Mini-Project: ResNet for CIFAR-10

Deepti Preta Gouthaman
New York University
dg3781@nyu.edu

Nobel Dang
New York University
nd2100@nyu.edu

Shivansh Sharma
New York University
ss14890@nyu.edu

Abstract

Deep Convolutional Neural Networks are expected to perform better when there are more layers. However, when they become too deep, it results in the vanishing gradients problem. Thus, the transfer of important information from the initial layers to the end layers vanishes affecting the model's performance. This problem can be solved using ResNets (He et al. 2015) [1]. In this paper we designed a ResNet model to classify images in CIFAR-10 dataset. After tuning the hyper-parameters and applying data augmentation techniques, with just three million parameters our model is able to classify images with 94.85% accuracy. Code: *https://github.com/nobeldang/deep-learning-resnet-CIFAR10.*

1. Introduction

Complex machine learning problems can be solved by adding layers to a deep neural network which results in improved performance. However, there is a limit for the depth of CNNs, after that limit the model stops learning and the performance starts degrading. This problem can be tackled by using Residual Networks or ResNets (He et al. 2015) [1].

ResNet consists of N Residual layers which contain one or more residual blocks. Between each residual block skip connections or shortcuts are used to jump over some layers. Typical ResNet models are implemented with double- or triple- layer skips that contain nonlinearities (ReLU) and batch normalization in between. Models with several parallel skips are referred to as DenseNets. In figure 1, the connection that skips some layers of the model is the skip connection.

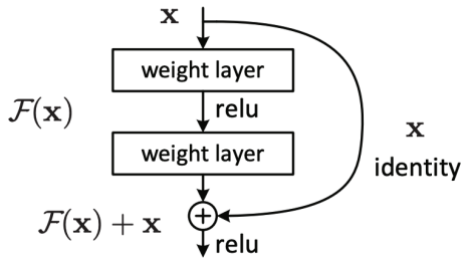


Figure 1. Residual Block
Source: Deep Residual Learning for Image Recognition [1]

The output of the block after skip connection is:

$$H(x) = F(x) + x$$

Unlike the layers in a traditional network, instead of learning true output H(x), the layers in a residual block are learning the residual F(x). Each residual layer has two convolution layers, each with batch-normalization and Relu activation. Multiple such blocks are applied and, in the end, average pooling and a fully connected layer is applied based on the number of classes.

2. Methodology

We started by creating training, validation and test dataset from CIFAR-10. Training data is around 50,000 images and test data is around 10,000 images. Atop of this we performed data augmentation in the hope our proposed model would be able to learn more nuances from the images. For data augmentation we performed random crops and horizontal flips.

We then took the opportunity to test and train various variations based on the ResNet-18 architecture as provided here. We modified the architecture to fit the condition that parameters should be < 5 million. We made tweaks on ResNet-18 architecture and created ResNet-10 as well as ResNet-14 architectures whose efficacy we experimented using CIFAR-10. In ResNet-10 we proposed eight blocks (as in the original ResNet-18) but each block with only one Convolutional layer rather than two. Whereas in ResNet-18 we put forth six blocks and each block with two Convolutional layers. So, in ResNet-14 architecture we basically removed the last two basic blocks of ResNet-18.

Model	**# Parameters**	**# Basic Blocks**
ResNet-10	4,903,242	3
ResNet-18	2,777,674	4

Table 1: Model Architectures

After deciding with the architectures, we investigated with different values of hyperparameters that we passed using the command line for training the model.

Parameters and Hyper-params	**Values**	**Default**
Learning Rate	{0.001, 0.01 0.03}	0.01
Optimizer	{SGD, Adam, adadelta, rmsprop}	SGD
Batch Size	{64, 128, 256}	128
Epochs	{50, 100, 150, 200}	50
Weight Decay	{5e-4}	5e-4

Table 2: Parameters and Hyperparameters with domains.

2.1 Architectural Considerations:

As we devised two architectures: ResNet-10 and ResNet-14.

Layer Name	**ResNet14**	**ResNet10**
Convolutional	Conv_1	Conv_1
Max Pool	MaxPool_1	MaxPool_1
Conv_2x with shortcuts	[3*3, 64] X 2 [3*3, 64] X 2	[3*3, 64] X 1 [3*3, 64] X 1
Conv_3x with shortcuts	[3*3, 64] X 2 [3*3, 64] X 2	[3*3, 64] X 1 [3*3, 64] X 1
Conv_4x with shortcuts	[3*3, 64] X 2 [3*3, 64] X 2	[3*3, 64] X 1 [3*3, 64] X 1
Conv_5x with shortcuts	None	[3*3, 64] X 1 [3*3, 64] X 1
Average Pool	AvgPool_1	AvgPool_1
Fully Connected	fcs_1	fcs_1
Softmax	softmax_1	softmax_1

Tabe 3: Residual model architectures

After experimentation with various hyperparameters for the classification task on CIFAR-10 we concluded that ResNet-14 architecture is better. This might be because it has more convolutional layers so it's able to capture more intricate features and has a more receptive field as well.

Figure 2 shows each Conv_ix for ResNet14 is like:

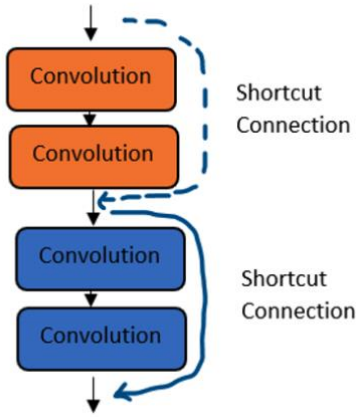


Figure 2. Conv_ix Layer

After experimentation with various hyperparameters for the classification task on CIFAR-10 we concluded that ResNet-14 architecture is better. This might be because it has more convolutional layers so it's able to capture more intricate features and has a more receptive field as well.

We performed training with architectures and tried changing hyperparameters methodologically and iteratively. We did comparative analysis on 15 models with different combinations. Initially we started with ResNet-10 and tried it with different optimizers like SGD (Robinds et al. 1951) [3] and adadelta where SGD performed better. Then again with SGD and adadelta with the same learning rate we increased our number of epochs (from 50 to 100) to see if the test accuracy increases as the model, was not overfitting. And to no surprise the test accuracy also increased with a greater number of epochs with both SGD and adadelta but the SGD outperformed adadelta. The bump in test accuracy was little but we were aiming to have more increase in test accuracy.

That's when we thought of using a deeper ResNet14 architecture. When performed with the same hyperparameters we saw an increase in test accuracy from 91.05 to 91.76 %. We still observe the trend where there was not much case of overfitting. Therefore, we tried decreasing our learning rate from 0.03 to 0.01 and increased our epochs from 100 to 150.

After that, since we were tuning hyperparameters we tried decreasing the batch size for training from 128 to 64 thinking that one gradient step will be influenced by less number of training points and therefore it should be able to take gradient steps more optimally. That's when we increased our maximum testing accuracy of 94.85 %. Then we played around with different optimizers where we observed that rmsprop didn't perform well. We also observed that Adam (Kingma et al. 2015) [4] also performs well enough (second best) but it needed more number of epochs and lesser learning rate and hence more training time. However, Adadelta (Zeiler 2012) [5] did not perform as expected.

2.2 Optimizers and Learning Rate:

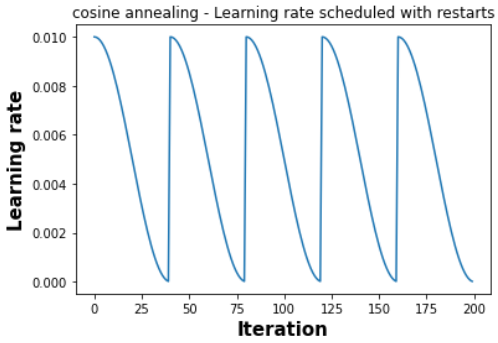
We used four optimizers to test our models, which are sgd, adadelta, adam, and rmsprop. The accuracies achieved with these models are shown in the figure: Optimizers vs accuracy, represents the best accuracies for the four different optimizers with varying learning rates.
As the training progresses the learning rate scheduler adjusts the learning rate between epochs. Earlier in the training, the learning rate is set to be large which later on decreases with the increase in the number of epochs. The most Common learning rate schedules include time-based decay, step decay, and exponential decay. We used stepwise decay and cosine annealing for our model. In the step decay learning rate scheduler, the learning rate is dropped by a factor after every few epochs. In this case, we drop the learning rate by half every 10 epochs.
In our code, we used the step decay function as an argument and returned the updated learning rate for the four different optimizers. Given below is the mathematical representation of Step decay:

$$lr = lr_o * drop^{floor(\frac{epoch}{epochs\_drop})}$$

2.3 Cosine Annealing:

An effective snapshot ensemble requires training a neural network with an aggressive learning rate schedule and cosine annealing best represents such an aggressive learning rate schedule. As shown in the figure 3. the learning rate is set high and then is dropped relatively to a minimum value near zero before it is increased again.
Below is the Equation for the Cosine Annealing Learning Rate Schedule. Where a(t) is the learning rate at epoch t, a0

$$\alpha(t) = \frac{\alpha_0}{2}\left(\cos\left(\frac{\pi \mathrm{mod}(t-1, \lceil T/M \rceil)}{\lceil T/M \rceil}\right) + 1\right)$$



is the maximum learning rate, T is the total epochs, M is the number of cycles, mod is the modulo operation, and square brackets indicate a floor operation. (Huang et al. 2017) [2]

Figure 3. Cosine Annealing

Figure 4. shows the accuracies of two 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.

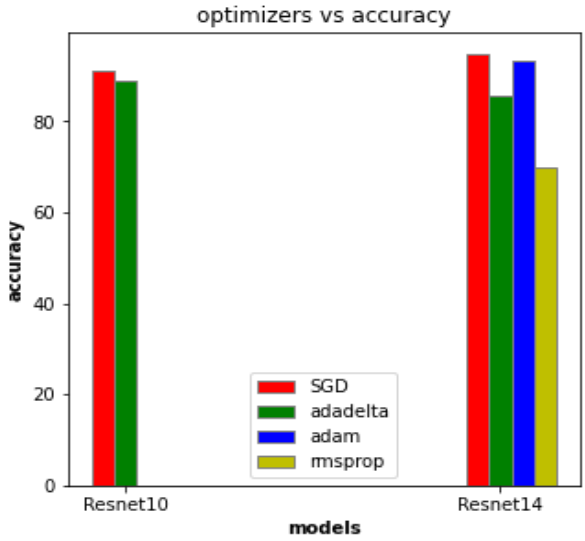


Figure 4. Optimizers vs Accuracy

3. Results

After testing ResNet-10 and ResNet-14 architectures with different hyperparameters, the best test accuracy that we got was 94.85 % with ResNet-14 with approx. 2.78 million parameters.

The hyperparameters responsible for results in figure 5. were: learning rate of 0.01, batch size of 64 and number of epochs as 150 with SGD optimizer.

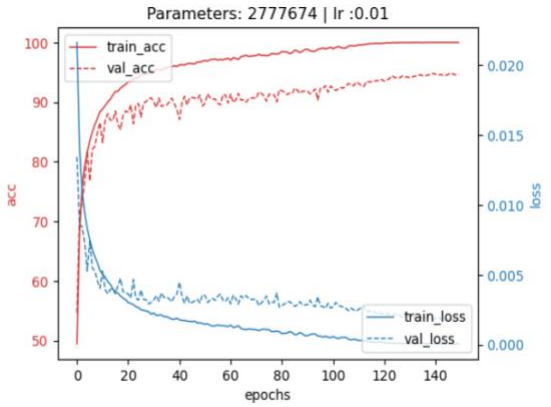


Figure 5. Accuracy and Loss for best model

The second-best model with test accuracy of 93.26 % was ResNet-14 with Adam optimizer.

The hyperparameters that concluded results in figure 6. were: learning rate of 0.001, batch size of 256 and number of epochs as 200 with Adam optimizer.

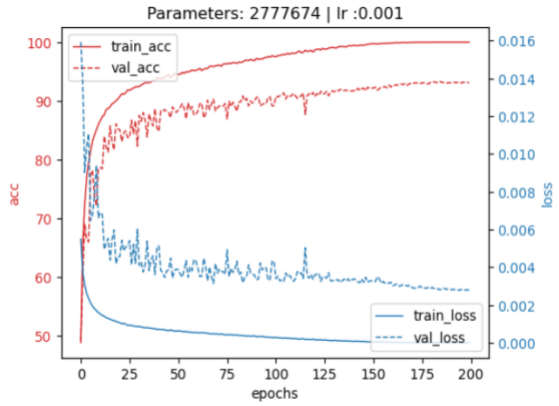


Figure 6. Accuracy and loss for second best model.

All the logs for 15 different models are maintained here.

References

[1] K. He, X. Zhang, S. Ren, J. Sun. Deep Residual Learning for Image Recognition. Computer Vision and Pattern Recognition (cs.CV). arXiv:1512.03385

[2] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, K. Q. Weinberger. Snapshot Ensembles: Train 1, get M for free. Machine Learning (cs.LG). arXiv:1704.00109

[3] H. Robinds and S. Monro, "A stochastic approximationmethod," Annals of Mathematical Statistics, vol. 22, pp.400–407, 1951.

[4] D. P. Kingma, J. Ba. Adam: A Method for Stochastic Optimization. ICLR 2015. arXiv:1412.6980

[5] M.D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. Machine Learning (cs.LG). arXiv:1212.5701