Deep Learning Mini Project 1
Cifar 10 with Resnets

Pratap Narra NYU pn2158@nyu.edu
Mudit Saxena NYU ms12768@nyu.edu
Rakshit Lodha NYU rl4563@nyu.edu

Abstract

There have been attempts to make deeper convolutional neural networks (CNN) since their advent as the performance is believed to be tightly related to the complexity of the network. Residual networks (ResNets) represent a powerful type of CNN's architecture which are widely adopted and used in various tasks. The objective of this study was to maximize accuracy on the CIFAR-10 benchmark using the ResNet-18 model while having less than 5M trainable parameters. Moreover this thing is achieved by changing various ResNet hyper-parameters along with different optimizers, data augmentation and data transformation techniques such as gray scaling, cropping, vertical flip , horizontal flip, SGD, ADAM etc.

1 Introduction

Image recognition has been one of the biggest challenges in the field of deep learning within the last 15 years. Through the years, we have found that CNNs have the capability to obtain higher level representation of image features without the use of feature engineering. CNNs have learnable parameters that the input image convolves with at specified strides. Using convolution and stride techniques, the trainable parameters are reduced. Although the trainable parameters are reduced, performance of deep learning models is not compromised since convolution techniques maintain the important features of an input image. Image classifiers are generally tested on the accuracy of testset, but high accuracy can mask a subtle type of model failure. We find that high scoring CNN's on popular benchmarks exhibit troubling pathologies that allow them to display high accuracy even in the absence of semantically salient features. When a model provides a high-confidence decision without salient supporting input features, we say the classifier has overinterpreted its input.

We have used a deep convolutional neural network - called ResNet [4] to perform image classification on the CIFAR-10 [8] dataset. ResNet was worked upon to help with two major issues - degradation and vanishing gradient. Unlike Deep Neural Network layers, residual helps to learn the following features at both low and high level during the process of training the network. In addition, CIFAR-10 is a dataset that consists of 60,000 images with 32x32 dimensionality across three color channels. It has 10 classes such as 'plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck' with each class having 6000 images. Which are further divided into 5000 training images and 1000 testing images per class. In order to get the best model on CIFAR - 10, we need to tune multiple hyperparameterk throughout the network. The hyperparameters being the Residual Layers - 'N', Residual Blocks in residual layer i - 'Bi', Residual Channels in residual layer i 'Ci', Conv kernel size "Fi', Average pool kernel size 'P' and Skip connection kernel size 'Ki' . We used Asynchronous Successive Halving Algorithm (ASHA) to tune all the above mentioned hyperparameters. To further increase the performance of the model, we have used regularization

techniques (dropout), data augmentation techniques (random cropping and horizontal flip) and data transformation techniques (normalization and greyscale).

The structure of the report is as follows, Section 2 Methodology consists of the process of how we went designed out ResNet model. The 3 Results section consists of all our hypothesized architectures and their performances.

2 Methodology

2.1 CNNs

CNN is a neural network that assigns weight and biases to different objects in a particular input image. With the use of appropriate filters, CNNs can capture spatial and temporal connections in an image. CNN architectures generally fit better to image datasets as in image datasets, there are a large number of less-important features. CNNs are composed of one input and output layer with multiple hidden layers. These hidden layers are made of convolutional layers. Also, CNNs usually have Rectified Linear Unit Layer as an activation function.

Since the convolution inherent model is linear so we need to add some non-linearity in the function. We can understand non linearity by considering the example of a simple table. If you look at a table from different angles you will still be able to identify it as a table. This is known as non linear property. Non linear activation layer is supposed to bring non linearity in our network. Pooling layer corresponds to a non-linear layer but is commonly used to down sample our data. It will acquire its input from preceding layer and the output is correlated to the window size. If my window size is 2×2 then it is going to take every 2×2 corresponding window from convolution layer and minimize its dimensions to a single pixel. Common way of implementing pooling is through Max Pooling or average pooling. Down sampling is essential to lower size of the representation otherwise it will become tedious for algorithm on higher dimension data. It further aides in lessening overfitting and extracts features from input tensor. The next layer is called fully connected layer. It is a simple feed-forward neural network and is one of the last few layers in the network. Input to this layer is the output of the final pooling/convolutional layer which is first flattened and then sent as an input to a fully connected layer. Its main motive is to be able to detect the final output categories.

2.2 Hyperparameter tuning

To train any deep learning model there are a number of hyperparameters that have to be set beforehand, but trying out all combinations of hyperparameters to find the best set can be compute-intensive as it requires training a lot of deep learning models. In this work, the Asynchronous Successive Halving Algorithm (ASHA) [1] is used to perform hyperparameter tuning.

2.3 Asynchronous Successive Halving Algorithm (ASHA)

ASHA is designed by exploiting asynchrony and maximizing the parallelism on the Successive Halving Algorithm(SHA) [1]. SHA starts with allocating a uniform budget to all the candidate hyperparameter configurations, i.e if learning rate and momentum are to be picked from the following sets 1e-1,1e-2 and 0.7,0.9 respectively, then there are a total of 4 initial candidate hyperparameter configurations. Then the model performance is evaluated on all the candidate configurations. Each round of promotion is called a rung. Later the top half candidate configurations will be promoted to the next rung for further evaluation and their budget is doubled, this process is repeated until one configuration remains.

In SHA the model must be evaluated on the entire set of configurations to select the next half. ASHA removes this bottleneck [1]. ASHA starts with assigning configurations to workers, when a

Table 1: Parameters of different combinations

Combination	N	C1	B	Parameters
1	5	16	[2,2,2,2,2]	2,811,818
2	5	32	[2,1,1,1,1]	4,935,434
3	4	64	[1,1,1,1]	4,903,242
4	4	32	[4,4,4,2]	3,585,802
5	3	64	[5,4,3]	4,771,146
6	3	64	[3,3,3]	4,327,754
7	2	128	[4,3]	4,401,226
8	1	128	[4]	1,119,562

worker finishes a job instead of waiting for all the workers to finish, the algorithm identifies if for configurations in the top half of each rung that can be promoted to the next one. Else a configuration from the lowest rung will be added to the worker [1].

2.4 Dropout Regularization

Deep Learning models can easily overfit when there are fewer samples in the dataset as generally deep learning models generally have many parameters to learn. Overfitting leads to poor generalization, hence the model's performance will deteriorate on external test sets. To overcome overfitting regularization is used, dropout is a simple way to prevent overfitting. When a dropout layer is added to a neural network few nodes will be temporarily dropped [2]. These nodes are selected randomly. Applying dropout is equivalent to sampling a 'thinned' network [2]. So during training as a few nodes are randomly dropped, a new thinned network is sampled and trained. Therefore training a neural network using dropout can be viewed as training a collection of thinned neural networks with extensive weight sharing, and merging them into one network that picks key properties of each thinned network [2].

2.5 Base Model Architecture

The ResNet model with parameters N = 4, B = [1,1,1,1] and C = 64 is selected as an initial base model to run data augmentation and input normalization experiments. The Skip Kernel size Ki is 1, Conv Kernel size Fi is 3 and Pool Kernel size P is 4.

2.6 Cosine Annealing

Since Cosine Annealing is known to work well with SGD optimizers, to further enhance the performance of our model, we have used Cosine Annealing learning rate scheduler. This scheduler constantly varies the learning rate such that it starts with a large learning rate and then reduces it to a certain minimum value before rapidly increasing it again [3].

3 Results

3.1 Data Augmentation

In the initial round of experimentation we train the base model with and without data augmentation. Then these models are evaluated based on the train, test accuracy. From Table 2 it is clear that the model trained with data augmentation has a better performance compared to the one trained without. The model trained without data augmentation is clearly overfitting as the performance on the test set

Table 2: Data Augmentation Experiments on Base Model

DATA AUGMENTATION	TRAIN ACC	TEST ACC
Yes	91.58	87.16
No	100	83.26

Table 3: Input Normalization Experiments on Base Model

Input Normalization	TRAIN ACC	TEST ACC
Yes	86.08	84.33
No	91.58	87.16

is bad. Therefore we observe data augmentation is good for generalization, so we use data next set of experiments.

3.2 Input Normalization

In the next round of experimentation the base model is trained with input normalization and compared with the best model in Table 2 which doesn't have input normalization. It is clear from Table 3 that the model without input normalization has a better performance. Therefore input normalization is not used in the further rounds of experiments.

3.3 Architectural Changes

Table 4: Performance of different combinations with their best hyperparameters after 25 epochs

Combination	Batch Size	Optimizer	Learning Rate	Train Accuracy	Test Accuracy
1	128	Adam	0.00129	83.17%	67.02%
2	256	SGD	0.03913	86.68%	72.17%
3	128	SGD	0.00887	87.14%	78.15%
4	256	Adam	0.00531	79.09%	70.03%
5	64	Adam	0.00020	87.96%	75.18%
6	128	SGD	0.01809	88.36%	75.92%
7	128	SGD	0.02373	87.99%	72.16%
8	64	Adam	0.0174	79.67%	73.21%

In the next set of experiments, the parameters of ResNet architecture are varied to find the best architecture. For all these experiments ASHA hyperparameter tuning which is discussed in Section xx is used. ASHA is used to find the best Learning rate, Optimizer, and Batch size for these models. The learning rate range is [1e-5, 1e-1], the batch size is selected from [16,64,128,256] and the optimizer is selected as one of the following [Adam, SGD, Adagrad]. All the architectures have the same kernel size of XX and YY for convolutional layers and skip connections. The average pool kernel size is also uniform across all the architectures. The number of residual layers(N), residual blocks in residual layer i(Bi), initial channel size(C1) are varied across all the architectures. Based on these variations 8 combinations are defined, these are depcited in Table 1

Table 5: Performance of top 4 combinations

Combination	Train Accuracy	Test Accuracy
1	99.99%	93.24%
6	99.99%	94.80%
4	99.98%	91.67%
5	94.68%	89.53%

Table 4 has the results for all the combinations, i.e the best configuration of the learning rate, optimizer, and batch size selected by ASHA. Moreover, the Train and Test accuracy are also provided in Table 4. Now we have selected the top 4 combinations and further trained those models after adding dropout regularization and using Cosine Annealing learning rate scheduler. These top four models are trained until the loss is saturated. The performance of these models can be found in Table 5. From the Table 5 it is clear that Combination 6 is the best of all top 4 combinations. So this combination is selected as the final architecture.

3.4 Training the final Architecture

The best architecture selected in the previous experiment(Combination 6) has a learning rate 0.01809, batch size 128 and optimizer SGD. The model is trained until the loss is saturated for 200 epochs. The final Architecture's Train Accuracy is 99.99% and Test Accuracy is 94.8%. Figure 1 depicts the architecture of the final model. Figures 2, 3 depict the accuracy and loss history of the model during training.

4 Conclusion

In this project through various meticulous experiments and model analysis, we trained our network to maximize accuracy on the CIFAR 10 benchmark. We achieved the best accuracy on testset of CIFAR-10 using number of residual layers = 3, number of blocks per layer = 3, input channel size = 64, input batch size =128, optimizer = SGD, learning rate = 0.01809, regularization technique = dropout, data augmentation = random crop and flip horizontal. This model yielded **94.80%** accuracy on test data.

The weights of the best model have been submitted as a project1_model.pt file and the model architecture has been submitted as a project1_model.py file. We as a team learnt a lot from this mini project and look forward to the next ones!

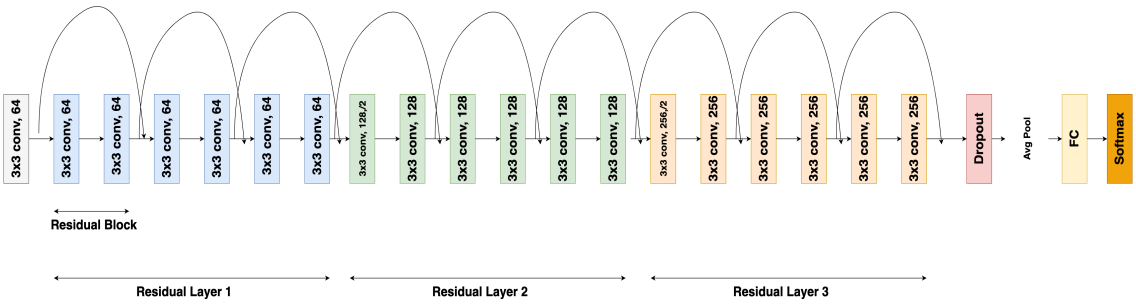
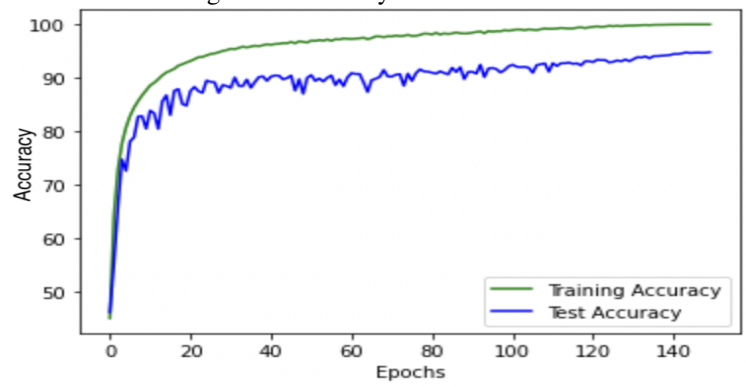


Figure 1: Final model Architecture

Figure 2: Accuracy Of Final Model



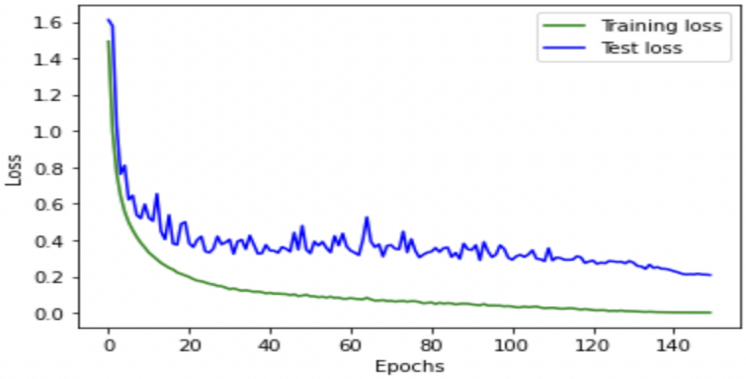


Figure 3: Loss of the final Model

References

[1] Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Ben-Tzur, J., Hardt, M., ... Talwalkar, A. (2020). A system for massively parallel hyperparameter tuning. Proceedings of Machine Learning and Systems, 2, 230-246.

[2] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958.

[3] Loshchilov, I., Hutter, F. (2016). Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983.

[4] He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[5] LeCun, Y., Bengio, Y., Hinton, G. (2015). Deep learning. nature, 521(7553), 436-444.

[6] https://pytorch.org/tutorials/beginner/hyperparameter$_{t}uning_{t}utorial.html$

[7] https://github.com/kuangliu/pytorch-cifar

[8] Krizhevsky, A., Hinton, G. (2009). Learning multiple layers of features from tiny images. [9] Doon, R., Rawat, T. K., Gautam, S. (2018, November). Cifar-10 classification using deep convolutional neural network. In 2018 IEEE Punecon (pp. 1-5). IEEE.