ResNet with Hyper Parameters Tuning

Mugdha Rane
mhr8596@nyu.edu
New York University

Sayali Jathar
snj4459@nyu.edu
New York University

Kawarjeet Yadav
ky2149@nyu.edu
New York University

Abstract

While more complex to train, deep neural networks perform well on picture classification tasks. It regularly takes a ton of time and more computational capacity to prepare further deeper neural networks. When compared to neural networks, deep residual networks (ResNets) can speed up the training process and improve accuracy. ResNets build on convolutional neural networks by adding a basic skip connection parallel to the layers. We have first developed a ResNet model that can perform the image classification task on the CIFAR-10 dataset with high accuracy, and then we compare its performance by changing several parameters of the model for attaining better accuracy. ResNets are more prone to overfitting despite higher accuracy. Our findings support this.

1 Introduction

Of Late, Convolutional Neural Networks (CNNs) have accomplished great progression in the field of image classification. Deep convolutional neural nets (CNNs) feature a layered structure with convolutional filters in each layer. Feature vectors for the following layer are generated by convolving these filters with the input image, and they may be easily learned by sharing parameters. Convolutional neural networks have early layers that represent low-level local features like edges and color contrasts, whereas deeper layers strive to capture more complicated forms and are more specialized Deepening the network can improve the classification performance of CNNs by increasing the variety and specificity of these convolutional filters. Although deep networks can perform better in classification most of the time, they are more difficult to train, especially in datasets like CIFAR-10 which require very deep network depth.

When deeper networks are able to start converging, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly.

1.1 Residual Blocks

To deal with problem residual blocks are added to the network. ResNets differ from other convolutional networks in that they feature parallel shortcut connections to their usual convolutional layers. These shortcut connections, unlike convolution layers, are always active, allowing gradients to quickly back propagate via them, resulting in faster training. In this project, we'll look into ResNets and learn more about how to use them to increase the accuracy of training on CIFAR-10 dataset[**?**]. A residual block contains two convolutional layers with a skip connection from the block's input to the block's output. You can load a pretrained version of the network trained on more than a million images from the CIFAR-10 database [1]. The pretrained network can classify images into several object categories, such as airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

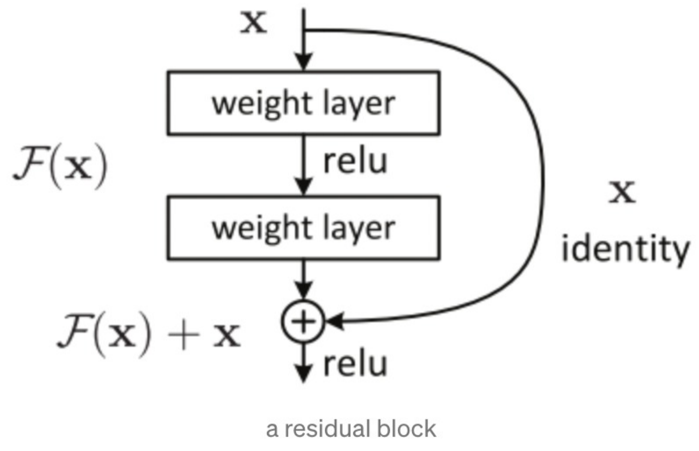

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu
a residual block

Figure 1: Residual Block

1.2 ResNet Architecture

In the project we have used ResNet-18 which is a convolutional neural network that is 18 layers deep. ResNet-18 is just another ResNet Architecture among many others. We have trained this architecture by changing multiple hyper-parameters to achieve highest possible accuracy.

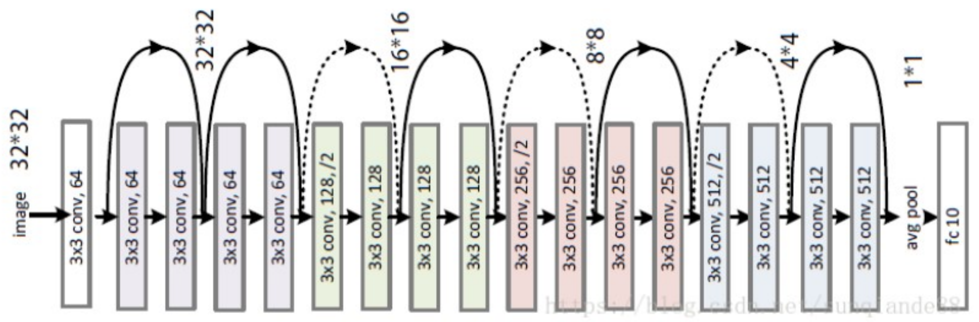


Figure 2: ResNet-18 Architecture

]

1.3 Optimizers

It is a way for updating various hyper parameters that can reduce losses with little effort. Let's look at some of the PyTorch framework's optimizers classes:

- The SGD, or Stochastic Gradient Optimizer, is a weighted optimizer that updates the weights for each training sample or a small portion of data. SGD with momentum speeds up the optimization and makes it easier to avoid local minima.
- Adam optimizer a.k.a Adaptive Moment Estimation is one of the most common optimizers. For greater convergence, Adam Optimizer employs both momentum and adaptive learning rate. This is one of the most extensively used optimizers for training neural networks in practice.

- The Adaptive Gradient Algorithm (Adagrad) is a gradient-based optimization algorithm that increases performance on problems with sparse gradients by giving each parameter its own learning rate. Adagrad penalizes the learning rate for frequently updated parameters, favoring sparse parameters, which are not updated as frequently. However, there are certain disadvantages, such as the fact that it is computationally expensive and that the learning rate is falling, making training slow.

2 Literature Review

Deep learning (DL) is a sophisticated machine learning field that has seen significant progress in a variety of fields. Many study topics, such as computer vision, object recognition, and speech recognition, have been studied in the recent decade. We reviewed several articles and papers to understand the past innovations and challenges faced by those architectures.

The article written by Sam Gross from Facebook AI Research and Michael Wilber from Cornell-Tech, mentions the effect of model depth and its effect on large sets like ImageNet and CIFAR-10. It also answers multiple questions related to batch normalisation and placement of ReLU layers at the end of each residual block. This article is based on Microsoft research on: Deep Residual Learning for Image Recognition carried on by researchers: Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun.

Deep Residual Learning for Image Recognition provides in depth comparison of several models based on ImageNet and CIFAR-10 datasets. It explains in depth study of several layered ResNet architectures and their performances on datasets. It further explains the effect of addition and/or deletion of layers, blocks and function on the training accuracy.

Train longer, generalize better: closing the generalization gap in large batch training of neural networks by Elad Hoffer , Itay Hubara, Daniel Soudry discussed the effect of batch size in the training of a model.

Study of Residual Networks for Image Recognition paper authored by Mohammad Sadegh Ebrahimi and Hossein Karkeh Abadi further explained the problem of Vanishing / exploding gradients and harder optimizations which emphasize the introduction of residual blocks in an architecture.

3 CIFAR-10 Dataset

We conducted more studies on the CIFAR-10 dataset [20], which consists of 50k training images and 10k testing images in 10 classes. We present experiments trained on the training set and evaluated on the test set. The CIFAR-10 dataset contains 60000 32x32 color images divided into ten classes, each with 6000 images. There are 50000 photos for training and 10,000 images for testing.

Each of the 10000 photos in the dataset is separated into five training batches and one test batch. The test batch contains exactly 1000 photographs from each class, chosen at random. The remaining photographs are distributed in training batches in a random order, however some batches may contain more images from one class than others. Each image is in color, but is just 32×32 pixels in size, so the input is a vector of $32 \times 32 \times 3 = 3072$ real values.

4 Implementation

We use an 18 layer architecture to train our model. Before training the model, we apply augmentation to the images. The images are padded by 4, horizontally flipped, and converted to tensors for implementation on the CUDA device. We adopt batch normalization right after convolution and before activation. We use Adam optimizer with varying batch sizes. For weight decay, we divide the learning rate by 3 after every 20 epochs. We do not use any dropouts. The visualization of the loss and accuracy is carried out using Pytorch's Tensorboard.

5 Experiments

We manipulated several hyper-parameters to try and achieve highest possible accuracy without over-fitting the model.

To tune our hyper-parameters for best accuracy, we created variables to hard-code the values of the optimizer, the learning rate, and the batch size. This allowed us to train the model for batch sizes (100, 500, 1000), learning rate (0.1, 0.01. 0.001) and batch shuffle(True, False). This resulted in a total of 18 hyper-parameters combination. The hyper-parameter combination with the best accuracy was then selected.

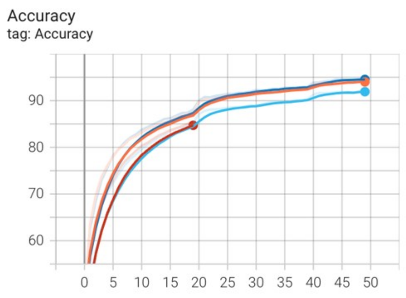
6 Result

Below is the table listing all the training and testing accuracy that we obtained after tuning batch size, learning rate and batch shuffle.

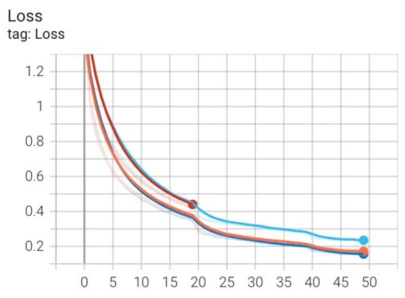
No. of Epochs	Batch Size	Learning Rate	Batch Shuffle	Training Accuracy	Testing Accuracy
50	100	0.1	True	73.48	72.05
50	100	0.1	False	77.07	75.13
50	500	0.1	True	85.12	80.83
50	500	0.1	False	86.36	83.00
50	1000	0.1	True	84.75	81.27
50	1000	0.1	False	85.10	80.58
50	100	0.01	True	95.54	88.30
50	100	0.01	False	95.87	88.16
50	500	0.01	True	95.34	87.87
50	500	0.01	False	95.81	89.12
50	1000	0.01	True	93.81	85.20
50	1000	0.01	False	94.14	86.51
50	100	0.001	True	94.07	88.29
50	100	0.001	False	94.61	87.24
50	500	0.001	True	92.04	86.69
50	500	0.001	False	92.43	86.20
50	1000	0.001	True	90.17	85.29
50	1000	0.001	False	90.67	85.30

7 Observation

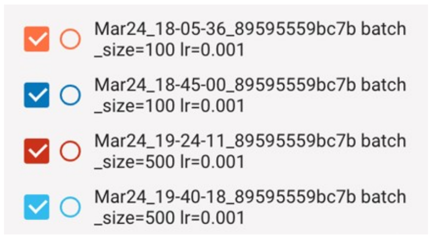
- We obtain the best accuracy when number of epochs is 50, batch size is 100, learning rate is 0.01, and batch shuffle is set to true. The accuracy obtained in this case is 95.54% for training dataset and 88.3% for testing dataset.
- We notice that the batch shuffle does not have any significant improvement in the accuracy for a set number of epoch, batch size and learning rate.
- Also as the batch size increases from the 100 to 500, we see a significant improvement in the accuracy. However, as we increase the batch size to 1000, the accuracy reduces.
- The Accuracy vs Epochs graph and the Loss vs Epochs graph in Figure 3 tells us that the accuracy somewhat saturates around epoch number 30, and does not increase too much from there on.
- We obtain the number of trainable parameters for our architecture to be 195,738.



(a) Accuracy vs Epochs



(b) Loss vs Epochs



✓ ○ Mar24_18-05-36_89595559bc7b batch _size=100 lr=0.001

✓ ○ Mar24_18-45-00_89595559bc7b batch _size=100 lr=0.001

✓ ○ Mar24_19-24-11_89595559bc7b batch _size=500 lr=0.001

✓ ○ Mar24_19-40-18_89595559bc7b batch _size=500 lr=0.001

(c) label 2

Figure 3: Accuracy and Loss Plot

8 Conclusion

The complexity of a neural increases with an increase in the depth. Resnets are one of the excellent architectures to take care of the vanishing/ exploding gradient issue for very deep neural networks. For an 18 layer resnet as implemented by us, the accuracy of the model is dependent on a number of factors. It is not necessary that greater values of these parameters, the better will be the results. We can use empirical analysis to find the best combination for these hyperparameters.

9 Acknowledgments

References

[1] Alexander, J.A. & Mozer, M.C. (1995) Template-based algorithms for connectionist rule extraction. In G. Tesauro, D. S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems 7*, pp. 609-616. Cambridge, MA: MIT Press.

[2] Bower, J.M. & Beeman, D. (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* New York: TELOS/Springer-Verlag.

[3] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* **15**(7):5249-5262.

[4] http://torch.ch/blog/2016/02/04/resnets.html

[5] Elad Hoffer, Itay Hubara, Daniel Soudry (2018) Train longer, generalize better: closing the generalization gap in large batch training of neural networks. https://arxiv.org/pdf/1705.08741.pdf

[6] Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun (2015) Deep Residual Learning for Image Recognition https://arxiv.org/pdf/1512.03385.pdf

GitHub Link

https://github.com/Kawarjeet/DeepLearning.git