COMP 576 HW 2

Peter Tang

October 2017

1 Visualizing a CNN with CIFAR10

1.1 CIFAR10 Dataset

1.2 Train LeNet5 on CIFAR10 and Visualizing the Trained Network

1.2.1 trial 1

Training Parameter:

Optimizer	Learning Rate	Batch Size	Max Step	Keep Rate
Adam	0.0001	100	10000	0.8

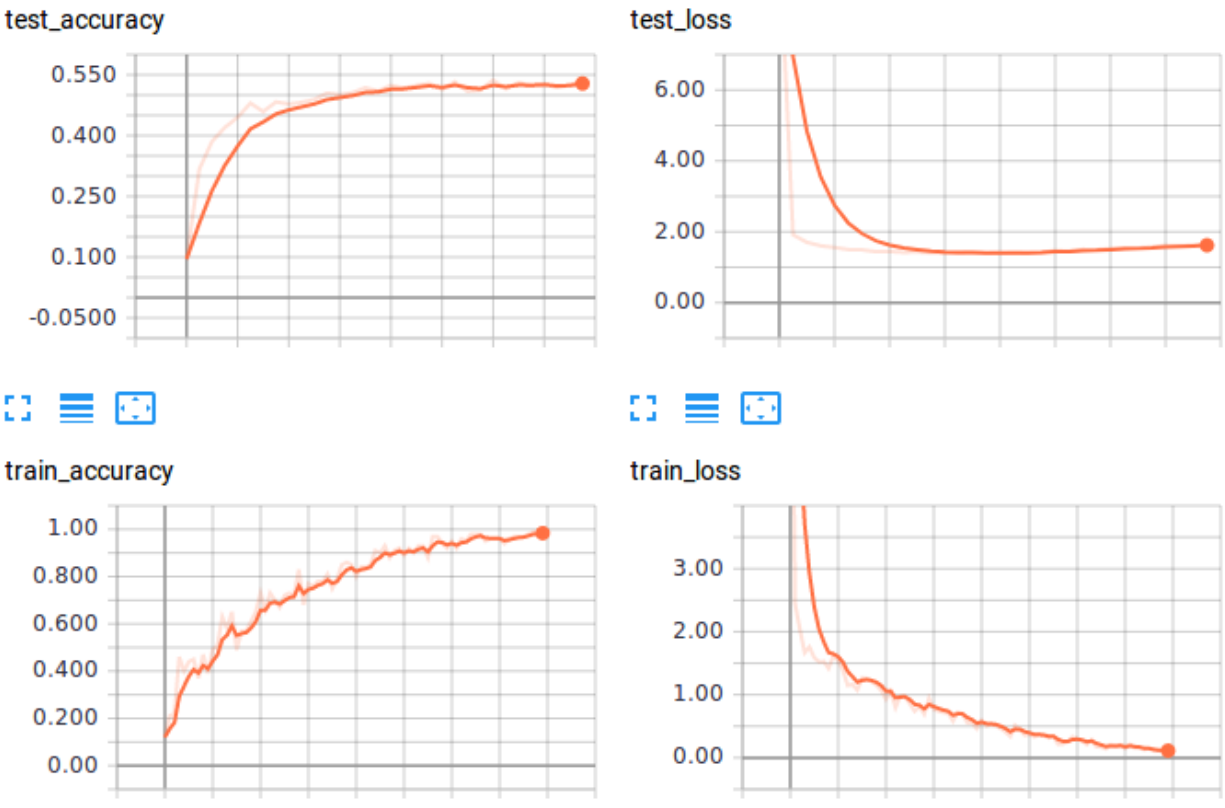

Figure 1: Loss and Accuracy for Training and Testing Set Over Time

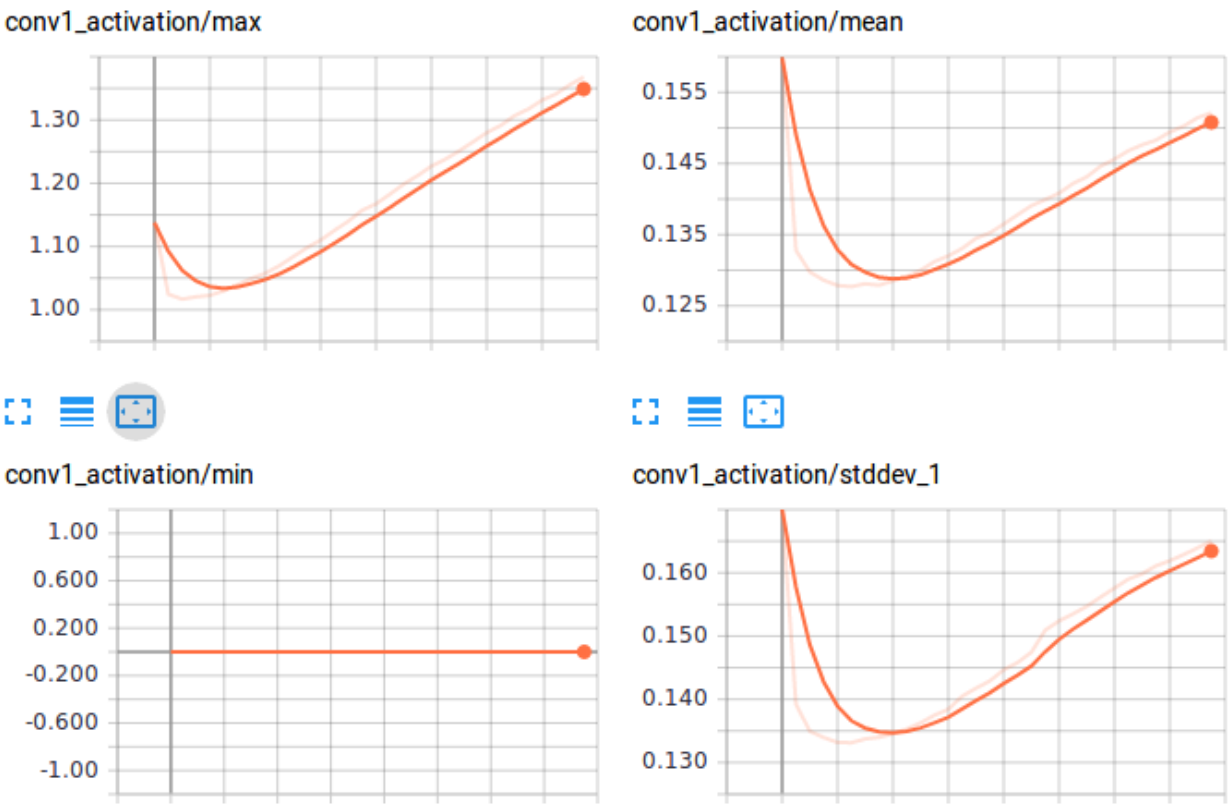

Figure 2: Activation Statistics for Testing Set Over Time

Summary:
this trial uses small learning rate with low drop out percentage. Batch size is as usual. The test loss and accuracy has clearly converged (loss plot looks even like it is about to rise again). More training is likely to lead to over fitting to the training set as the training accuracy looking like it is still going up at the end.

The activation statistics looks normal. Max, mean and standard deviation decrease first then increase. This seems to correspond to the time-point when testing accuracy begins to level off. The increase in amount of neuron firing at more intense rate then correspond to no increase in accuracy. It could be due to it is slowly trying different types of fit for the training set, and eventually preferentially over-fitting the training set. The activation histogram looks like the distribution shifts towards higher value as it trains. So maybe this min point could be a good indicator to stop training the model if there weren't a validation set available?

The filters didn't change much over the training process. Therefore, most of the filter remains to be the randomly generated dots.

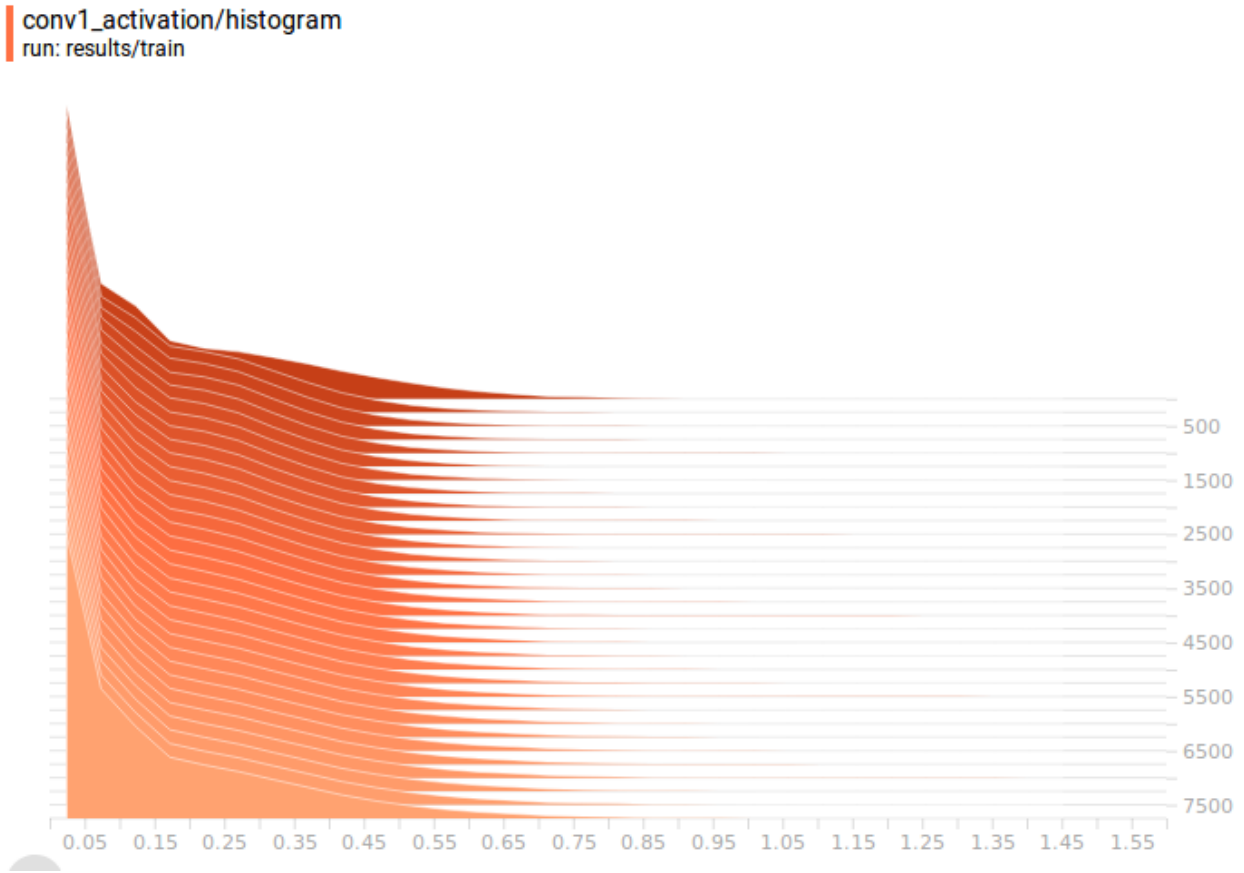

Figure 3: Activation Statistics for Testing Set Over Time

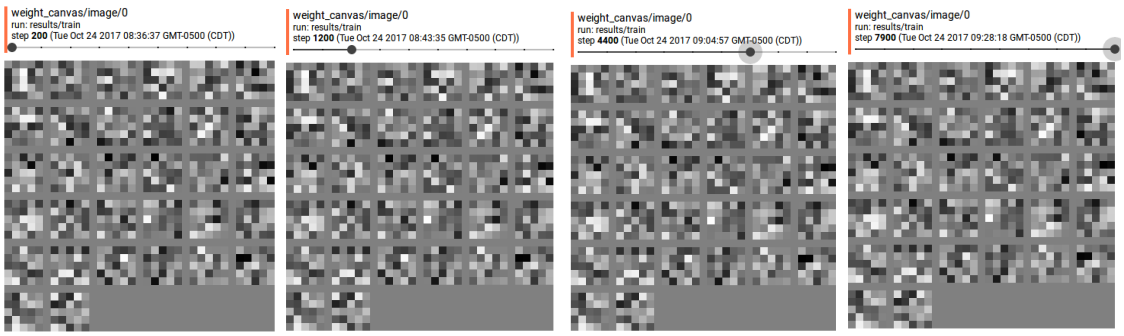

Figure 4: Filter Weights Throughout Training

1.2.2 trial 2

Training Parameter:

Optimizer	Learning Rate	Batch Size	Max Step	Keep Rate
Adam	0.0001	100	10000	0.6

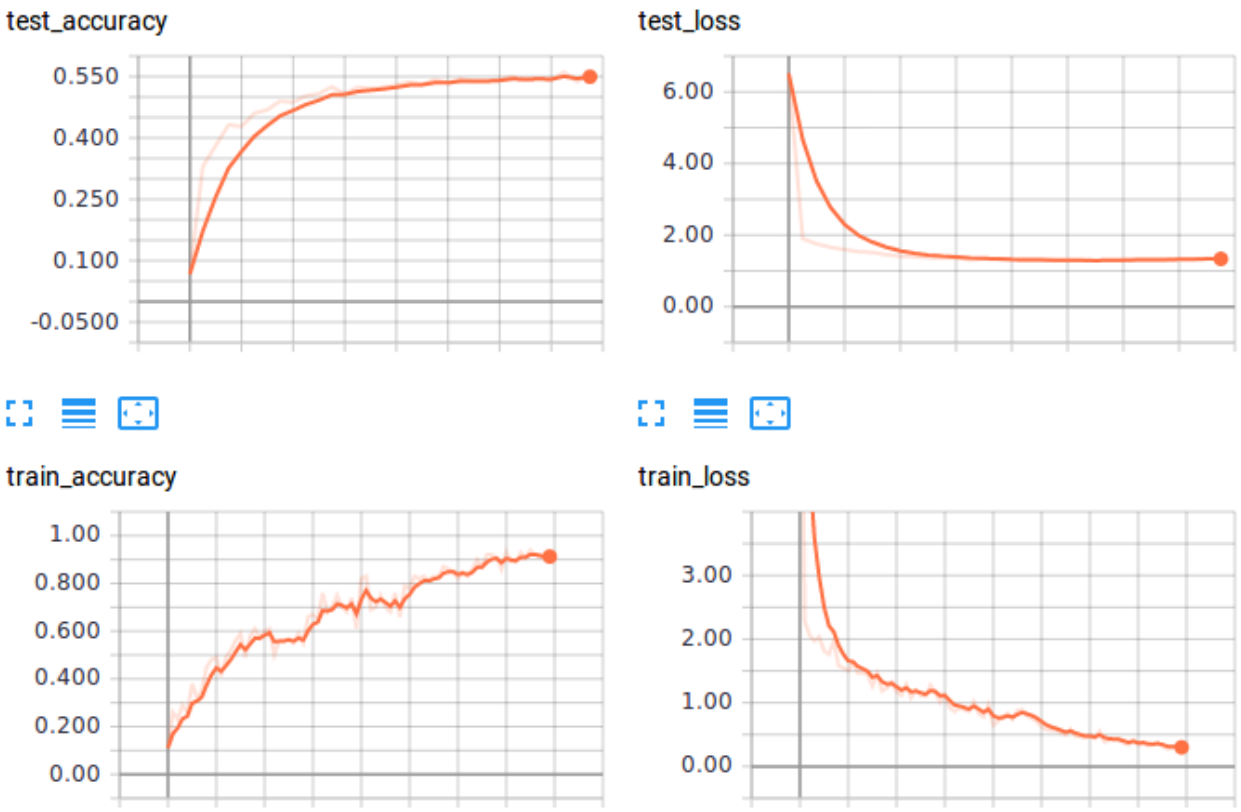

Figure 5: Loss and Accuracy for Training and Testing Set Over Time

Summary
This trial decreases the learning rate from Trial 1 from 0.8 to 0.6. The general shape of loss and accuracy curve is very similar to the previous trail. The training accuracy is lower than previous, but the loss for test-set settles down slightly slower than previous trial, and the accuracy for test set is higher than previous trial, and loss for test set is lower and looks like it has less trend for over-fitting towards the end.

For the activation, it has similar trend as previous trial. However, both max and mean are shifted towards higher value than the last trial. Higher mean and max activation could indicate a better model, because nodes share the classifying power more evenly throughout. Visual inspection of histogram of activation indicates that activation value is shifted even more towards higher value than last trial.

The filters didn't change much over the training process. Therefore, most of the filter remains to be the randomly generated dots.

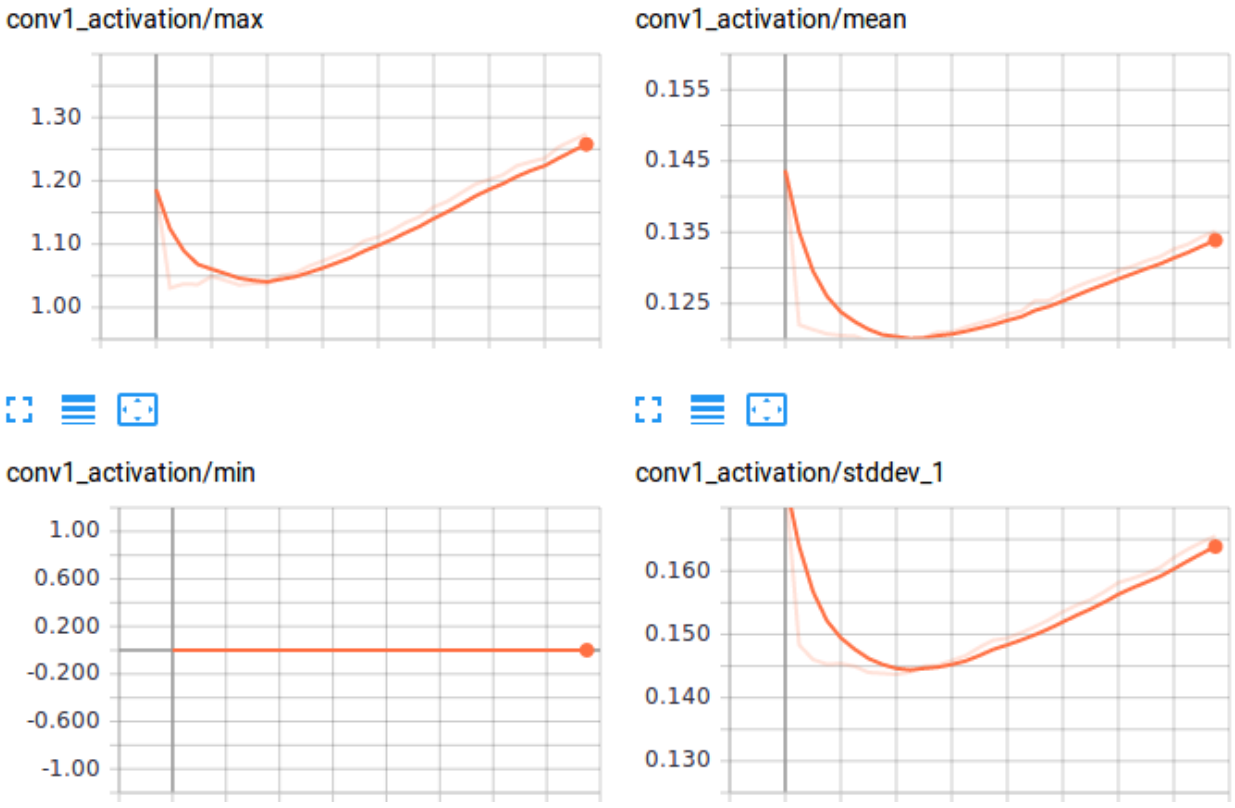

Figure 6: Activation Statistics for Testing Set Over Time

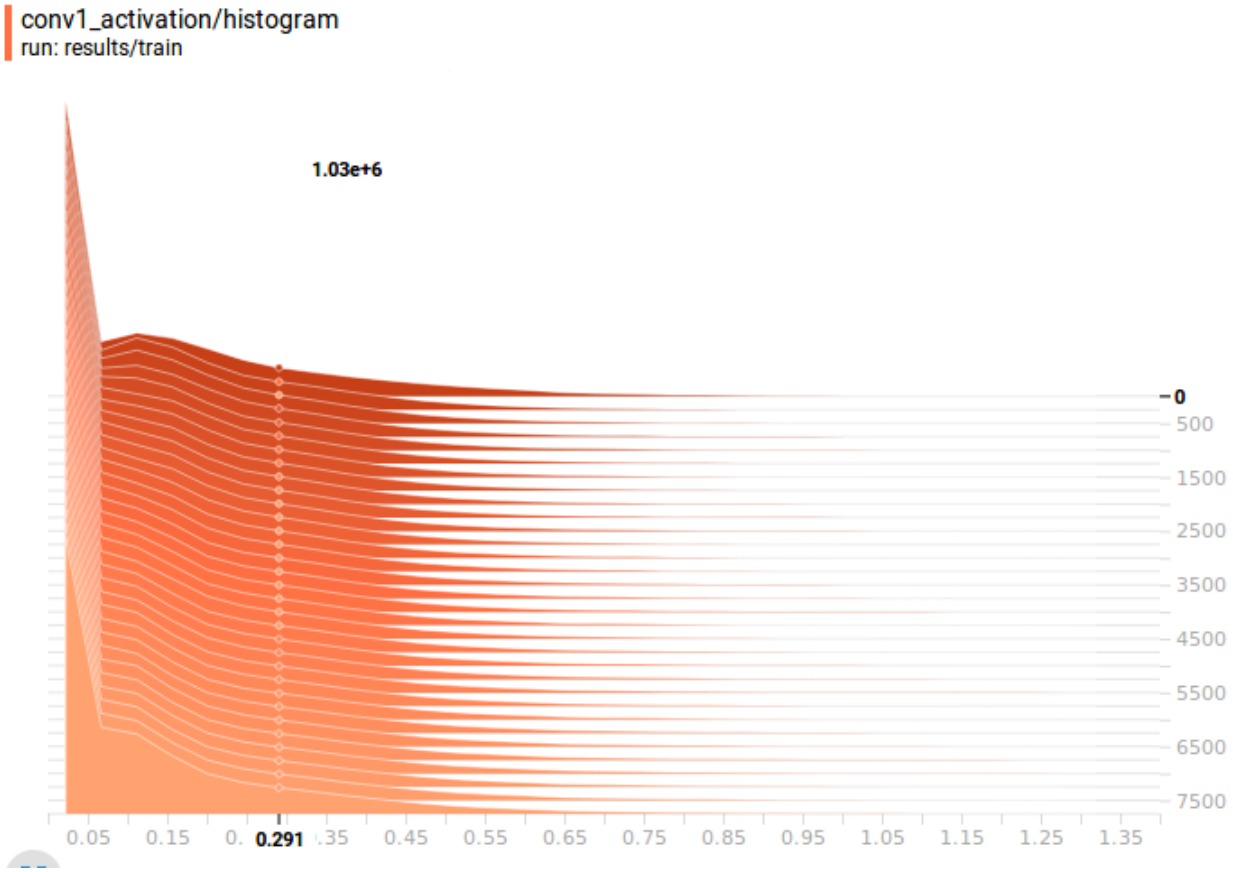

Figure 7: Activation Statistics for Testing Set Over Time

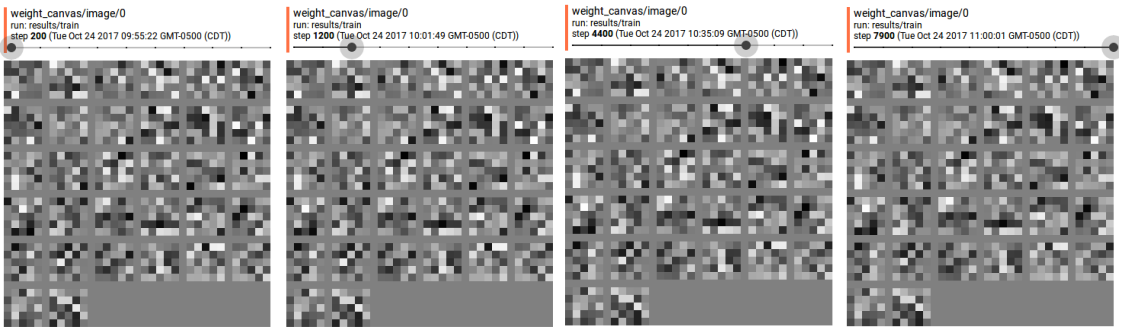

Figure 8: Filter Weights Throughout Training

1.2.3 trial 3

Training Parameter:

Optimizer	Learning Rate	Batch Size	Max Step	Keep Rate
Adam	0.0001	100	8000	0.6

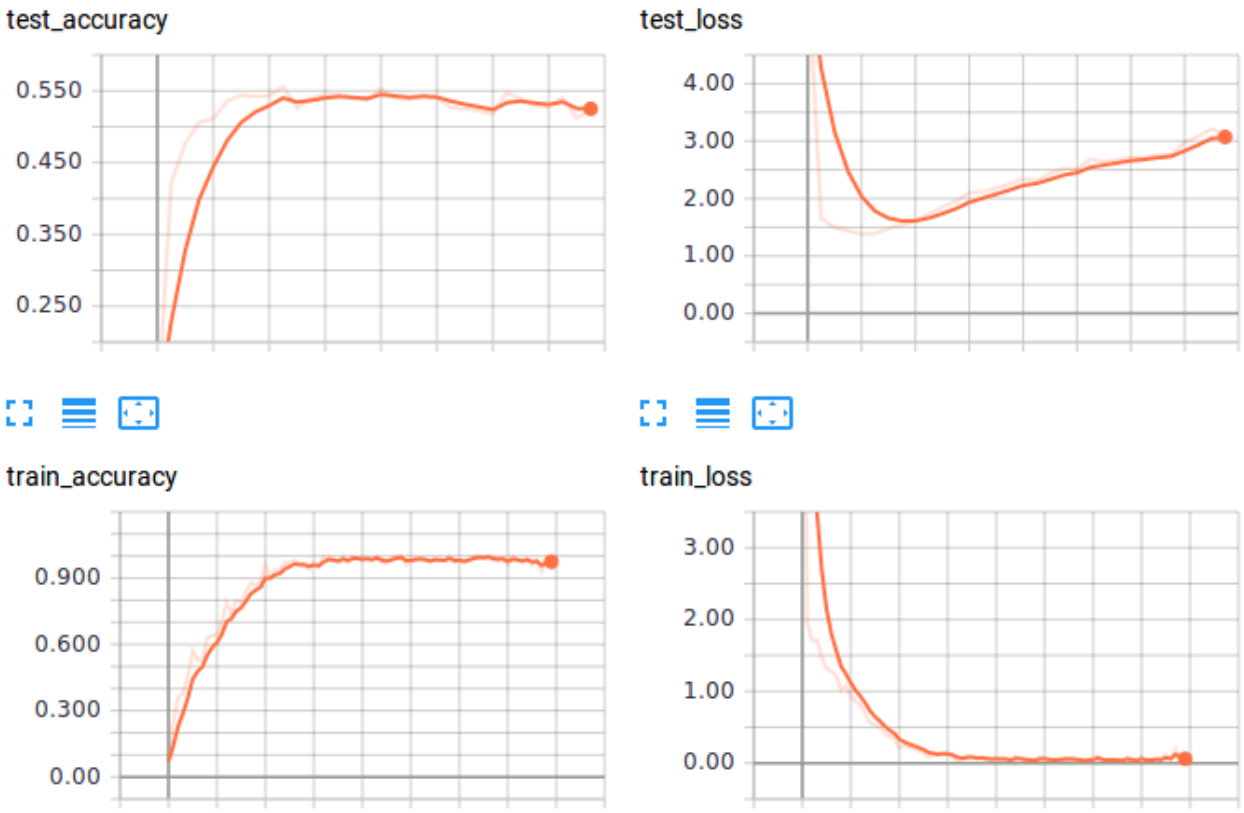

Figure 9: Loss and Accuracy for Training and Testing Set Over Time

Summary:
The loss and accuracy plots looks different this time. The model took longer to plateau on accuracy for training set, and remains at similar accuracy as trial 2. However, the model converged much faster on testing set and reached a higher accuracy than both previous trials. The increase in test set loss after convergence is also lower than both of previous trials.

On the side activation, strangely, the dipping point for max is earlier than the plateau stage of accuracy during training. In fact, the accuracy still increases as max activation increases. Max also increased significantly over the training period. The activation mean, instead of increasing after dipping, remained nearly constant after accuracy plateaus and even decreases after more training. But overall, the values are smaller than previous trial. Less standard deviation than previous trial, and standard deviation increase after dipping is much slower than previous run. This seems to indicate that a good model is ones with activation that has low standard deviation, higher maximum activation for smaller amount of neuron, and average lower activation. Comparing with previous models, the best performance of test sets most likely to occur as a function of (max, min, and mean). The higher max, the lower mean and standard deviation, aka a sparse matrix, has better performance.

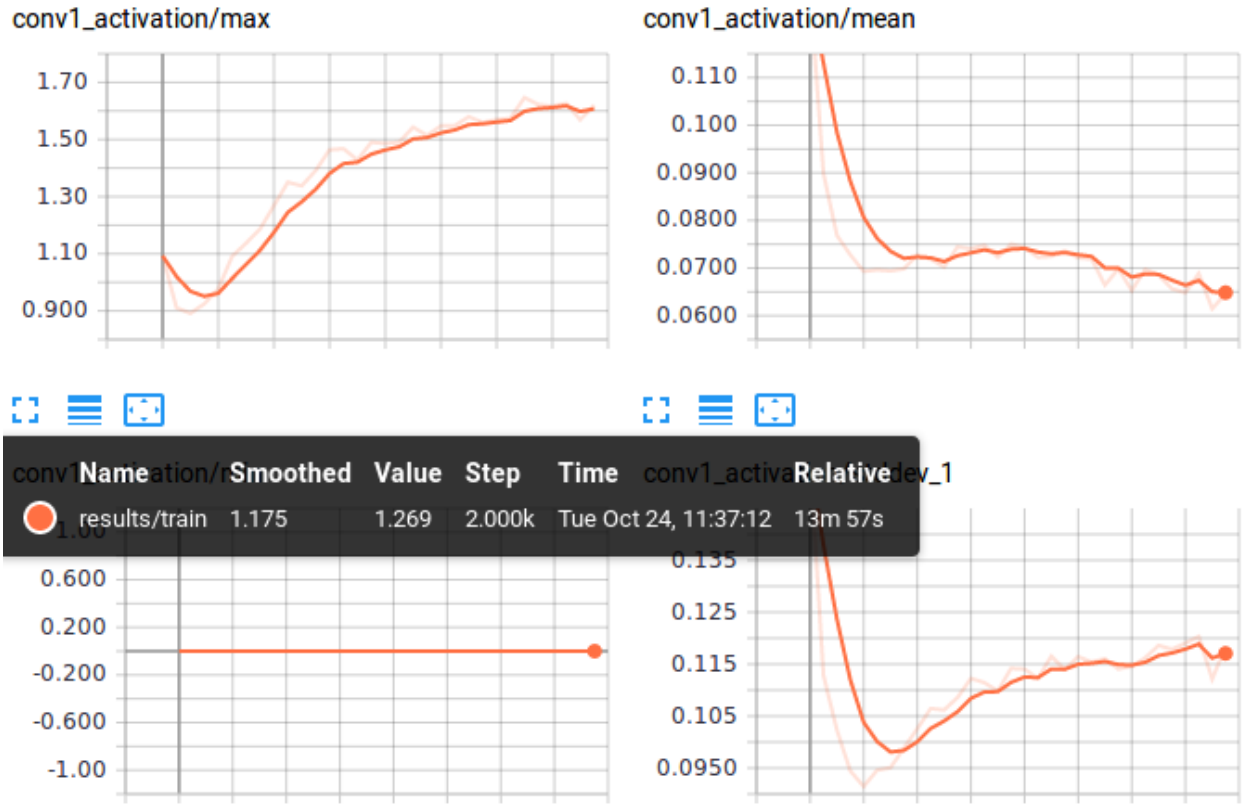

Figure 10: Activation Statistics for Testing Set Over Time

This time the weights are finally changing! the filters that have initially been assigned to a pattern similar to edge or corner resembles more like an edge or corner towards the end, and those that don't blurs out towards a 0 after training. HUMMM, is there a way to intelligently design these filters in the first place so we don't have to have all the other ones that don't look like edge or corner go to waste??

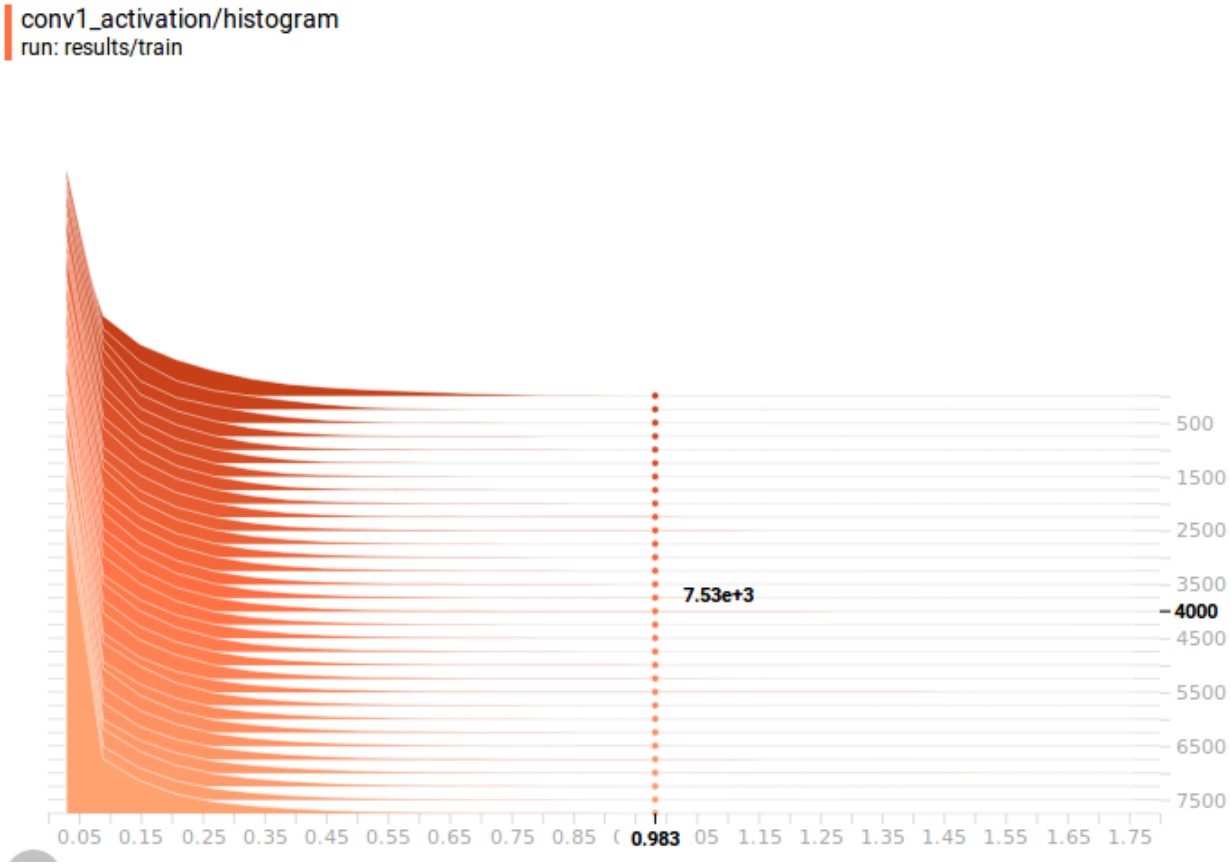

Figure 11: Activation Statistics for Testing Set Over Time

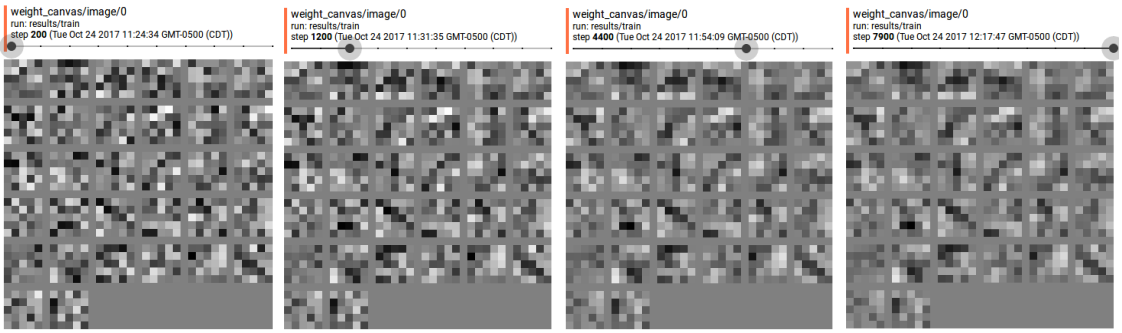

Figure 12: Filter Weights Throughout Training

1.2.4 trial 4

Training Parameter:

Optimizer	Learning Rate	Batch Size	Max Step	Keep Rate
Adam	0.0001	500	8000	0.4

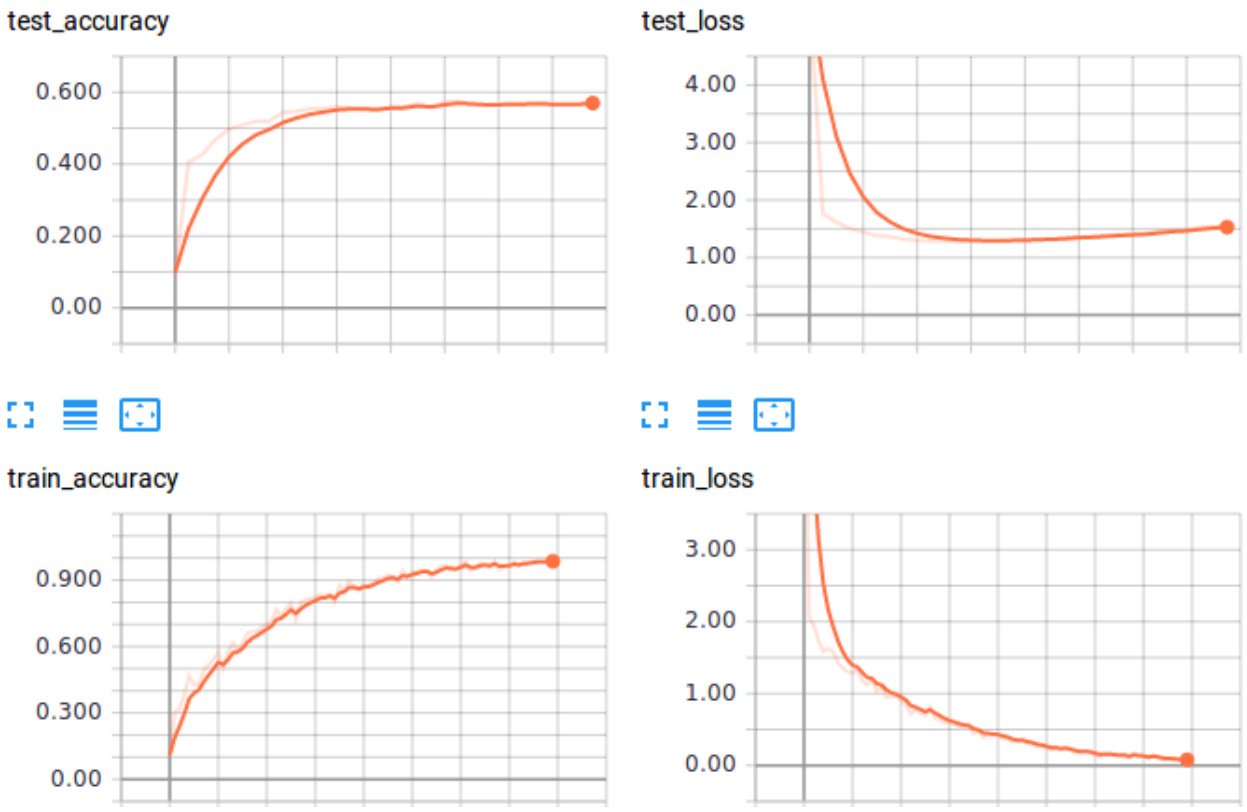

Figure 13: Loss and Accuracy for Training and Testing Set Over Time

Summary
In this trail, I decreased the over all iteration step from 10000 to 8000 just to save time. The keep rate is kept at 0.4 but the batch size is increased from 100 to 500.

The test accuracy converged slowly like trial 3, but reached the highest accuracy out of all trials. The test loss shows a steady drop with almost no increase after the minimum point. The train accuracy and loss converges even more slowly than the test accuracy, and does not seem to be plateauing even though we terminated the training. The curve is also much more smooth than smaller batch, because larger batch size indicating a sample that resembles the population's statistic more.

For the activation statistics, the max is slightly shifted smaller, mean is a steady higher, and standard deviation is lower. All statistics increases as training proceeds. This doesn't really fit the hypothesis that sparse activation is better, maybe because the batch size is different. Another reason this model is superior could be the slop in mean and standard deviation changes direction earliest during training.

This time, the activation plot didn't really changed much

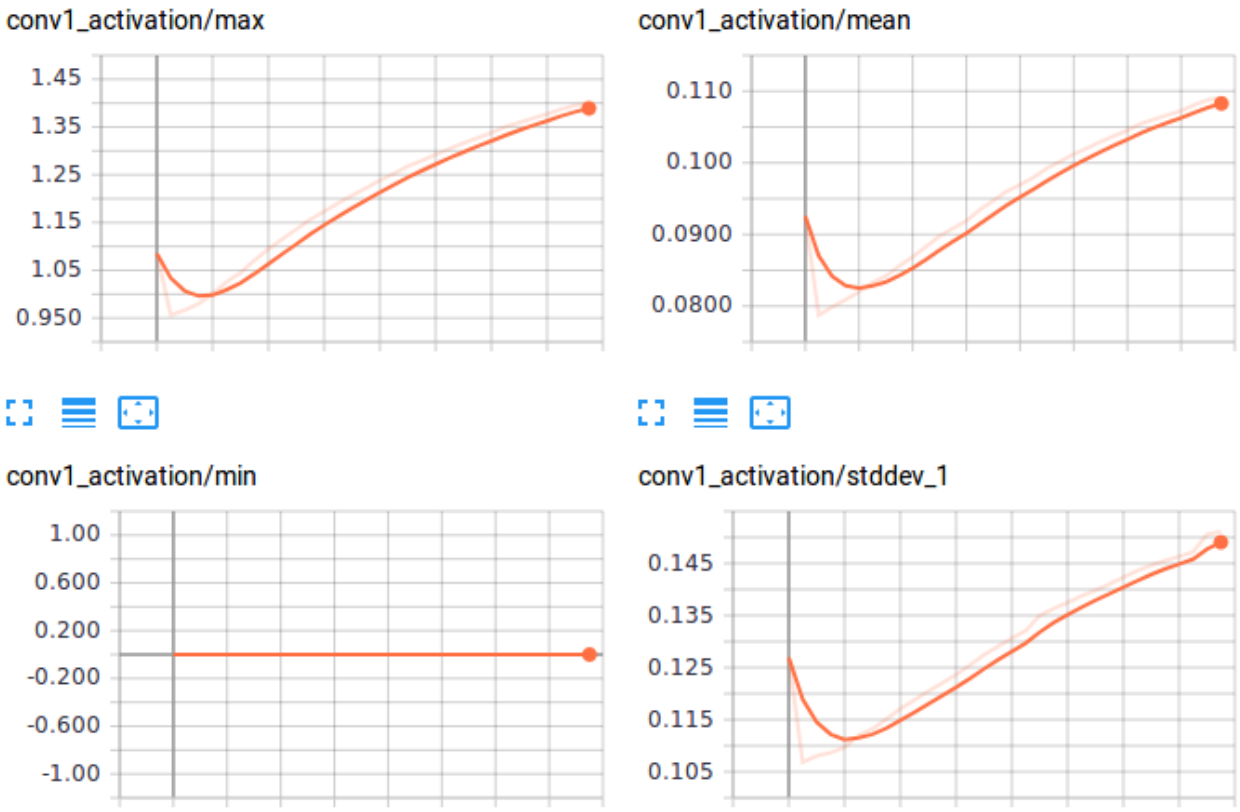

Figure 14: Activation Statistics for Testing Set Over Time

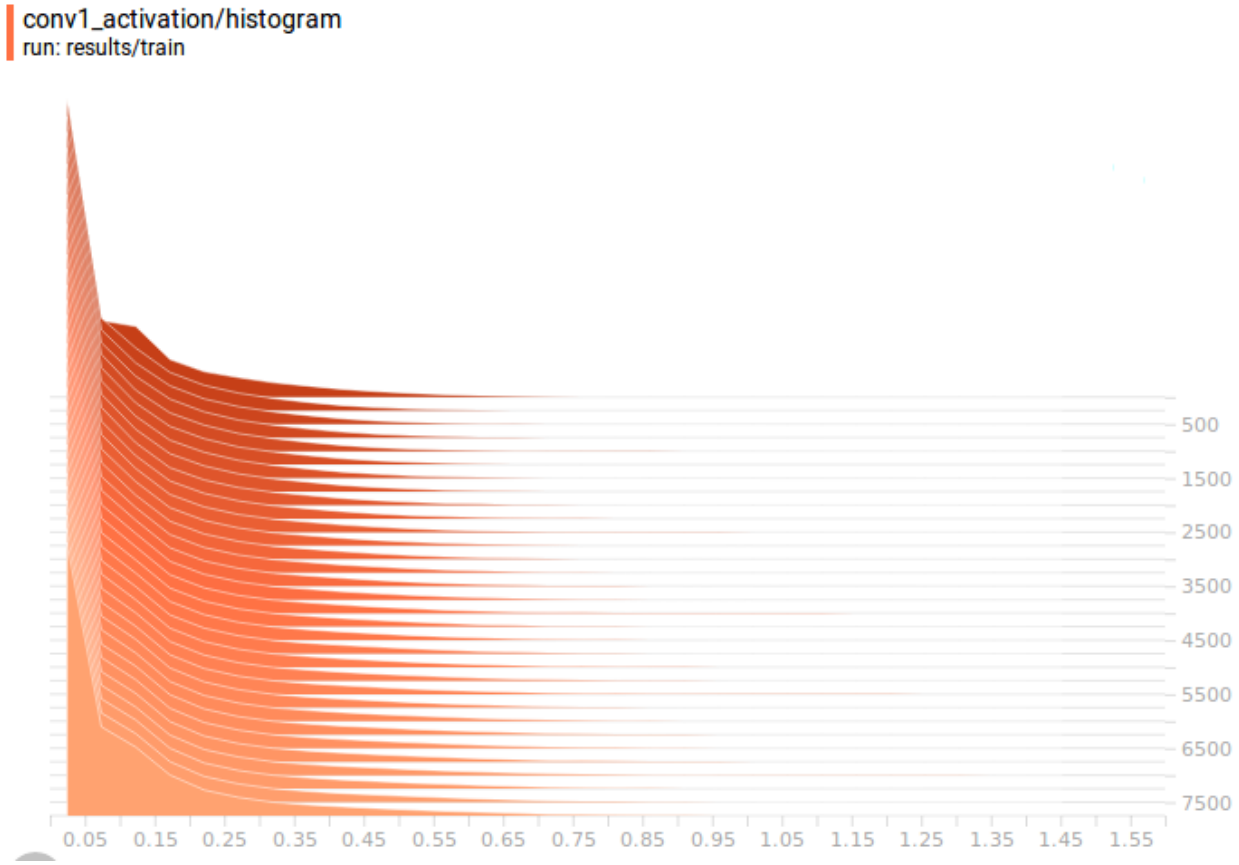

Figure 15: Activation Statistics for Testing Set Over Time

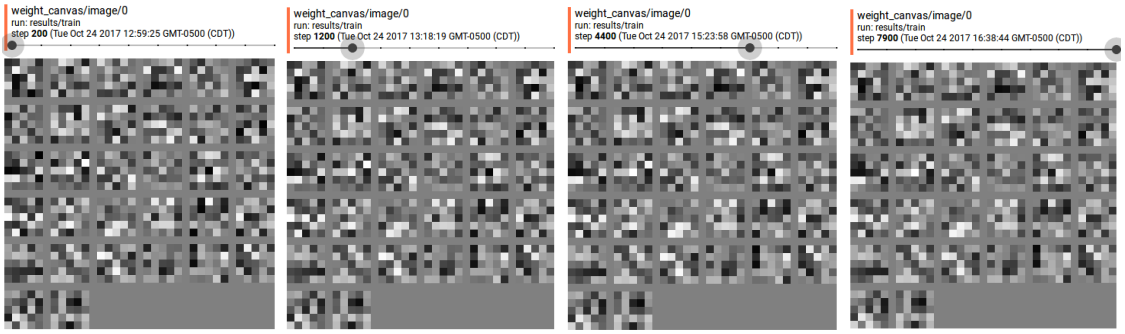

Figure 16: Filter Weights Throughout Training

1.2.5 trial 5

Training Parameter:

Optimizer	Learning Rate	Batch Size	Max Step	Keep Rate
Gradient Descent	0.01	500	8000	0.6

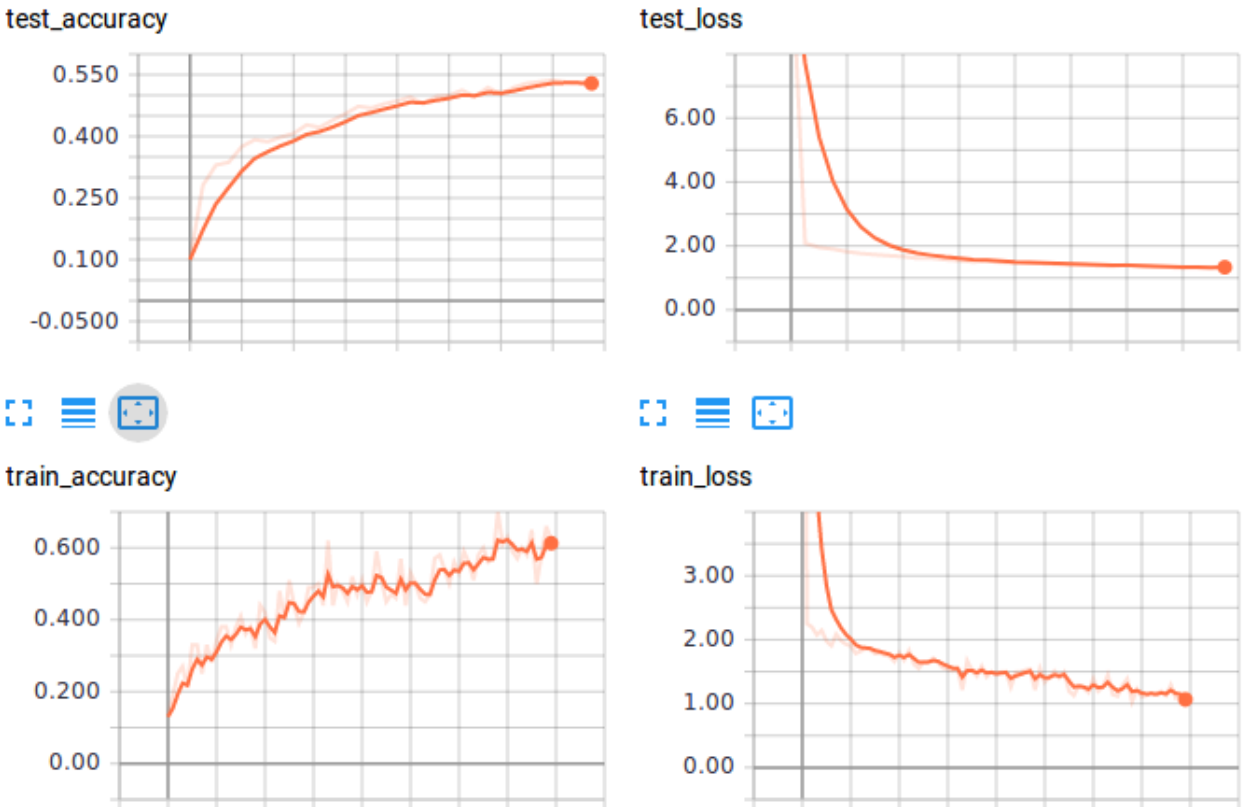

Figure 17: Loss and Accuracy for Training and Testing Set Over Time

Summary
This trial we dialed back the keep rate from 0.4 to 0.6 and used gradient descent with hundred times the previous learning rate, keeping batch size the same

Gradient descent seems much more unstable in improving training accuracy and loss. Ther are constant ups and downs.The test loss, however, never seem to increase after plateauing, and the test accuracy never really plateaued. To achieve a better result, gradient descent needs higher learning rate.

As for activation, the max is much larger than previous trial. The mean and standard deviation are much lower. This is almost a text-book sparse matrix. However, the performance is not as high as expected maybe because the max are not high enough. Because the mean has remained small, the slow increase cannot offset the quick increase in max of activation, so the performance continue to increase.

Most of the filter weights are blurred out. only those that are initially similar to edge and corner get to develop into what we consider an edge or corner. The vertical and horizontal edge is very obvious, these are the max of the activation. For small, grey-scale image set like this, not many features are needed overall, so blurring out most of the other filters is expected because they would just cost more to maintain, might as well get a few filter to become

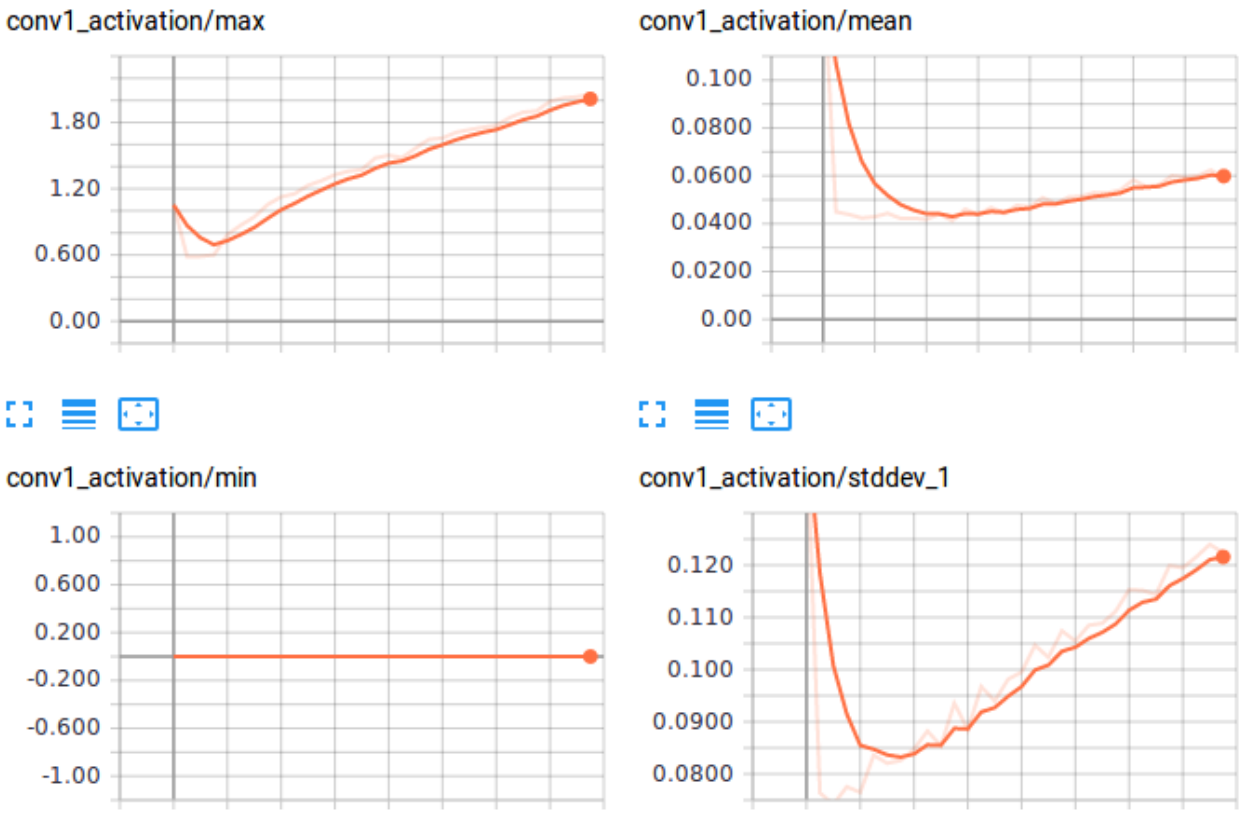

Figure 18: Activation Statistics for Testing Set Over Time

universal edge and look at the same feature over and over.

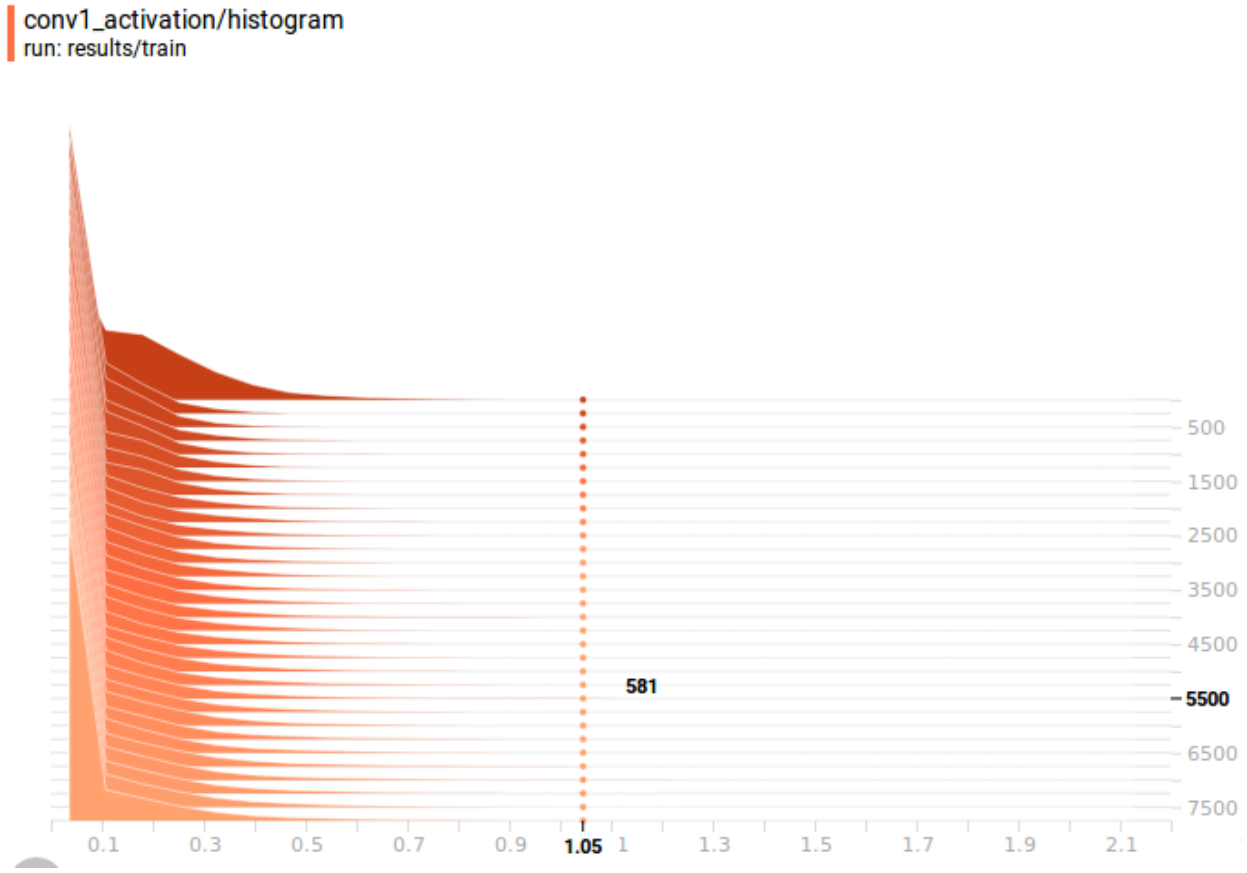

Figure 19: Activation Statistics for Testing Set Over Time

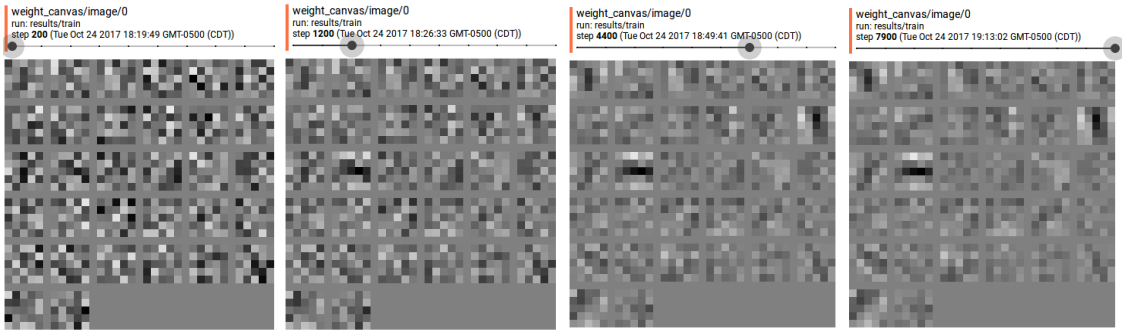

Figure 20: Filter Weights Throughout Training

1.2.6 trial 6

Training Parameter:

Optimizer	Learning Rate	Batch Size	Max Step	Keep Rate
Gradient Descent	0.05	100	8000	0.6

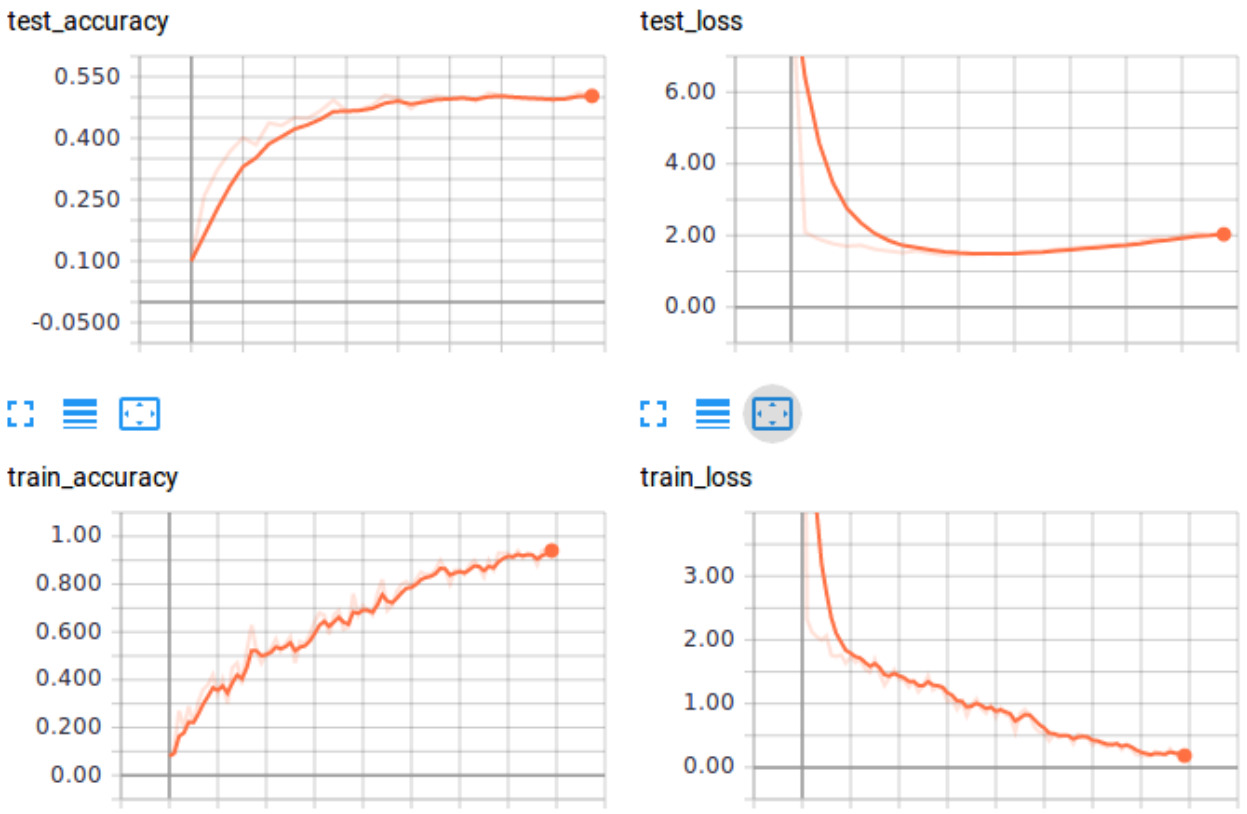

Figure 21: Loss and Accuracy for Training and Testing Set Over Time

Summary
The test accuracy plateaued very slowly, and plateaued at a lower accuracy than previous models.The activation plot looks like a perfect sparse matrix. There are few spots that constantly fire, and most of the other filters went to waste. This is corresponded to the filter weight visualization where most of filters becomes grey and only 4 filters are remotely recognizable as corner. Perhaps in order have a good classifier, we need interdisciplinary work. so we need a bit more feature filters.

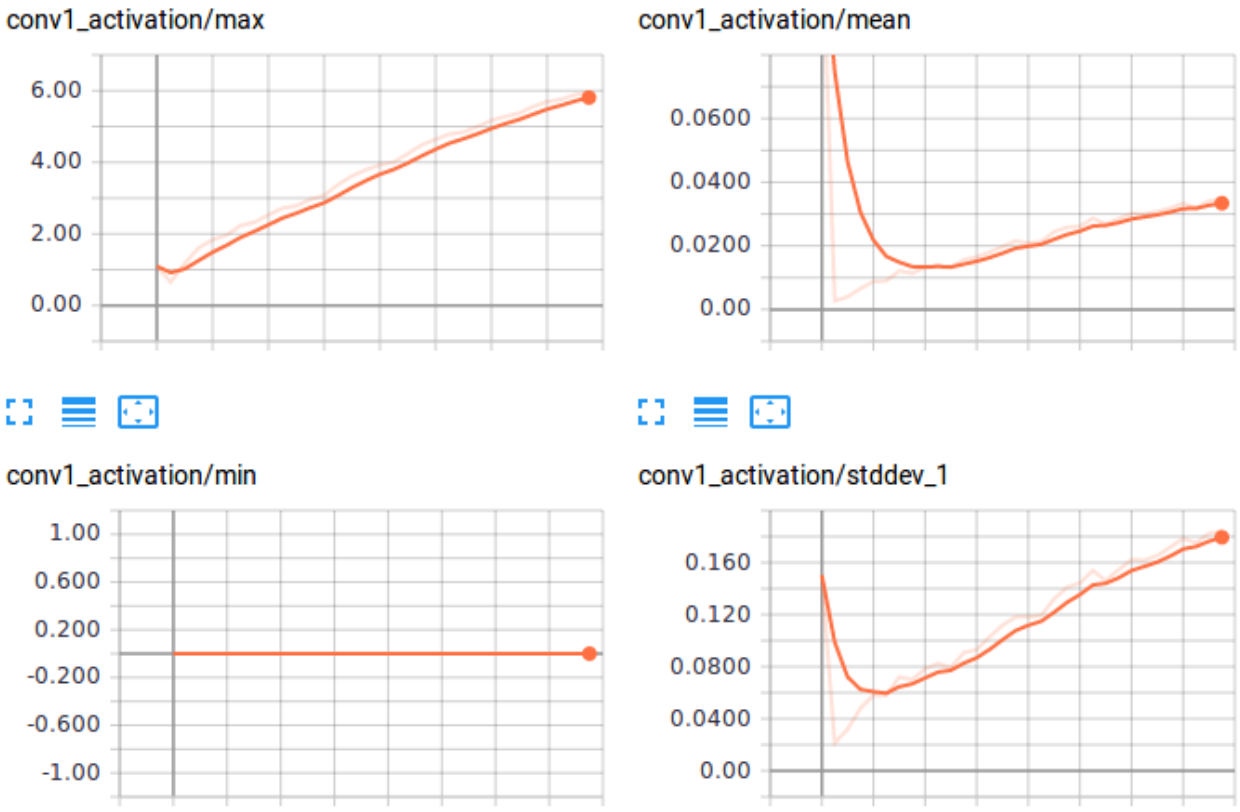

Figure 22: Activation Statistics for Testing Set Over Time

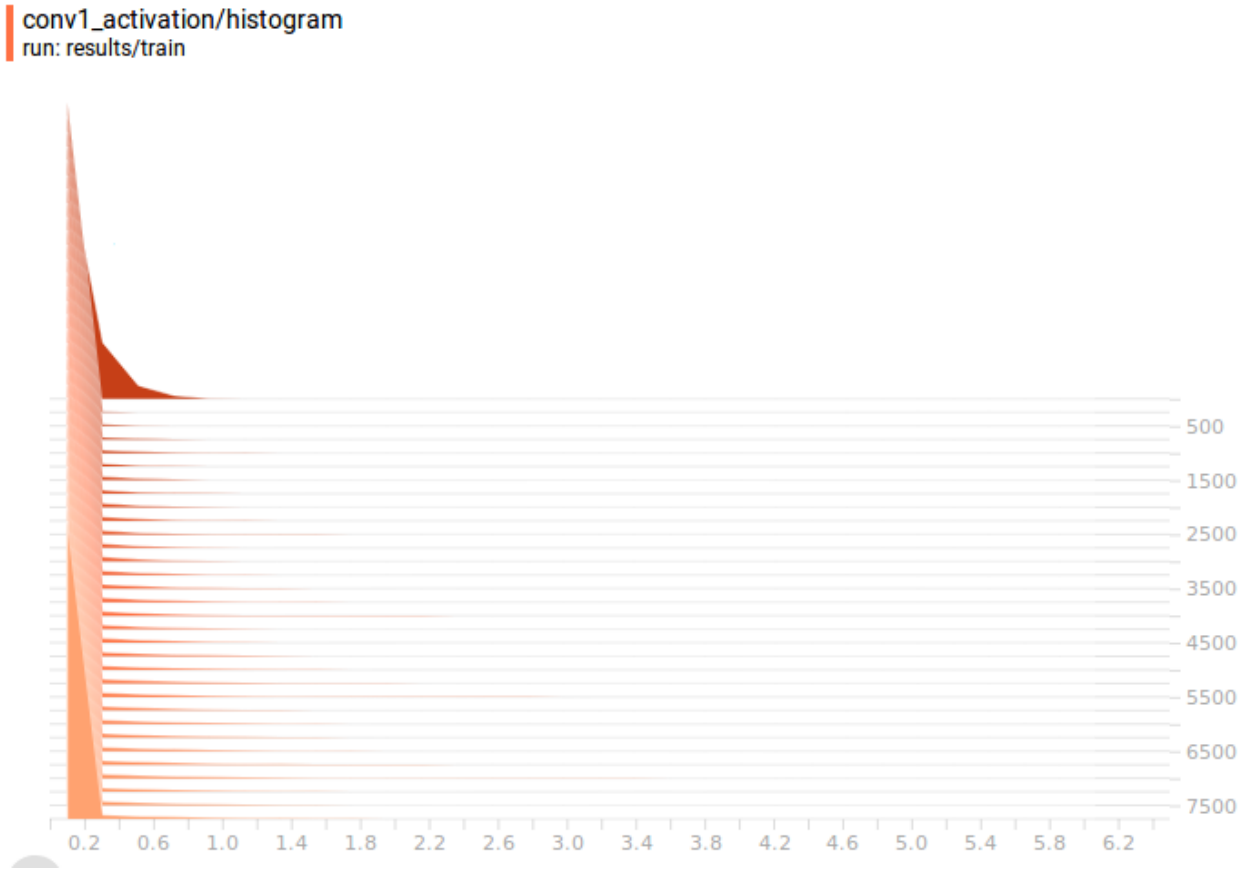

Figure 23: Activation Statistics for Testing Set Over Time

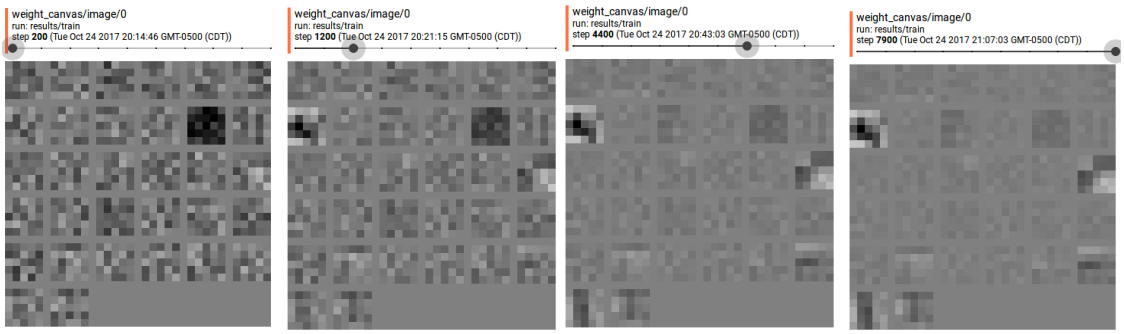

Figure 24: Filter Weights Throughout Training

1.2.7 trial 7

Training Parameter:

Optimizer	Learning Rate	Batch Size	Max Step	Keep Rate
RMS Prop	0.0025	200	5000	0.5

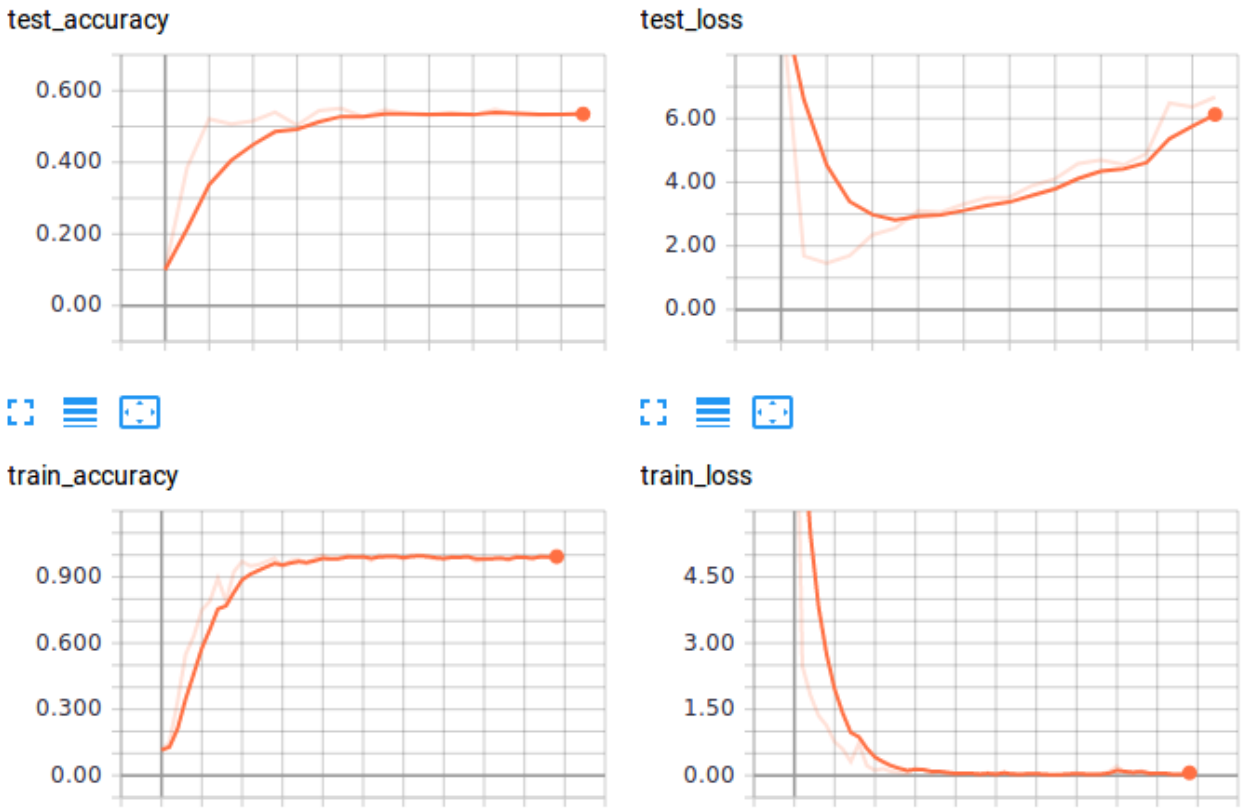

Figure 25: Loss and Accuracy for Training and Testing Set Over Time

Summary:
Standard deviation and mean of activation remains low while max activation increase. When mean and standard deviation reaches bottom, the test accuracy reaches the max value. This confirms the hypothesis that sparse matrix is better. RMS prob is also the only optimization method so far that made the mean and standard deviation of mean decrease over training. The feature looks a lot more like edges or corners after even just 4400 iteration.

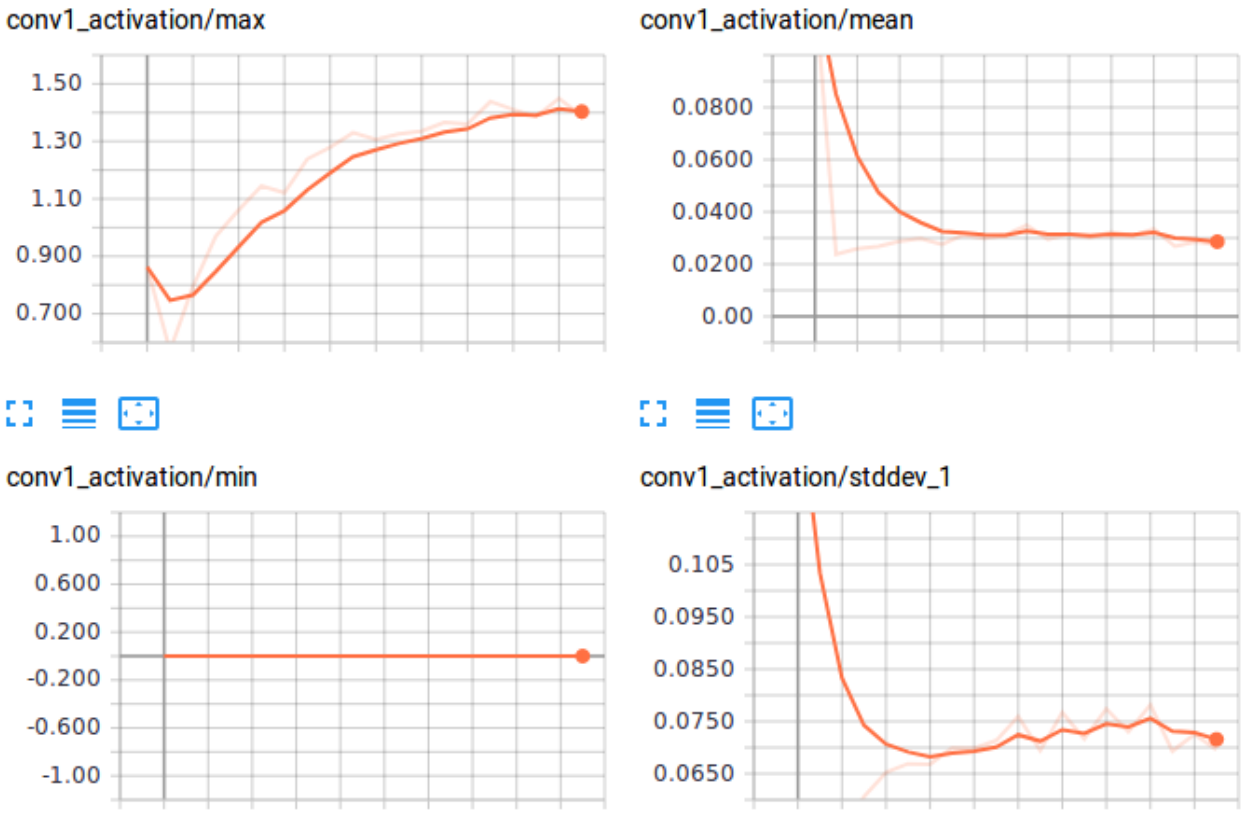

Figure 26: Activation Statistics for Testing Set Over Time

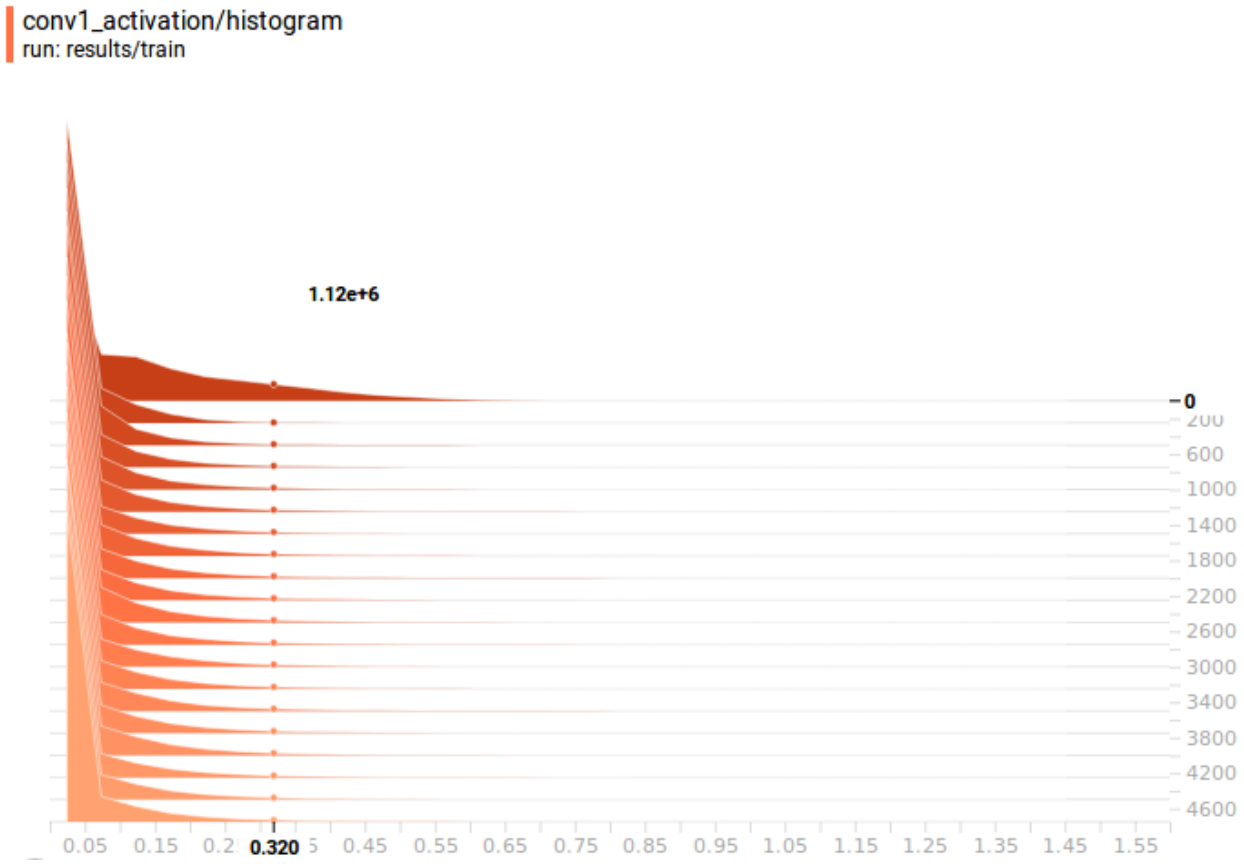

Figure 27: Activation Statistics for Testing Set Over Time

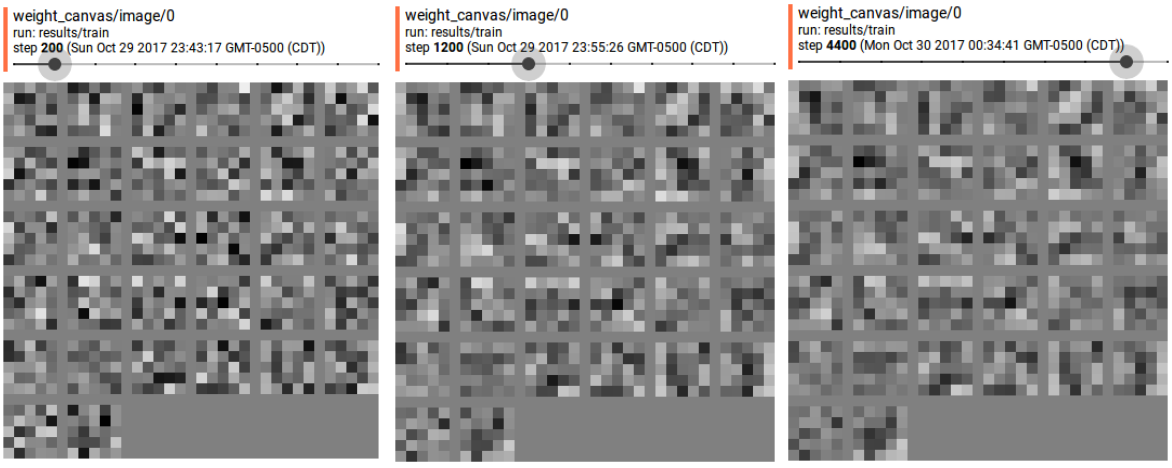

Figure 28: Filter Weights Throughout Training

1.2.8 trial 8

Optimizer	Learning Rate	Batch Size	Max Step	Keep Rate
RMS Prop	0.0005	500	20000	0.4

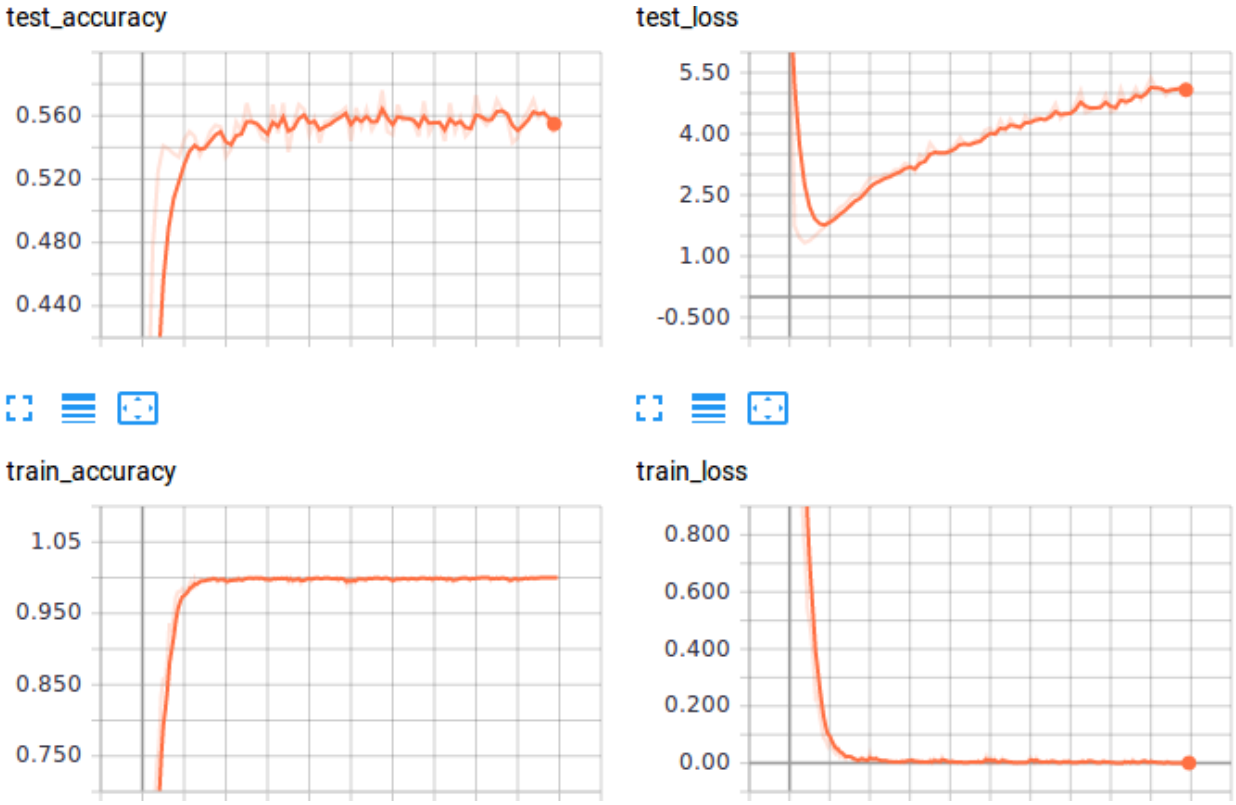

Figure 29: Loss and Accuracy for Training and Testing Set Over Time

Summary:
I increased max step to 4 times of last time, decreased keep rate a bit, increased batch size, and decreased learning rate by 5 fold. With the changes, both mean and standard deviation of activation dipped quickly, while max activation is still on the rise. Correspondingly, the model converged very quickly. RMS prop is also the only algorithm so far that results in eventual decrease in mean and standard deviation of model, which I don't understand. It also seems that the model learned faster than a larger learning rate. The faster convergence could be due to a smaller keep rate.

Conclusion:
1. Keep Rate affects how fast the convergence is. Smaller value increase the rate of convergence for model performance on both train and test set. Even though it may not result in highest training accuracy, the test accuracy will likely be higher
2. Sparse activation provides better performance. In terms of statistics, a smaller set of neural firing at higher rate, with most of neuron firing at low rate, just like division of labor!
3. Not too sparse! At least have a few filter left.
4. Gradient descent needs high learning rate, and converges activation quickly to sparse matrix
5. Performance of model depends on activation. The training usually reaches highest performance when mean and std of activation reaches minimum while max activation is on the rise.
6. RMS prop is the only optimization methods so far that results in eventual decrease in mean and standard deviation

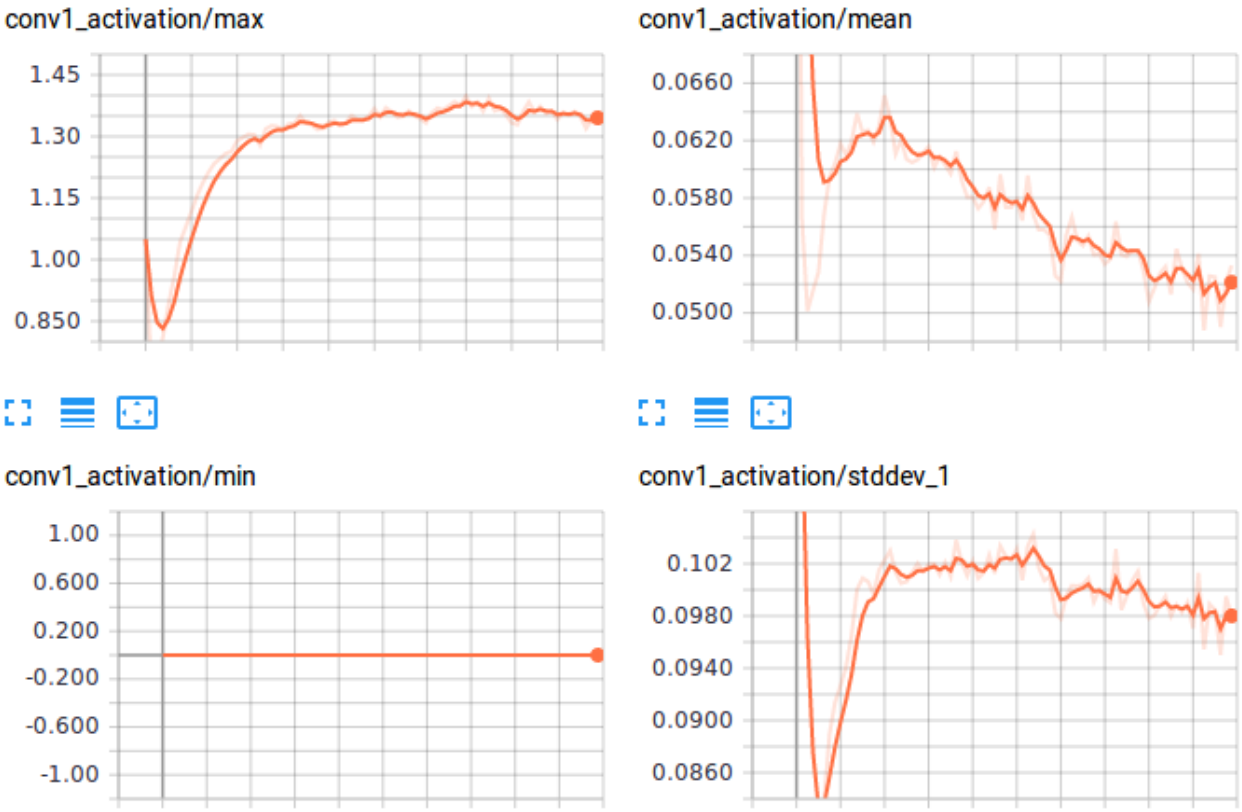

Figure 30: Activation Statistics for Testing Set Over Time

of activation.
7. larger batch size either help with converging faster or converging to a higher accuracy

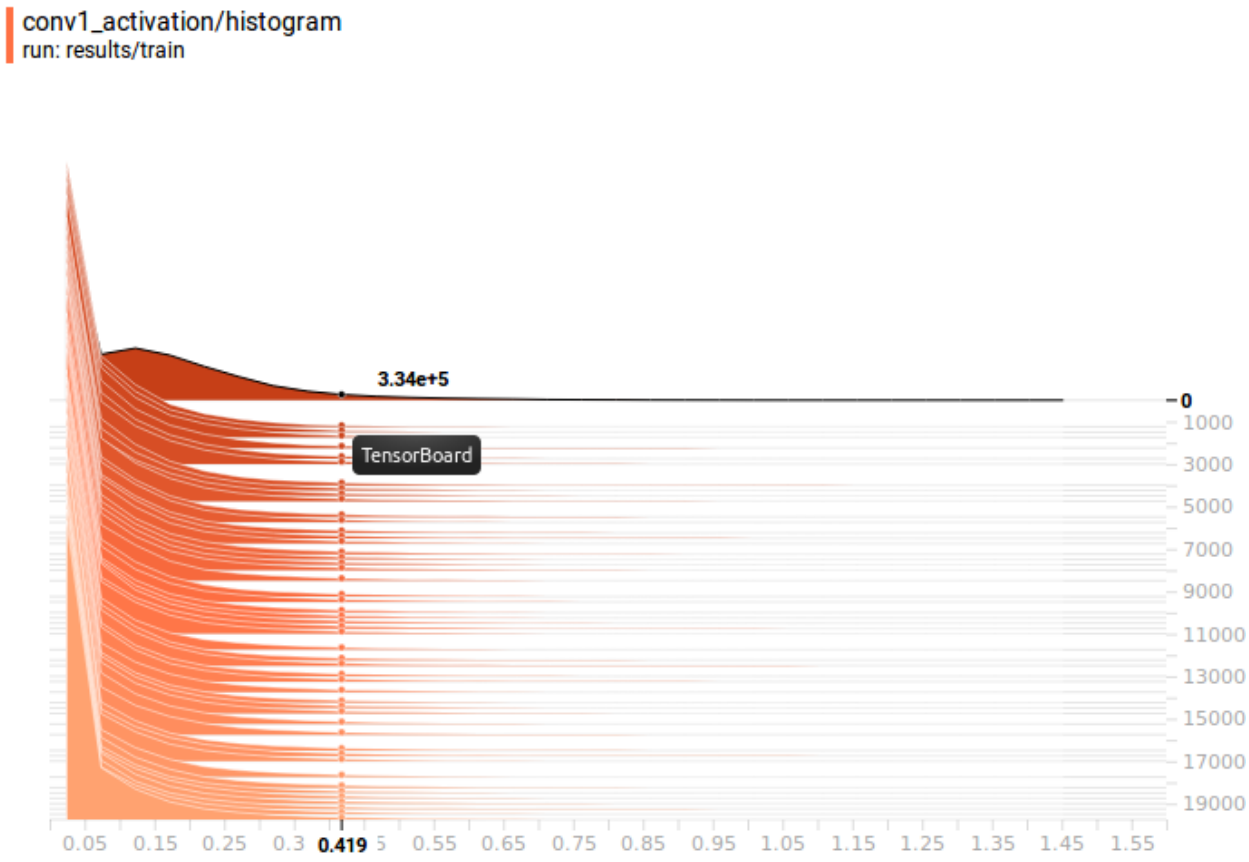

Figure 31: Activation Statistics for Testing Set Over Time

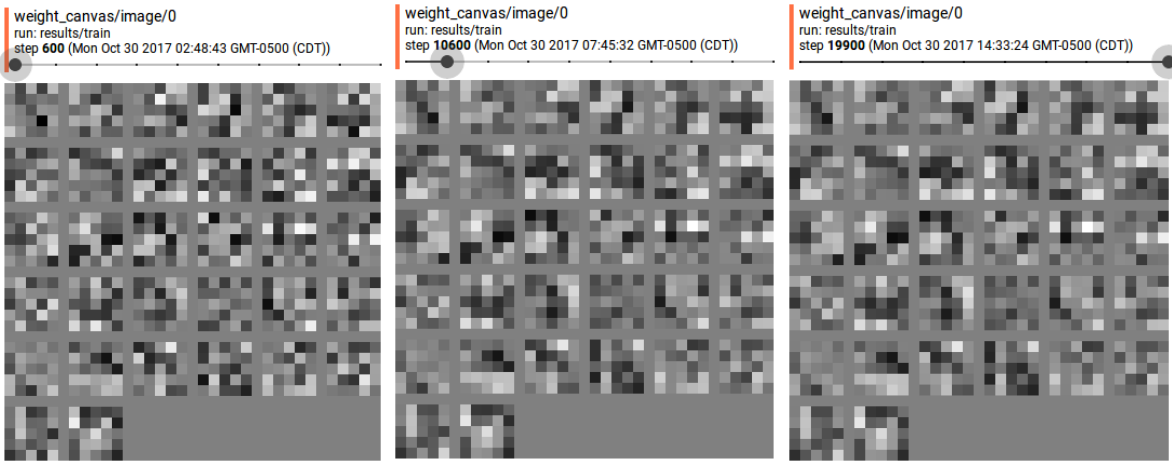

Figure 32: Filter Weights Throughout Training

1.3 Visualizing and Understanding Convolutional Networks

Introduction:
- Since ImageNet, many good models have popped out. Needs better way to visualize network to understand how training works
- Related work
Hard to interpret deeper layers
Instead this paper will try to plot features maps that fit the graph
- Approach
Convnet: convolution maxpool fully connected softmax - Visualization with a convent
Use deconvolutional network to visualize weights
One deconvnet attach each conv layer
To example one layer, keep other activation zero, pass in feature map, unpool, rectify, filter to reconstruct activity
Unpooling: keep set of switch variables (location of where the max is with in each pooling group
Rectification: uses relu activation, ensure maps always positive
Filtering: d-conv net flips filter vertically and horizontally
- Convnet Visualization
Feature Visualization:
Layer 2 responds to corner and edge color conjunctions
Layer 3: more complex invariance, similar textures (mess pattern)
Layer 4: significant variation, more class specific (dog faces)
Layer 5: entire object without pose variation (dogs)
Feature evolution during Training
Lower layer converge within few epochs, upper layer require 40-50 epochs to train
Feature invariance:
Small transformation have dramatic effect in first layer model, lesser impact later.
Network output is stable to translation
Output not invariant to rotation except objects with rotational symmetry
Architecture selection
To solve problem of first layer being mix of extremely high and low frequency information with no coverage of middle frequency, and aliasing artifacts caused by large stride size
Reduce 1st layer filter size
Reduce stride size
Occlusion sensitivity
Sensitivity of classification is based on image's surrounding context. Location matters a lot
Correspondence Analysis
By occluding eyes, nose, and specific features, we can see systematic changes in the weights. We can conclude that model does establish correspondence of feature between different images
- Experiments
imageNet 2012
reduced to almost half of best current error rate
get rid of fully connected layer didn't increase error rate too much. Get rid of middle conv layer didn't increase error rate too much, but can't get rid of both of them at same time. Overall depth is good for good performance
tried existing model, transfer learned on different data sets, does better than best existing model with minimal training image needed, but does badly if train from scratch
feature analysis
replace each layer with svm, and predict, as train on deeper layer, becomes more accurate

1.4 Build and Train an RNN on MNIST

1.4.1 Comparing LSTM, GRU, and basic RNN, varying hidden nodes

Optimizer	Learning Rate	Batch Size	Max Step	Keep Rate
Adam	0.001	100 (200 for GRU)	50000	0.6

Basic RNN:
LSTM:

Figure 33: RNN loss and accuracy using 50, 100, and 500 hidden nodes

GRU:
Summary:
From all three model, the less nodes there are, the more training it needs to converge, which is a little counter intuitive, because there are less parameters to train. Increasing number of hidden nodes does not necessarily correspond to increasing test accuracy. 100 is currently the best size to pic. The batch size determines how the size of jaggedness on the training plot
I can't really tell any differences between RNN, GRU and LSTM other than RNN performed the worst using this set of parameter and LSTM performed the best using this set of parameter.

Figure 34: LSTM loss and accuracy using 50, 100, 200 and 500 hidden nodes

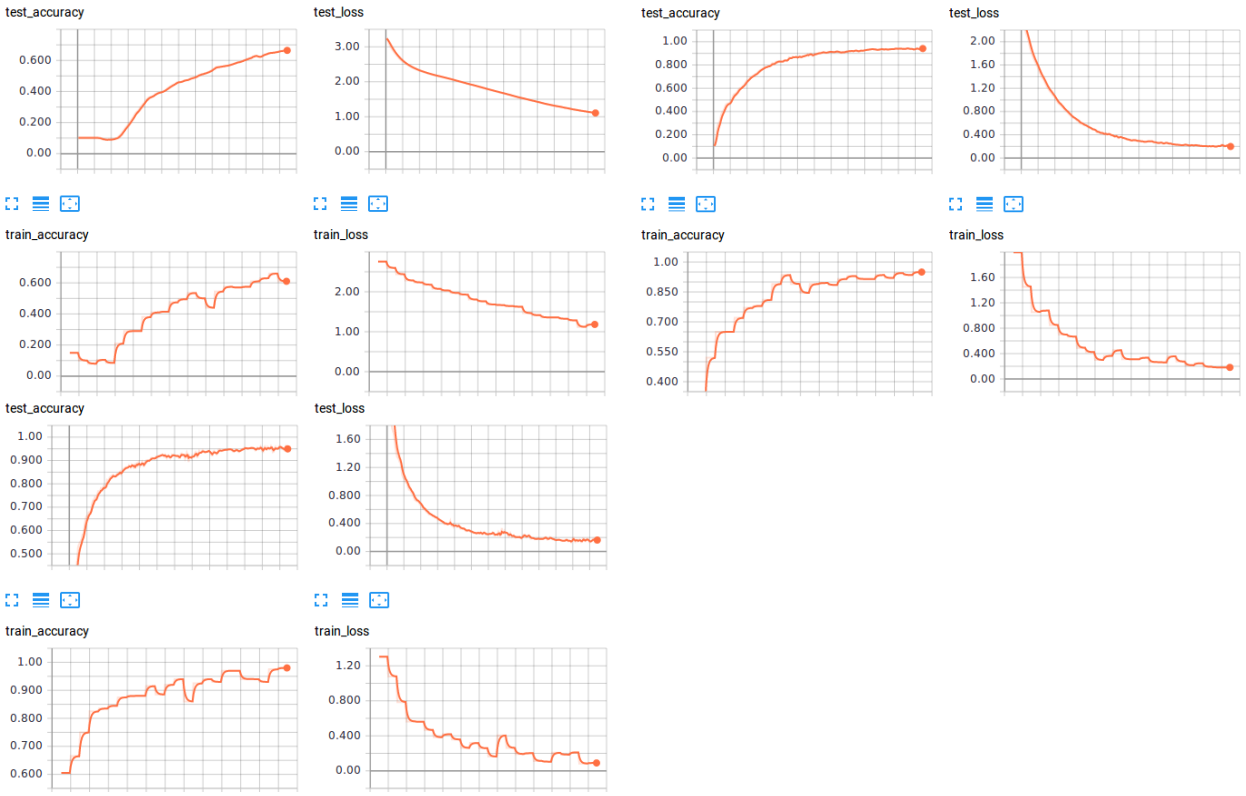

Figure 35: GRU loss and accuracy using 20, 100, and 500 hidden nodes

1.4.2 Comparing RNN with CNN

In general, RNN trains much faster than CNN, however, CNN performed slightly better in test accuracy. CNN test accuracy also seems to modulate more on the plateau as RNN is more stable. With proper optimization with hyper-parameter, both of them can achieve high accuracy.