Comparison on VGGNet and LeNet Using Stochastic Gradient Descent and Adam Optimizer

David Tea
University of California San Diego

Abstract

Convolutional Networks have been responsible for the popularity of of modern neural networks. Starting with Yann LeCun's LeNet in the 1990's which was first used on the MNIST dataset[1]. *Then from the ILSVRC challenge, we had AlexNet, ZF Net, GoogLeNet, VGGNet, ResNet, and others. This projects aims to compare the LeNet and VGGNet on the CIFAR-10 dataset.*

1. Introduction

Convolutional neural networks have been mainly used to great effect for image classification[1]. Most notable is LeNet on the MNIST dataset which showed the power of convolutional neural networks and paved the way for other convolutional networks to come into being, like VGGNet[1]. Convolutional neural networks are a layered neural network. It splits the image up into features and performs a convolution, pooling, or activation function for each layer. These help train the convolutional neural network on each separate feature of the input images. Each layer has a number of parameters that can be modified. Convolutions can change the number of features an image will be broken into and how big those features will be, pooling layers can change the size of the features that are pooled, and activation layers have many different activation functions like ReLu, sigmoid, and tanh.

This project will focus on the LeNet model and the VGGNet model. Each model takes in images from the CIFAR-10 dataset and before training begins, the images are subjected to random modifications of horizontal and vertical shifts and horizontal flips to better help the networks learn the dataset. LeNet's architecture consists of a convolution layer, a ReLu activation layer, a max pooling layer, a convolution layer, a ReLu activation layer, a max pooling layer, a hidden layer to flatten and ReLu activation and the output layer using the softmax activation[3].

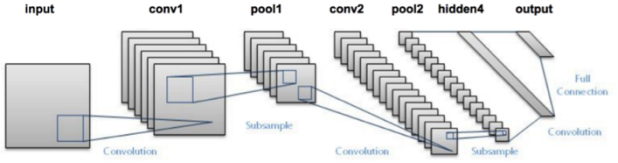

Figure 1. LeNet Architecture. Picture from: http://www.pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python/

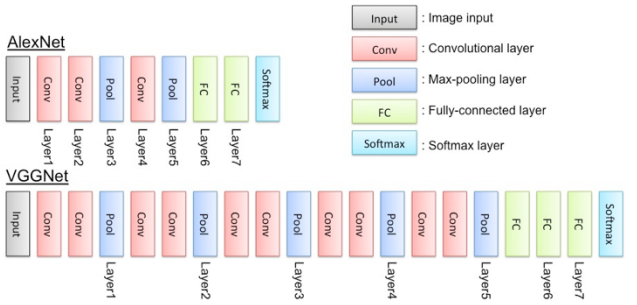

Figure 2. The VGGNet architecture. This neural networks has 16 layers consisting of various convolutions, max pooling, ReLu activations, and a softmax output layer. http://www.hirokatsukataoka.net/research/cnnfeatureevaluation/cnnfeatureevaluation.html

The VGGNet architecture consists of 16 layers, each with two convolution layers with ReLu activation and a max pooling layer[2]. VGGNet was the runner-up in the ILSVRC 2014 challenge. VGGNet's uniqueness was its 16 layer deep network and the exclusive use of 3x3 convolutional layers and 2x2 max pooling layers[2].

The CIFAR-10 dataset consists of 60000 32x32 color images, separated into 10 different classes. The classes are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each class contains 6000 images and are further separated into training and test sets, with 5000 in the training set and 1000 in the test set.

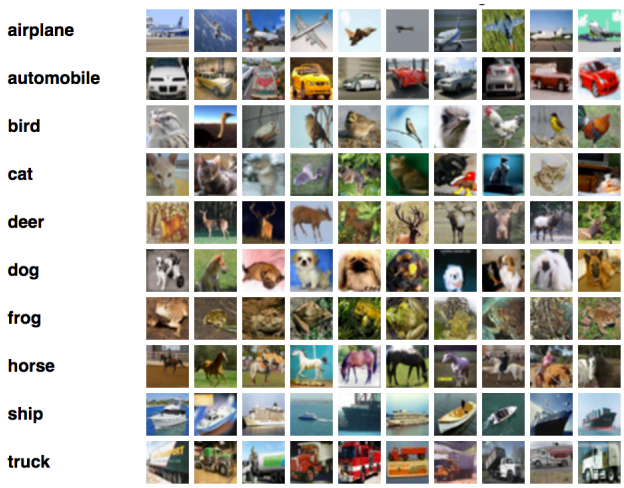

Figure 3. Sampling of CIFAR-10 dataset with 10 examples for each class. Picture taken from https://www.cs.toronto.edu/~kriz/cifar.html

2. Implementation

Both networks were created to be as similar to the original as possible but also including some image preprocessing. The datasets for each LeNet and VGGNet were not originally CIFAR-10 so some adjustments were made to allow these networks to train on CIFAR-10. LeNet was originally used on the MNIST dataset which had 28x28 images and VGGNet was originally used on the Imagenet dataset which had 224x224 images. CIFAR-10 has 32x32 images. This project used Keras with a Tensorflow backend to create these networks. The images were subjected to a random vertical or horizontal shift or a horizontal flip. Each network was trained for 100 epochs with a batchsize of 50. The images were normalized to 0-1 for easier training.

3. Experiment

This project tested the LeNet architecture and VGGNet architecture using stochastic gradient descent with a learning rate of 0.0005, decay of 0.00005, and nesterov accelerated gradient with a momentum of 0.9. The networks were also trained using adam optimizer instead to compare the accuracy and speed of the different optimizers. Also to test how the different optimizers might lead to different results.

VGGNet and LeNet have very different architectures. VGGNet is much deeper than LeNet and uses a homogenous structure of 3x3 convolutional layers and 2x2 max pooling layers with ReLu activations. LeNet uses two 5x5 convolutional layers and one 1x1 convolutional layer at the end. LeNet also uses two 2x2 max pooling layers but uses the sigmoid activation instead of ReLu. The difference between sigmoid and ReLu is that sigmoid allows wrong predictions to strongly influence the training and sigmoid cannot be used for deep networks because it suffers from vanishing gradient. ReLu does not let wrong predictions influence the training, instead it just ignores mistakes and focuses only on correct predictions. ReLu does not suffer from vanishing gradients because it does not normalize its output to between 0 and 1 like sigmoid, instead it lets the positive output be as high as it originally was.

4. Results

VGGNet with stochastic gradient descent had a training accuracy of 91.89% and a test accuracy of 83.31%. The loss value for training was 0.2338 and the loss value for test was 0.5790. VGGNet with adam optimizer had a training accuracy of 9.90% and a test accuracy of 10.00%. The loss value for its training was 2.3027 and the loss value for its test was 2.3026.

LeNet with stochastic gradient descent had a training accuracy of 31.68% and a test accuracy of 35.68%. The loss value of its training was 1.8673 and the loss value of its test was 1.8024. LeNet with adam optimizer had a training accuracy of 52.10% and test accuracy of 58.30%. The loss value of its training was 1.3392 and the loss value of its test was 1.1836.

Network	Train Accuracy	Test Accuracy	Train Loss	Test Loss
VGGNet w/ SGD	91.89%	83.31%	0.2338	0.5790
VGGNet w/ Adam	9.90%	10.00%	2.3027	2.3026
LeNet w/ SDG	31.68%	35.68%	1.8673	1.8024
LeNet w/ Adam	52.10%	58.30%	1.3392	1.1836

Figure 4. Table of train/test accuracy and loss for VGGNet and LeNet with SGD and Adam optimizer.

Looking at the table, the different optimizers resulted into very different results. This can be attributed to the differences in how stochastic gradient descent and Adam optimizer works. Stochastic gradient descent is slow at learning but Adam optimizer is much faster because it uses momentum. The results of VGGNet uses stochastic gradient descent are decent. The training accuracy is much higher than the test accuracy which means that it was over fitting the training data. At around epoch 60, the network started to over fit to the training, resulting in a high training accuracy but lower test accuracy. The results of VGGNet with Adam optimizer are dismal, the accuracy isn't any better than random guessing. This may be caused by learning rate being too high for Adam optimizer so the network could not find the local minimums effectively.

The results for LeNet with stochastic gradient descent aren't very accurate. This can be explained by slow learning rate of LeNet. From the graph, it seems like 100 epochs was not nearly enough time to train because it starts to increase in accuracy rapidly towards the end. The same can be said for LeNet with Adam optimizer, it also looks like it needed more time to train. With the present data, it look like both LeNet models are overfitting on the training set slightly.

5. Conclusion

VGGNet and LeNet are both convolutional networks but vary differently due to their structures. VGGNet has a homogenous structure of 3x3 convolutional layers and 2x2 max pooling layers and it is 16 layers deep. LeNet only has 5 layers and uses 5x5 convolutional layers and a 1x1 convolutional layer with 2x2 max pooling layers. VGGNet uses ReLu activation

but LeNet uses sigmoid because of the differenet depths of the networks. In terms of efficiency, VGGNet is very expensive. VGGNet uses massive amounts of memory for its 16 layers and also takes an enormous amount of time to train. LeNet is much more memory efficient and much faster to train because it is a much smaller network. In this project, VGGNet with stochastic gradient descent achieved the best results but this is because adam optimizer needed a different learning rate to work correctly and LeNet needed more time iterations to train.

If I were to redo this experiemnt, I would try to tune the learning rate for adam optimizer and give each model much more time to train. 100 epochs was not enough for the models to reach their best minimums.

6. Graphs of the Results

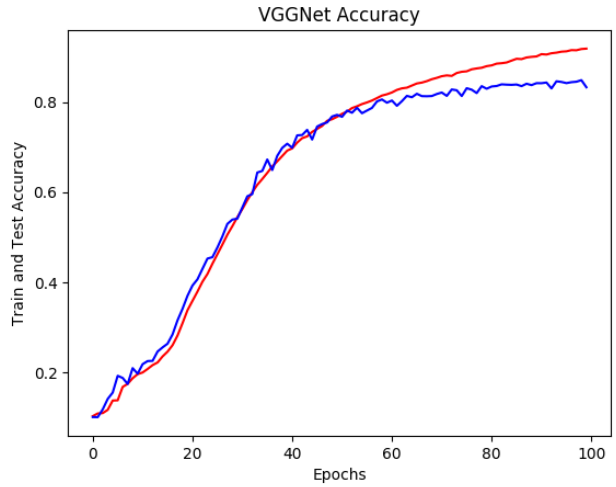

Figure 5. Accuracy of VGGNet with SGD optimizer. Red is training and Blue is test.

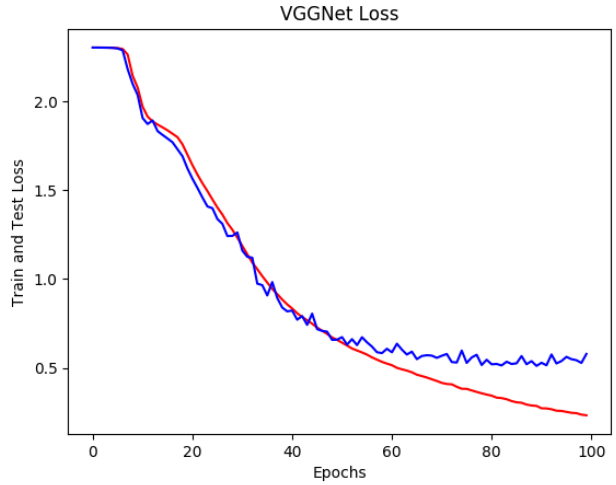

Figure 6. Loss Values of VGGNet with SGD optimizer. Red is training and Blue is test.

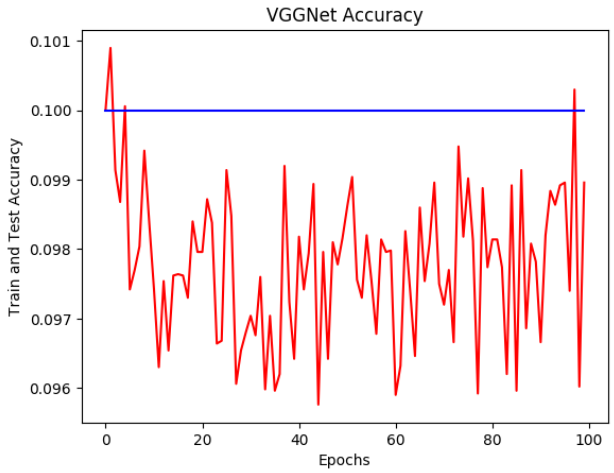

Figure 7. Accuracy of VGGNet with Adam optimizer. Red is training and Blue is test.

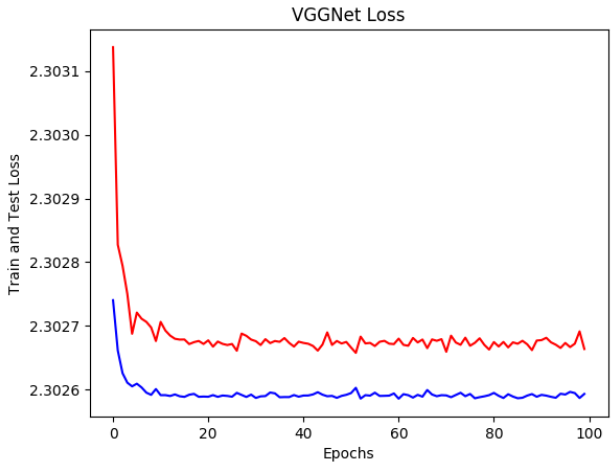

Figure 8. Loss Values of VGGNet with Adam optimizer. Red is training and Blue is test.

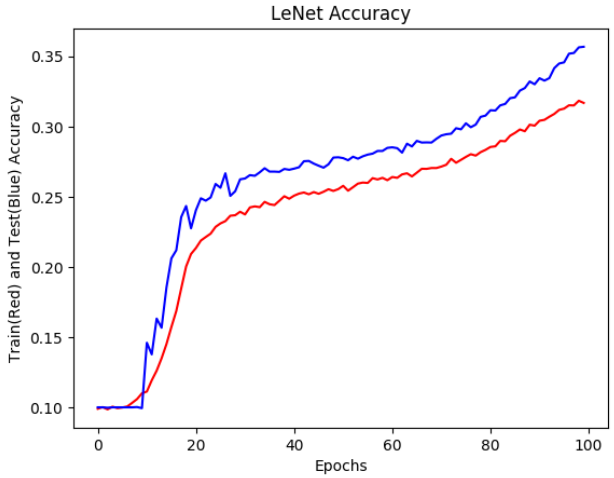

Figure 9. Accuracy of LeNet with SGD optimizer. Red is training and Blue is test.

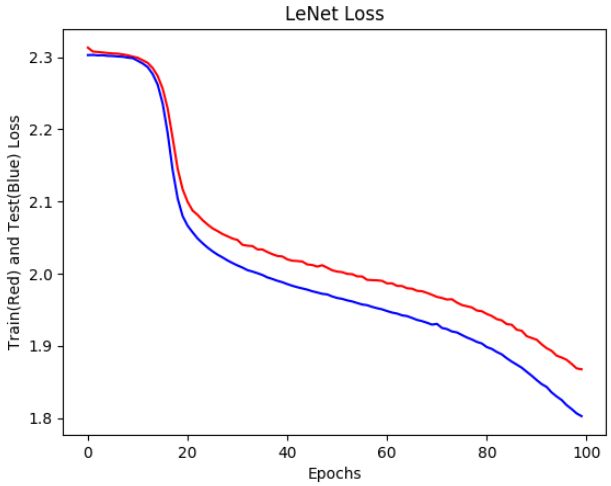

Figure 10. Loss Values of LeNet with SGD optimizer. Red is training and Blue is test.

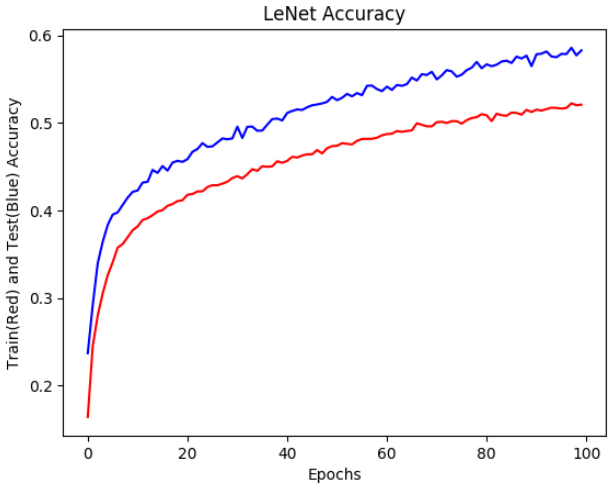

Figure 11. Accuracy of LeNet with Adam optimizer. Red is training and Blue is test.

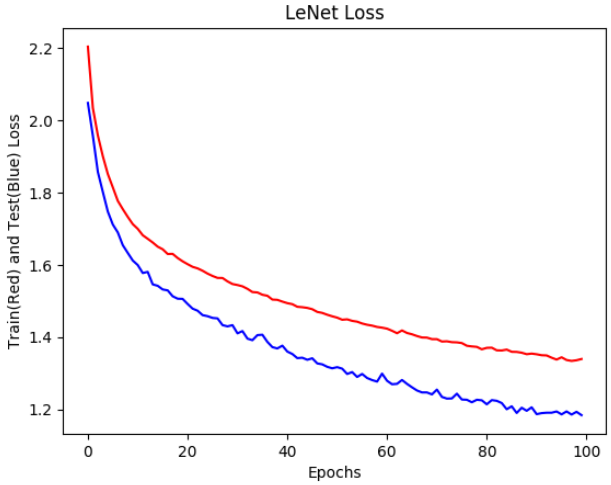

Figure 11. Loss Values of LeNet with Adam optimizer. Red is training and Blue is test.

7. Referernces

[1] Schmidhuber, JÃ¼rgen. "Deep Learning in Neural Networks: An Overview. *Neural Networks* 61: 85-117. January 2015.

[2] S. Karen, Z. Andrew: Very Deep Convolutional Networks For Large-Scale Image Recognition, *Computer Vision and Pattern Recognition*: arXiv:1409.1556, September 2014

[3] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: Gradient-Based Learning Applied to Document Recognition, *Proceedings of the IEEE*, 86(11):2278-2324, November 1998