Sample Neural Network Using Batch Normalization

James Boyer

October 2019

Abstract

This is a LATEX document for the Math DRP program summarizing various experiments for different models neural networks applied to the MNIST dataset.

Keywords: Machine Learning, Gaussian Kernel

1 Introduction

As part of the Math Directed Research Program, what follows is a learning project in implementing a neural network using the MNIST dataset and completing basic experiment comparing different layers and the usefulness of convolution with a Gaussian Kernel.

2 Model Layers and Features

This model follows a basic convolutional network using the following layers and techniques.

2.1 Convolutional2D Layer

The Convolutional2D Layer applies learnable kernels or filters that are convoluted across the input image.

2.2 MaxPooling2D Layer

MaxPooling2D is a simple layer halves the size of the input by sliding a 2x2 window across the 2D input space and takes the maximum value within the window.

2.3 BatchNormalization Layer

BatchNormalization is a technique that calculates the mean and variance of mini-batches in order to learn how to normalize the data. The goal is provide the next layer of the model a consistent distribution to learn on.[3]

2.4 Dense Layer with Relu Activation Function

The Dense Layer is a matrix vector multiplication that takes in all inputs and figuratively connects every input to every node for a given set of nodes. This specific type of layer also utilizes the simple Relu Activation function. [1]

$$f(x) = \max(0, x)$$

2.5 Dropout Layer

The Dropout Layer randomly ignores inputs between layers in order to prevent the model from overfitting. [2]

2.6 Dense Layer with Softmax Activation Function

The Softmax Activation Function is a smooth approximation to the arg max function by taking the average of the exponentials of the input.

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

2.7 Prepossessing the Data With Gaussian Kernel

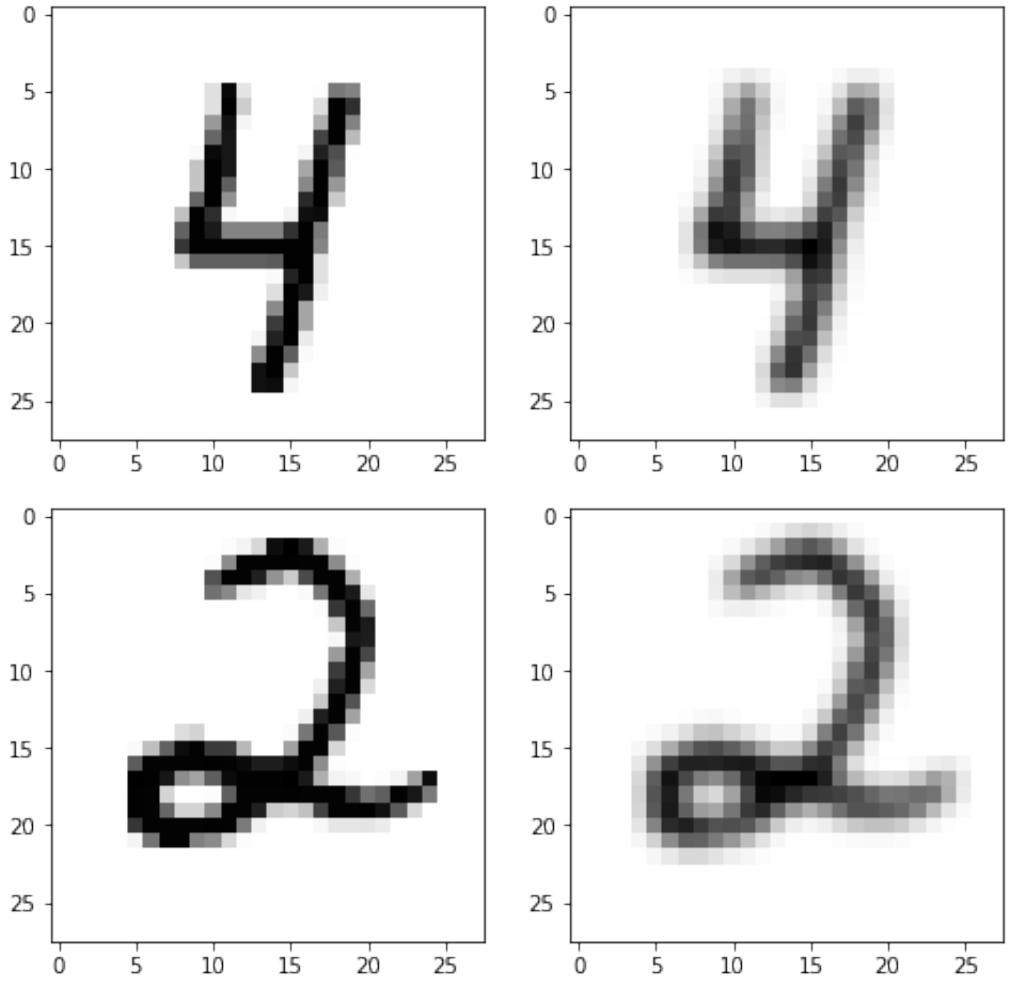
The Gaussian Kernel is a 3x3 filter that when applied looks like a blur effect. The filter is created with the following function:

$$G(x, y) = \frac{1}{2\pi\sigma^2}\exp(-\frac{x^2 + y^2}{2\sigma^2}),$$

Which in practice becomes the following matrix:

$$\begin{bmatrix} 0.05854983 & 0.09653235 & 0.05854983 \\ 0.09653235 & 0.15915494 & 0.09653235 \\ 0.05854983 & 0.09653235 & 0.05854983 \end{bmatrix}$$

Examples of Images Convoluted With Gaussian Kernel



3 Compiling the Model

3.1 Adam Optimizer

The Adam Optimizer is a function that uses Stochastic Gradient Descent with a changing learning rate. This learning rate factors in not only the average of the recent gradients but also the variance [4]

3.2 Sparse Categorical Cross Entropy Loss Function

Sparse Categorical Cross Entropy is a loss function that compares predicted and actual values for each number or category and returns a value indicating if there is a dissimilarity. Specifically, for discrete probability distributions p and q with the same support χ, the Cross Entropy function for each x in the support is:

$$H(p,q) = -\sum_{x \in \chi} p(x) log(q(x))$$

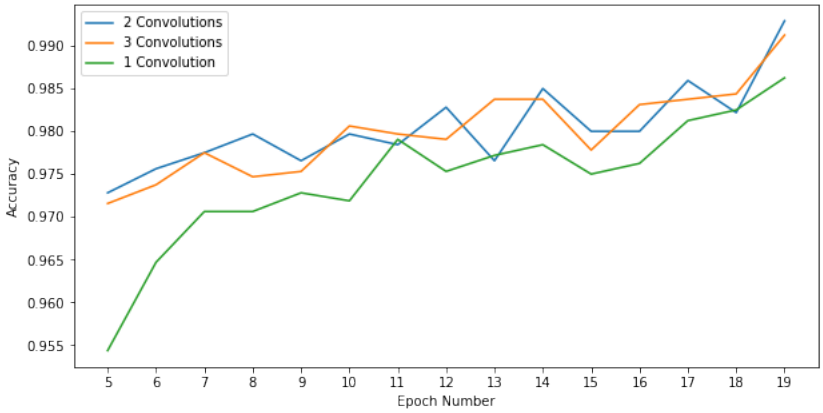
4 Model Experiments

4.1 Number of Convolutions

The first step was to determine the ideal number of convolutions. Each convolution tested is a combination of a 2D Convolutional layer followed by a 2D Max Pooling layer. I tested models with 1, 2, or 3 convolutions with an increasing number of filters in each 2D Convolutional layer.

The results indicate that the models with 2 or 3 convolutions trained faster and performed better than the model with 1 convolution. Since the results between the models with 2 or 3 convolutions were similar, I chose the one with 2 convolutions for the sake of efficiency.

Results of Training Models with Different Numbers of Convolutions

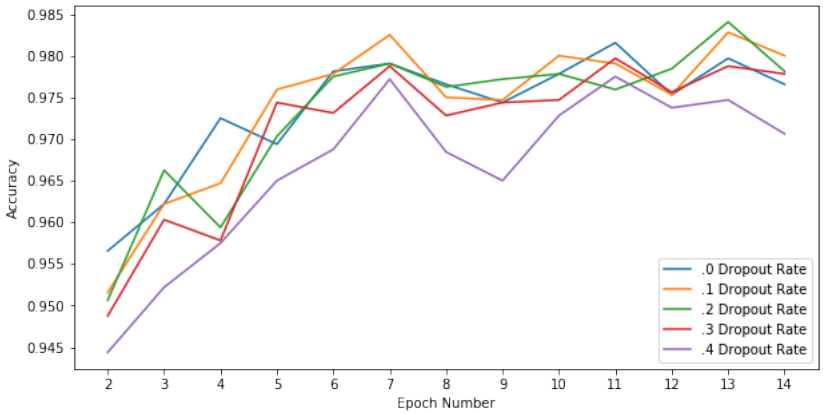


4.2 Dropout Layer Percentage

The next step was to determine the percentage for the dropout layer situated before the final dense layer with 10 nodes. I tested the layer with a range of values from .0 to .4 percent.

I settled on a Dropout Rate of .3 since the only percentage that did substantially worse than the other layers was .4. Further training may make it more clear which percentage is best.

Results of Training Models with Different Dropout Layer Percentages

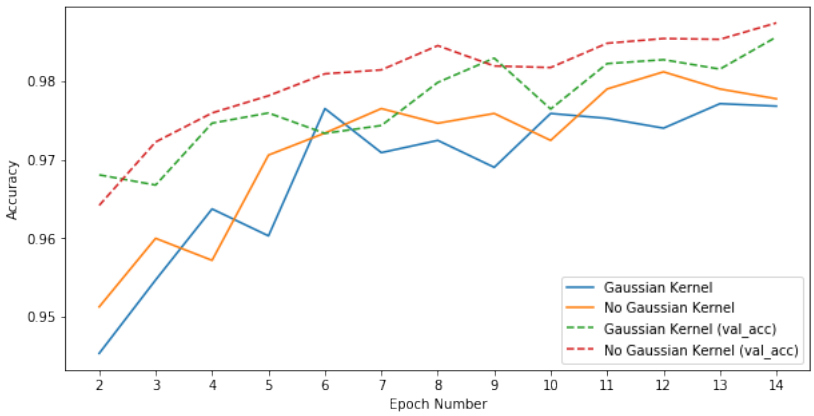


4.3 Gaussian Kernel Train Data

I then trained the model on two different sets of images. The first set was smoothed by a Gaussian Kernel.

The results unfortunately indicate the the blurring effect did not help in the training process at all and the normal images led to a slightly higher accuracy.

Results of using a Gaussian Kernel

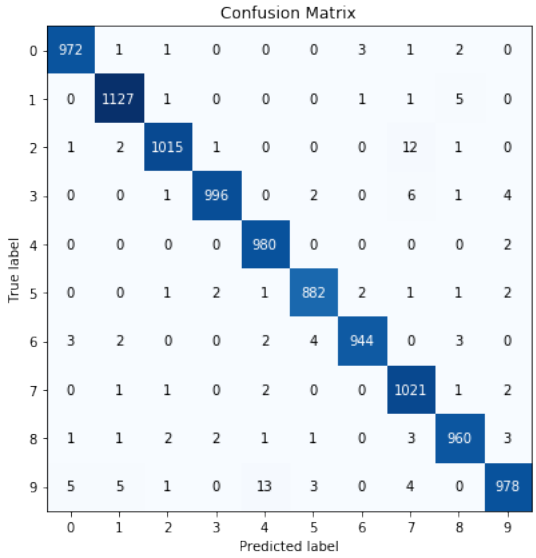


(val_acc) refers to results when the model was tested on the testing data

4.4 Final Model

The final function has an accuracy of 98.75% accuracy when tested across the test data. Below is a Confusion Matrix that highlights the major mistakes made by the model such as mixing up the numbers 2 and 7 or 4 and 9.

Performance of Model



5 Conclusion

This project was a fun experience learning how to use tensorflow to train a neural network. The majority of the work for the project resulted from setting up a system where I could created the training and testing data I need and an easy system to train and evaluate the models. Future developments not implemented here include comparing the benefits of different numbers of filters in the convolution layers and testing the effectiveness of class weights when dealing with an imbalanced dataset.

References

[1] Glorot, X., Bordes, A., and Bengio, Y. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (2011), pp. 315–323.

[2] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580* (2012).

[3] Ioffe, S., and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[4] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).