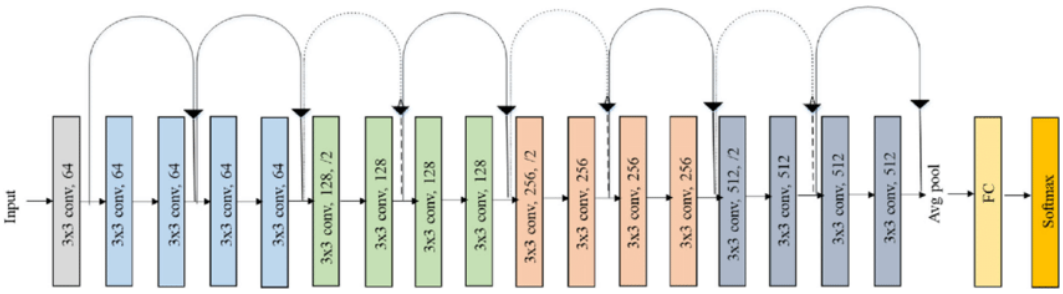

Figure 1: Example network ResNet-18

ResNet model first starts with a convolutional layer, then a set of residual layers made out of residual blocks then there is average pool layer and it ends with fully connected layer with softmax activation for the cross entropy loss.

2 Methodology

We have two major constraints in designing our model. The first one is that we are not allowed change the network architecture except for a set of hyperparameters and the second one is that number of trainable parameters should be less than 5 million. The hyperparameters are the number of residual layers: N, number of residual blocks in layer i: B_i, number of channels in layer 1: C_i, convolutional kernel size in layer i: F_i, Skip connection kernel size in layer i: K_i and average pool kernel size P. We have divided our optimization into two parts: the first one is the model hyperparameter selection and the second one is the training part (data augmentation, optimizer, batch size, regularization et cetera).

Before moving into a more in depth analysis we have started our experiments by first using the original ResNet code given to us and trying some basic parameters so that we can have some baseline. After some trial and error we have decided on starting with ADAM optimizer and a learning rate (lr) of 0.001 and a batch size of 64. In our first set of experiments we did not apply any augmentation or regularization. In the next two subsections we go in detail on how we selected the hyperparameters and training strategy.

2.1 Architectural Hyperparameters: N,B_i,C_1,F_i,K_i,P

We start by analysing the effect of each of the 6 hyper parameters on the test accuracy and number of parameters individually without changing other computational settings. Our starting point is the original ResNet settings from the starter code ($N = 4$, $B_i = 2$, $C_i = 64$, $F_i = 3$, $K_i = 1$, and $P = 3$) which has 11M parameters. For the sake of time, we ran these single parameter trials for 10 epochs. Based on the results for 10 epochs, we decide which settings to test in our next step: joint optimization over 50 epochs. For the sake of simplicity we have chosen B_i, C_i, F_i and K_i to be same across all layers. Although we also looked at asymmetric cases briefly.

In our joint optimization, we selected a subset of the original parameters tested based on criteria outlined in section 3. We run a joint optimization for each combination of parameters along with the augmentations and training model described in section 2.2 for 50 epochs, eliminating settings that exceed 5 million parameters.

We select the best performing combination of parameters from the joint optimization and increase the number of channels such the number of parameters get as close to 5 million as possible. Finally, we run this parameter assignment for 500 epochs and select it as our final result. During this final training, we also tried different regularization settings, optimizers and schedulers.

2.1.1 Preliminary Test Results

In this subsection we presents our results for single parameter analysis described in section 2.1 and give insights on how these findings affected our choices. Note that we used starter code settings with ADAM optimizer, constant lr = 0.001 and without any data augmentation. To start our experiments, we utilized the online tutorials [1] and [2].

Variation of N		
Value	Test Accuracy	# Parameters
2	0.8154	676,682
3	0.8352	2,777,674
4	0.8292	11,173,962
5	0.8136	44,743,754
6	0.8174	178,992,202

Table 1: Table for N vs Test Accuracy

We see in Table 1 marginal improvements in test accuracy when we increase N from 2 to 4. When we increase N by 1, the number of trainable parameters increases by 4. So we see this exponential trend with increasing N. We do not test N=1 as we assume a network must have at least 2 residual layers so that it is a proper ResNet. Because of the exponential increase of number of parameters we decided that it is not feasible to go above 5 layers so in our joint optimization, we test N from 2 to 5.

Variation of B_i		
Value	Test Accuracy	# Parameters
1	0.8016	4,903,242
2	0.8258	11,173,962
3	0.8338	17,444,682
4	0.8333	23,715,402
5	0.8366	29,986,122
6	0.8371	36,256,842
7	0.8481	42,527,562
8	0.8418	48,798,282
9	0.8340	55,069,002
10	0.8333	61,339,722

Table 2: Table for B_i vs Test Accuracy

In Table 2 we see increases in test accuracy when varying B_i from 1 up to 7 while the number of trainable parameters increases by approximately linearly with B_i. As mentioned before, for the sake of simplicity we take $B_i = B_1$ for all i. Since it only affects number of parameters linearly we can test for the whole range from 1 to 7 in our joint optimization.

Variation of C_i		
Value	Test Accuracy	# Parameters
2	0.4722	11,420
4	0.5843	44,622
8	0.6928	176,402
16	0.7407	701,466
32	0.7918	2,797,610
64	0.8292	11,173,962
128	0.8363	44,662,922
256	0.8426	178,585,866

Table 3: Table for C_i vs Test Accuracy

We see statistically significant improvements whenever we double the value of C_i in Table 3. Doubling the value of C_i quadruples the number of trainable parameters. So the number of parameters depends approximately quadratically in C_i. We strictly see that increasing C_i increases test accuracy. We also see C_i allows the most granular variation in number of trainable parameters. While increasing or decreasing other parameters like N by 1 changes the complexity by a factor of 4, if we change C_i from 64 to 65, we see only a marginal increase in number of trainable parameters. Therefore once all other parameters are selected, it is always in our best interest to maximize C_i such that it gets as close to 5 million trainable parameters as possible. Therefore, for each run in our joint optimization, we choose the maximum allowable C_i based on the other parameters.

Variation of F_i		
Value	Test Accuracy	# Parameters
1	0.3128	1,407,562
3	0.8292	11,173,962
5	0.8197	30,702,154

Table 4: Table for F_i vs Test Accuracy

We test only odd kernel sizes because they are symmetric about the origin and avoid aliasing errors seen in even kernels. In Table 4 we see poor performance from a 1x1 kernel and therefore do not consider it in our joint optimization. We see that the difference in performance between a 3x3 kernel and a 5x5 kernel is statistically negligible while the 5x5 kernel almost triples the number of trainable parameters. Therefore, we select $F_i = 3$ as the setting in our final model. Also, as mentioned in [3] small filters were shown to be very effective which further strengthens our decision.

Variation of P		
Value	Test Accuracy	# Parameters
1	0.8244	11,250,762
2	0.8250	11,189,322
3	0.8272	11,173,962
4	0.8255	11,173,962

Table 5: Table for P vs Test Accuracy

Table 5 shows the effect of different pooling size P. Note that in this test we have 4 layers, because we have a stride of 2 at start of each residual layer after the first, the image size halves 3 times. So before the fully connected layer we end up having 4x4 image. P can't be greater than 4, in general for any N, P has to be less than $32/2^{(N-1)}$. We see that size of pooling layer in this case does not have a big impact on the performance. We also see a slight decrease in number of parameters. In light of these results and looking at [4], we have decided to use global average pooling layer ($P = 32/2^{(N-1)}$) for the joint optimization.

Variation of K_i		
Value	Test Accuracy	# Parameters
1	0.8255	11,173,962
3	0.8292	12,550,218
5	0.8197	15,302,730

Table 6: Table for K_i vs Test Accuracy

Table 6 shows the effect of varying K_i, the skip kernel size, on test accuracy. We can see that increasing K_i increases parameter size although less so than F_i because there are less skip layers. There seems to be no significant effect on test accuracy. Note that although we tested for different skip kernel sizes, the reason we had skip kernels was to be able to accommodate the identity function into our network so we decide to stick with the original ResNet idea and keep it as $K_i = 1$.

2.2 Training Hyperparameters

2.2.1 Learning Rate

Since the learning rate can be kept fixed or changed, we tried both ways in our project. We tried different fixed learning rates and different mechanisms and rates of learning rate decay. We were getting around 0.88 accuracies with a fixed 0.0001 learning rate. After changing it to start from 0.001 and decay by 0.9 at each 10 epochs, we reached accuracies above 0.90 within 50 epochs. This is caused by the fact that while we can start with a higher learning rate, as we get close to a minima, lowering the learning rate helps us get a smoother convergence. We continued tuning other parameters with this decaying learning rate.

2.2.2 Optimizer

For the optimizer we have chosen to try SGD and ADAM. We have tried them at the beginning of our tests with the model from starter code. Figure 2 shows the performance of ADAM and SGD optimizers. We can clearly see that for our model with ADAM is a better choice. We have have chosen the stick with default β values of $(0.9$ and $0.999)$.

2.2.3 Batch Size

For the reasons that were explained above, we will be using ADAM optimizer and tuning its batch size hyper parameter. Even though greater batch sizes means a higher computation time in general,

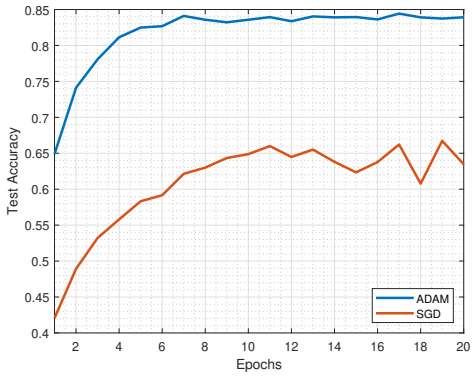

Figure 2: Comparison of performance for two batch sizes

since we are using GPU's, they utilize parallel computing which weakens the effect of increased batch size on the computation time. We are able to increase it and get less noisy gradient estimations. Although we can go for higher batch sizes, we will still be in the "small batch size" regime because in [5] they observe that larger batch causes a degradation in the quality of the model's ability to generalize. So in our tests we have tried batch sizes of 16 and 64. Figure 3 shows comparison of performance for batch sizes of 16 and 64. We can see that in our setting, they both behave similarly. So we have chosen to use batch size of 64 mainly because it capitalized on GPU's parallelization better hence was faster to train.

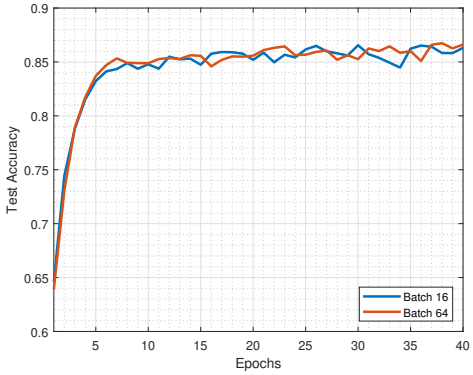

Figure 3: Comparison of performance for two batch sizes

2.2.4 Data Augmentation

Based on our previous knowledge and experiments, we wanted to see how much data augmentation will help. We first checked online tutorials [6] and [7] to see the type of transformations they used. Through the PyTorch documentation for torchvision.transforms, we chose the suitable ones regarding the nature of the daataset and task. For instance, we used random rotation and random flips, because these do not affect our task negatively; a cat is still a cat when it is flipped and rotated, and actually they tend to flip and rotate in real life which enhances our model's capability of generalization. Regarding the data and task, we tried and chose to horizontally flip, rotate by an angle, crop (occlusion robustness), color jittering, and affining.

The final transformations we used:

- RandomCrop(size=[32,32], padding=4)
- RandomAffine(0, shear=10, scale=(0.8,1.2))
- RandomRotation(10)

- ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2)
- RandomHorizontalFlip()

Data Augmentation		
# epochs	Test Acc, No Aug	Test Acc, with Aug
20	0.8463	0.8664
40	0.8549	0.8892

Table 7: Comparison of Data Augmentation vs no Data Augmentation

2.2.5 Regularization

Regularization techniques help us prevent overfitting. The architecture we were given already has batch normalization layers after every convolution layer which introduces some regularization. We have also tried to including dropout layers which randomly zero some weights of the previous layers during training. Dropout layers are originally implemented after fully connected layers [8], which our model has at the last layer. We included that with dropout probability 0.5. The result was significantly worse than what we had with our final model. As an alternative, we tried the method described in [3]. They had a dropout layer inside the residual blocks in between convolutional layers. The dropout probability is usually lower when applied to convolutional layers, so we picked it as 0.2. Although the performance was better than applying dropout after last layer we still didn't get an improvement over our final run. In the end we concluded that batch normalization layers were enough to prevent overfitting.

3 Results

3.1 Joint Optimization

The joint optimization was the last step before we chose our final model. In section 2.1.1 we picked values of $F_i = 3$, $K_i = 1$ and $P = 32/2^{(N-1)}$ based on our initial results. In this section we show our results for joint optimization of N, B and C. We have tested for $N \in \{2, 3, 4, 5\}$ and for each N we tried $B \in \{1, 2, 3, 4, 5, 6, 7\}$ for every permutation less than 5M parameters. We chose the maximum C that gave us a total number of parameters less than 5M. Table 7 shows the values that we tested, their corresponding number of parameters and the test accuracy after 50 epochs. In these tests, we have also implemented our data augmentations so the results are in general significantly better than those in the previous section.

By looking at these 22 different sets of parameters, we can safely say we have tested for large variety of wide and deep networks. For our next step, we compared the two models with the best results highlighted in the table in red by training them for 500 epochs. This comparison is also satisfying in the sense that, #3 is a relatively wide ResNet while the #9 is a deeper ResNet so we are comparing fundamentally different networks. Figure 4 shows Train Loss vs Epochs and Test Error vs Epochs for the two best candidates. For both models, test error goes below 8%. We can see from the plots that Test #9 slightly outperforms Test #3 although the difference in test error is around 0.02%. In light of this we have chosen Test #9 as our final model in section 3.2

3.2 Final Model

In this section we give our final results. Our network model parameters are N=3, $B_i = 2$ for all i, $C_1 = 85$, $F_i = 3$, $K_i = 1$ and $P = 8$. We have used ADAM optimizer to train our model with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We have also used a learning rate scheduler where we start with lr=0.001 and decay it 10% every 10 epochs. We have applied the random transformations mentioned in section 2.2.4. Detailed network structure can be seen in Figure 5.

The figures below shows loss/accuracy graphs of our final training. Figure 6 shows test and train loss vs epoch number plots, we can see that train loss decays rather smoothly because of our choice of optimizer and learning rate decay. We can see that it plateaus around epoch 300. Test loss also reaches a stable point after epoch 300.

Joint Optimization					
Test #	N	B	C	# Parameters	Test Accuracy
1	2	1	256	4,943,114	0.8644
2	2	2	172	4,916,630	0.8926
3	2	3	140	4,990,870	0.9066
4	2	4	120	4,965,730	0.9
5	2	5	106	4,920,212	0.9009
6	2	6	96	4,962,154	0.8988
7	2	7	90	4,985,740	0.9019
8	3	1	128	4,911,754	0.8943
9	3	2	85	4,895,755	0.9104
10	3	3	68	4,884,926	0.9041
11	3	4	58	4,835,412	0.907
12	3	5	52	4,905,742	0.8985
13	4	1	64	4,903,242	0.8974
14	4	2	42	4,815,940	0.8939
15	4	3	34	4,928,412	0.8979
16	4	4	29	4,875,287	0.89
17	4	5	26	4,955,636	0.8828
18	5	1	32	4,904,618	0.8772
19	5	2	20	4,824,403	0.8633
20	5	3	17	4,939,173	0.8656
21	5	4	14	4,556,072	0.8617
22	5	5	13	4,968,077	0.8614

Table 8: Results for joint optimization over N, B and C

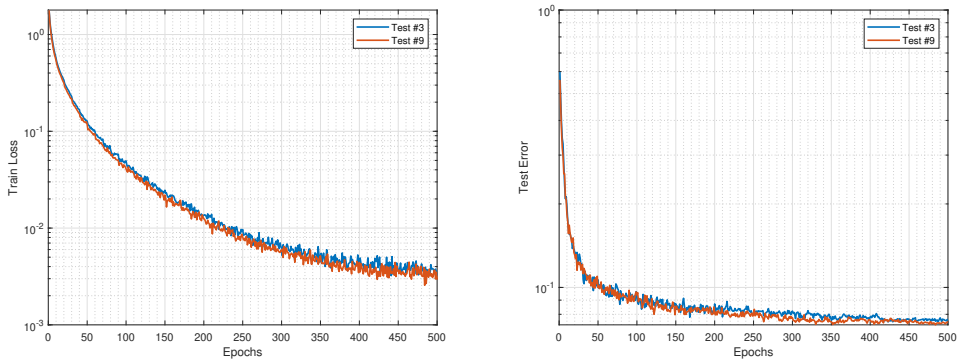

Figure 4: Comparison of Test #3 and Test #9

Figure 6 also shows the train and test accuracy vs epoch number plots. Similar to the loss plots, both accuracies flatten after around 300 epochs. The training accuracy is almost 100% while the final test accuracy is around 92.5%.

4 Conclusion

In this project we trained a ResNet model to obtain 92.5% test accuracy on the CIFAR-10 dataset with a parameter budget of 5M. We have identified and tested different combinations of ResNet specifc hyperparameters, general hyperparameters, and transformations. The final ResNet architecture parameter assignment was N = 3, $B_i = 2$, $C_i = 85$, $F_i = 3$, P = 8, $K_i = 1$. We used ADAM optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$, batch size 64, learning rate 0.001 with 10% decay every 10 epochs. The final transformations used were a crop, an affine, a rotation, a color jitter, and finally an horizontal flip.

We learned that we can improve performance from the original ResNet starter code within our computation limit by modifying the parameters N, C_i, and P. The original values in the starter code

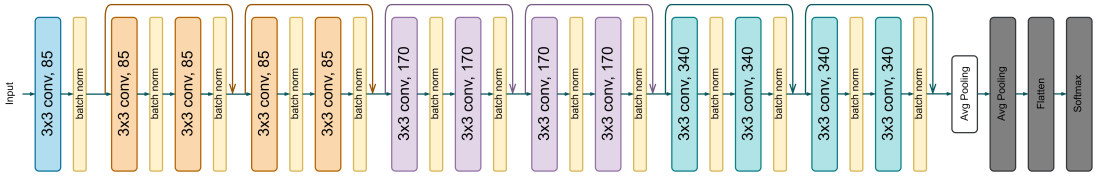

Figure 5: Our Final Network (Replace with the actual one)

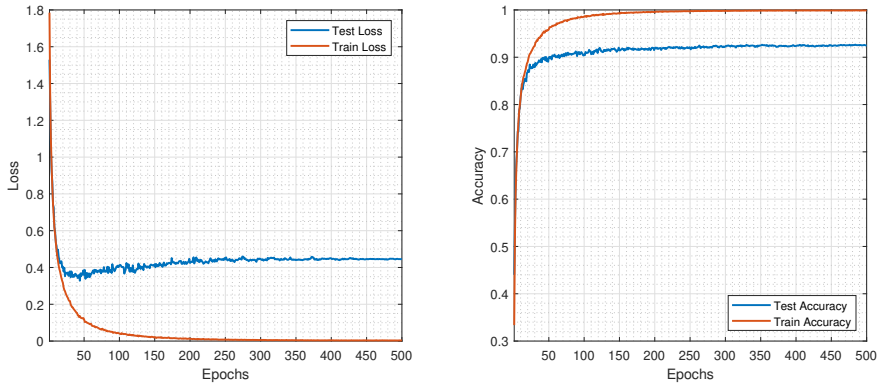

Figure 6: Final Network: Training and Test Loss/Accuracy Graphs

were $N = 4$, $B_i = 2$, $C_i = 64$, $F_i = 3$, $P = 3$, $K_i = 1$ and this assignment exceeded the 5 million parameter limit. We generally saw that increasing the value of a ResNet parameter increased the test accuracy of the model. We saw this benefit plateau or even reverse at a certain value for 5 out 6 ResNet parameters, while increasing the number of channels consistently increased test accuracy for C_i values up to 256 (the largest value we tested). Therefore within our testing scope, we found it to always be advantageous to increase C_i. You can find the code to reproduce our results here in here: https://github.com/nuzalice/deep-learning-mini-project1

References

[1] D. Karani, "Experiments on hyperparameter tuning in deep learning-rules to follow," *Medium*, Nov 2020. [Online]. Available: https://towardsdatascience.com/experiments-on-hyperparameter-tuning-in-deep-learning-rules-to-follow-efe6a5bb60af

[2] Kmldas, "Cifar10 resnet: 90+% accuracy;less than 5 min," Jan 2021. [Online]. Available: https://www.kaggle.com/code/kmldas/cifar10-resnet-90-accuracy-less-than-5-min/notebook

[3] S. Zagoruyko and N. Komodakis, "Wide residual networks," *CoRR*, vol. abs/1605.07146, 2016. [Online]. Available: http://arxiv.org/abs/1605.07146

[4] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[5] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *CoRR*, vol. abs/1609.04836, 2016. [Online]. Available: http://arxiv.org/abs/1609.04836

[6] R. Benenson, "What is the class of this image," *Classification datasets results*. [Online]. Available: http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

[7] "Transforming and augmenting images." [Online]. Available: https://pytorch.org/vision/master/transforms.html

[8] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012. [Online]. Available: http://arxiv.org/abs/1207.0580