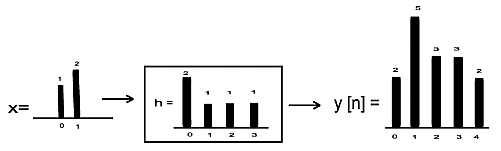
**Convolution Neural Network**

Convolution layers are used to extract the features of input training samples. Each convolution layer has a set of filters that helps in feature extraction. In general, as the depth of CNN model increases, complexity of features learnt by convolution layers increases. For example, first convolution layer captures simple features while the last convolution layer captures complex features of training samples.

1 line 🡪 2 line 🡪 Square 🡪 Combination of Squares 🡪 Combinations of squares and other shapes

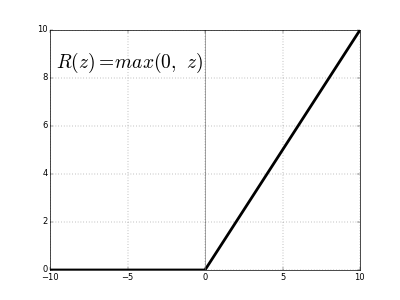
Features are extracted by taking the convolution of portion of data sample under consideration. The amount of data portion that the filter traverses each time is proportional to the stride length and padding value. Data samples may/ may not be subjected to zero padding before convolution.



Convolution of a Signal

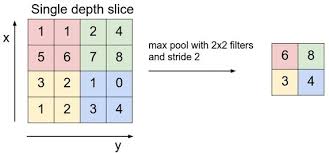
The output of a filter is then passed through an activation unit called ReLU (Rectified Linear Unit). This unit converts the data into it’s non-linear form. The output of ReLU is clipped to zero only if the result is negative else the output of convolution layer is retained.

Sigmoid units are not preferred as activation unit because of vanishing gradient problem. If the depth of CNN is large, then by the time the gradient found at the input layer reaches the output layer, the value of gradient found would diminish largely. This results in the output of the network changing only marginally. This, in turn, results in slow/no convergence. To avoid such a situation, ReLU is preferred.



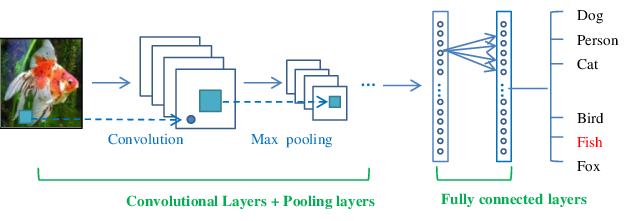
Output of ReLU

The output of ReLU is then passed through a pooling layer. This is used to remove any redundant features that’s captured during convolution. Thus, this layer reduces the size of data sample. The principle behind pooling is that it assumes that adjacent values of image pixels are nearly identical. The average/minimum/maximum of four adjacent pixel values are used to carry out pooling. In general, size of input image is reduced by half with help of a 2\*2 filter. The input data may/ may not be subjected to zero padding before pooling.



Max Pooling

This process of passing data through convolution and pooling layer successively is repeated per the design of CNN model. For learning purpose, this process is repeated 2-4 times. The output from successive convolution and pooling layer is then passed through a fully connected neural network layer. It’s functioning is like that of a multi-layer neural network. Here, each neuron unit acts as feature map that carries information about a unit.

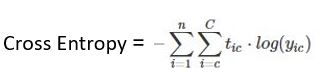


Design of Convolution Neural Network

Dropout layer is used to make the CNN model robust against noise. These layers are generally introduced between 2 fully connected neural network layers. They temporarily cut a portion of data flowing between two fully connected layers. This is equivalent to making the model learn to classify accurately in presence of noise. Thus, chances of model classifying inaccurately because of overfitting is reduced.

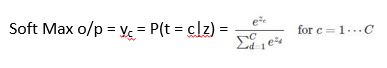
The output of CNN model is calculated using SoftMax function. SoftMax is preferred as it gives the probability of outputs for different classes rather than just >= 0.5 in the case of sigmoid output. The usage of SoftMax function to find output results based on the highest probability of class results in an increase in accuracy the of output.

Cross entropy is used to measure the performance of the system. They are calculated with help of a SoftMax function. The advantage here is that the SoftMax output is the trace of the elements corresponding to the class that we know that the output belongs too. This, in general, saves the computation time.



Here,

* Tic = Target Output
* yic = Soft Max Output
* C= = Number of Classes
* N = Number of Data Samples



Here,

* yc = probability of current output belonging to class c
* Numerator = exponential of weighted sum o/p of class c
* Denominator = sum of exponential of weighted sum o/p of classes 1 to C

**Overview of CNN Layers**

|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Purpose | Implementation | Reason for Working |
| Convolution | Extract features from i/p data sample | y(t) = x(t)\*h(t-T) | Small size filters trace the entire data sample at a faster rate. Hence, they learn from the mapping with pictures effeciently. |
| ReLU | Convert data to non-linear form | y = y, when i/p > 0  0, otherwise | Removes problem of vanishing gradient descent |
| Pooling | Remove redundant features  Make model invariant to translation, rotation and scaling | y = max of 4 adjacent  feature values | Features next to each other have similar values |
| Fully Connected Neural Network | Classification of i/p data | Back Propagation Algorithm | Each neuron acts as a feature map |
| Dropout | Make model robust to noise | Temporarily cut the flow of small portion of data b/w 2 fully connected layers | Reduces overfitting by making the model less complex |
| Regularization | Make model robust to noise | Penalizes cost function and weight updates for every wrong prediction | Reduces overfitting by making the model less complex |
| Multiple Convolution Layer | Extract high level/more complex features | y(t) = x(t)\*h(t-T) | Lesser number of filters enable faster extraction of features from data set |

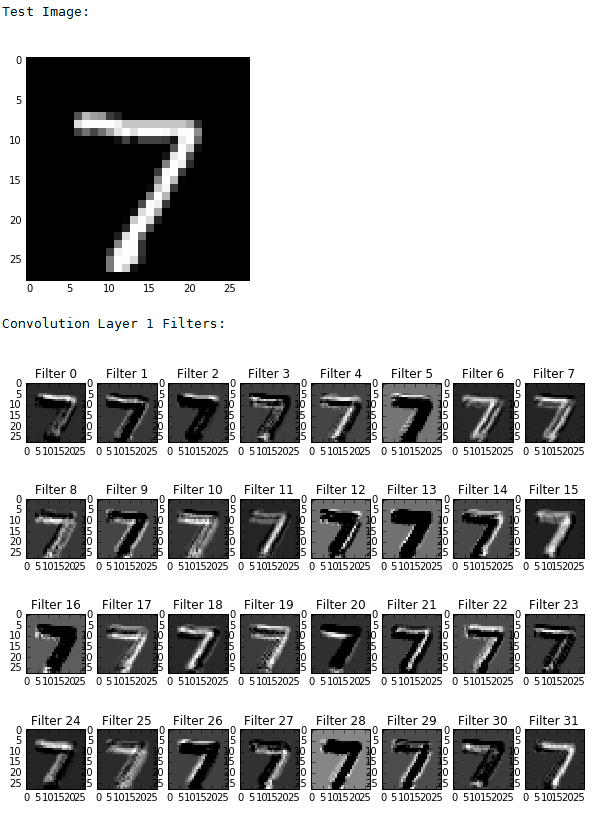


Figure 1 Convolution Layer 1 Output

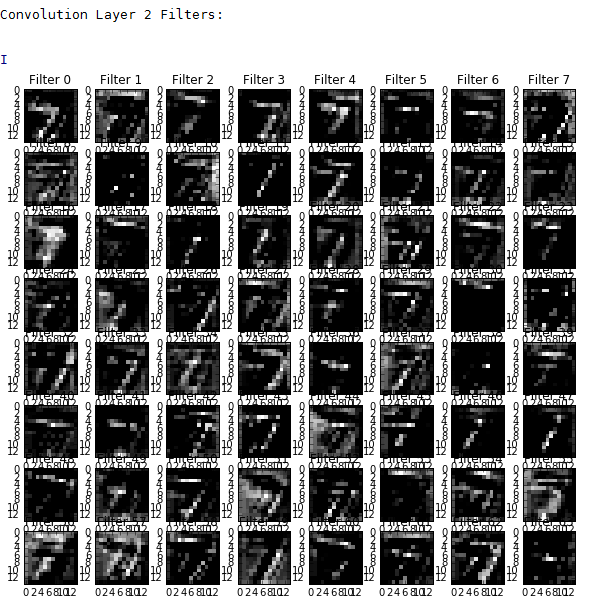


Figure 2 Convolution Layer 2 Output

**Note:**

The convolution outputs represents how that image activates the neurons of the convolutional layers. Each filter learns to activate optimally for different features of the image.

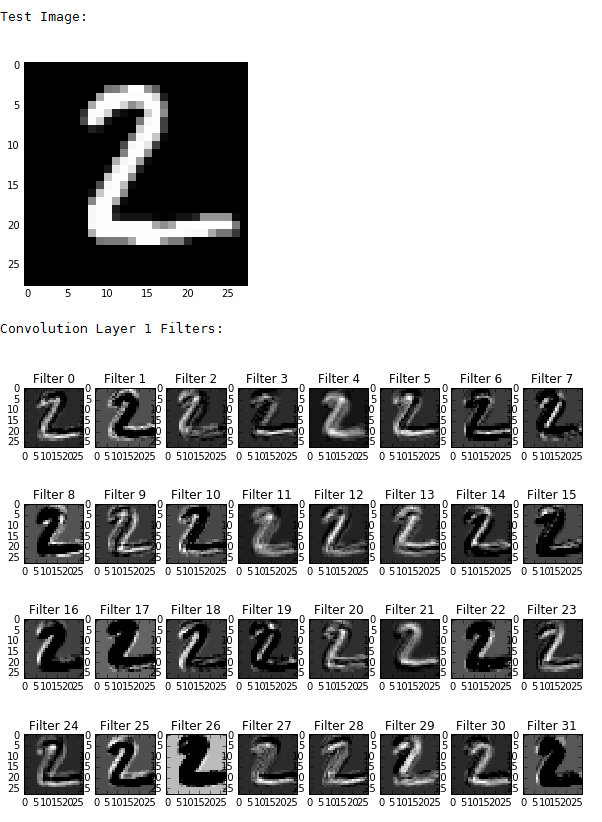


Figure 3 Convolution Layer 1 Output

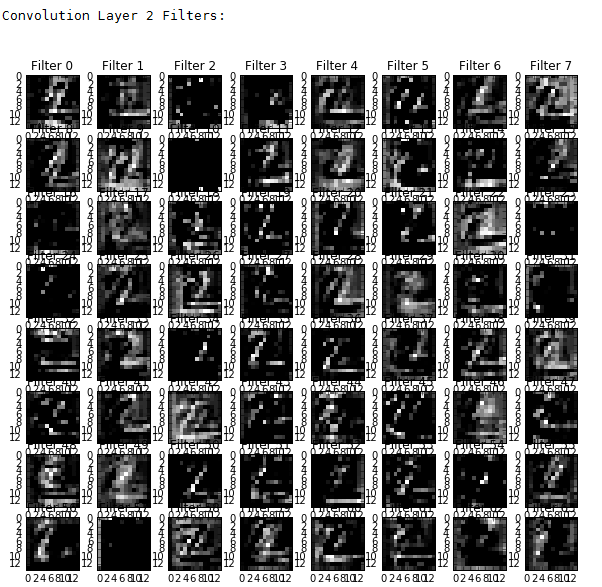


Figure 4 Convolution Layer 2 Output

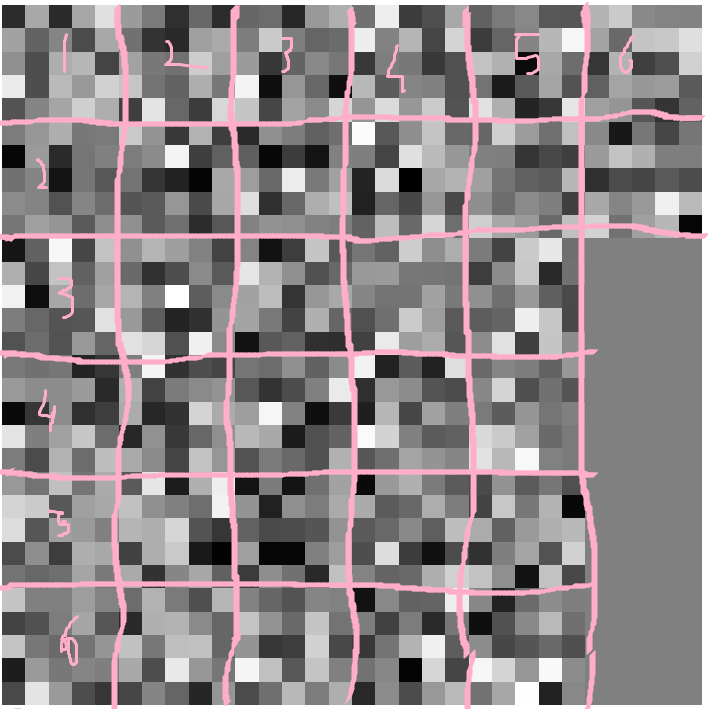


Figure 5 Convolution Layer 1 Filter Output

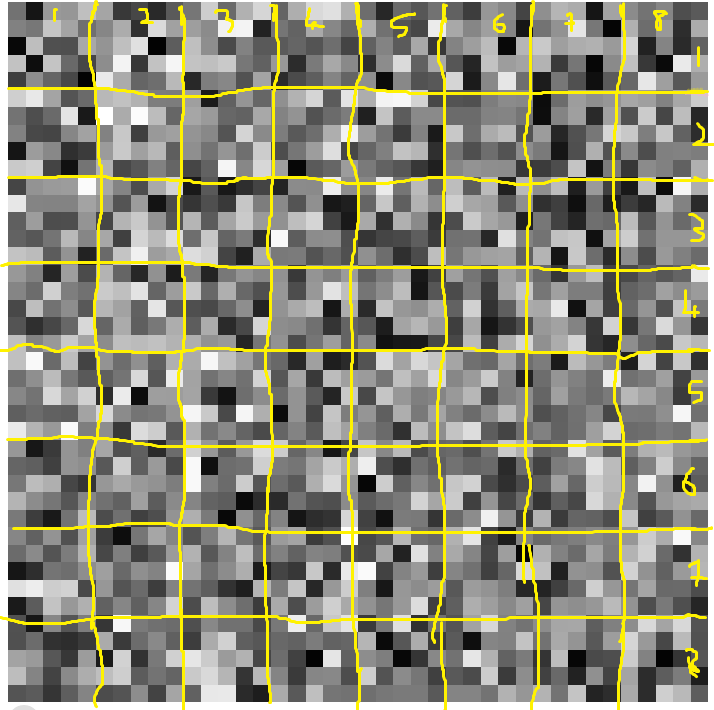
****

Figure 6 Convolution Layer 2 Filter Output

**Architecture Design**

1. Default

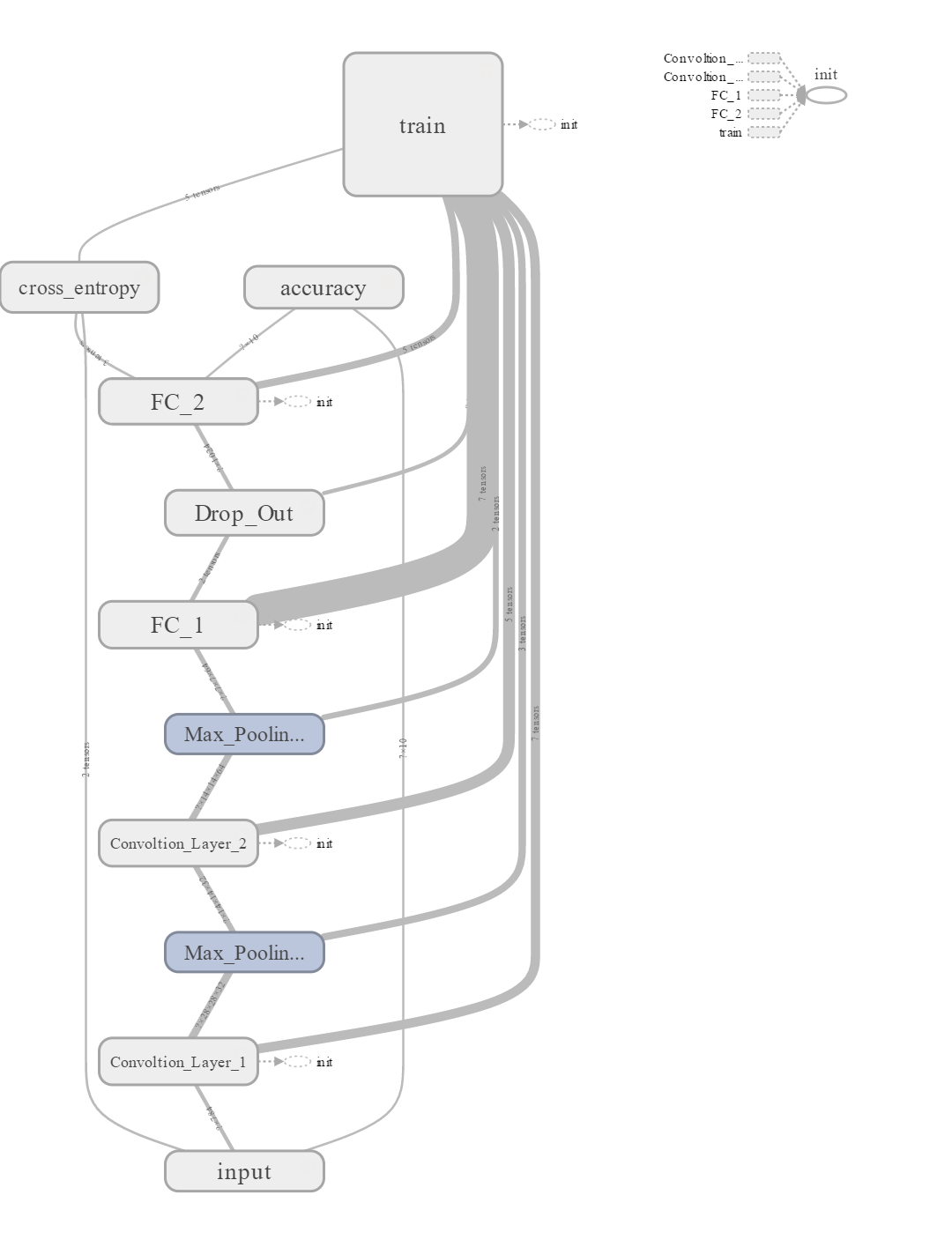
|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Input | Output | Filter |
| Conv\_1 | (?, 32, 32, 1) | (?, 28, 28, 32) | 5\*5 |
| Pool\_1 | (?, 28, 28, 32) | (?, 14, 14, 32) | 2\*2 |
| Conv\_2 | (?, 14, 14, 32) | (?, 14, 14, 64) | 5\*5 |
| Pool\_2 | (?, 14, 14, 64) | (?, 7, 7, 64) | 2\*2 |
| Flatten | (?, 7, 7, 64) | (?, 3136) | N/A |
| FA\_1 | (?, 3136) | (?, 1024) | N/A |
| FA\_1 | (?, 1024) | (?, 10) | N/A |

1. LeNet

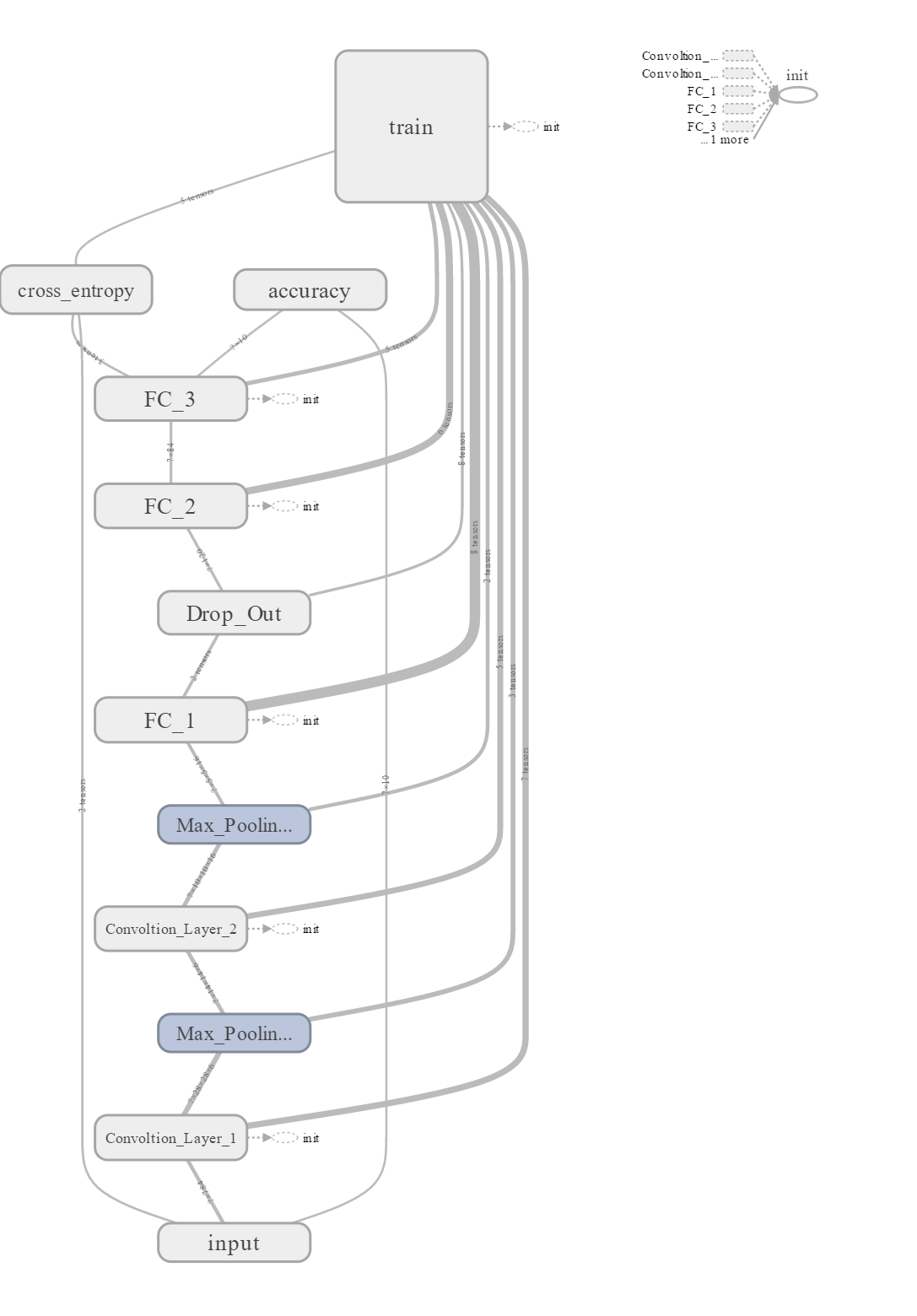
|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Input | Output | Filter |
| Conv\_1 | (?, 32, 32, 1) | (?, 28, 28, 6) | 5\*5 |
| Pool\_1 | (?, 28, 28, 6) | (?, 14, 14, 6) | 2\*2 |
| Conv\_2 | (?, 14, 14, 6) | (?, 10, 10, 16) | 5\*5 |
| Pool\_2 | (?, 10, 10, 16) | (?, 5, 5, 16) | 2\*2 |
| Flatten | (?, 5, 5, 16) | (?, 400) | N/A |
| FA\_1 | (?, 400) | (?, 120) | N/A |
| FA\_2 | (?, 120) | (?, 84) | N/A |
| FA\_2 | (?, 84) | (?, 10) | N/A |

1. Modified

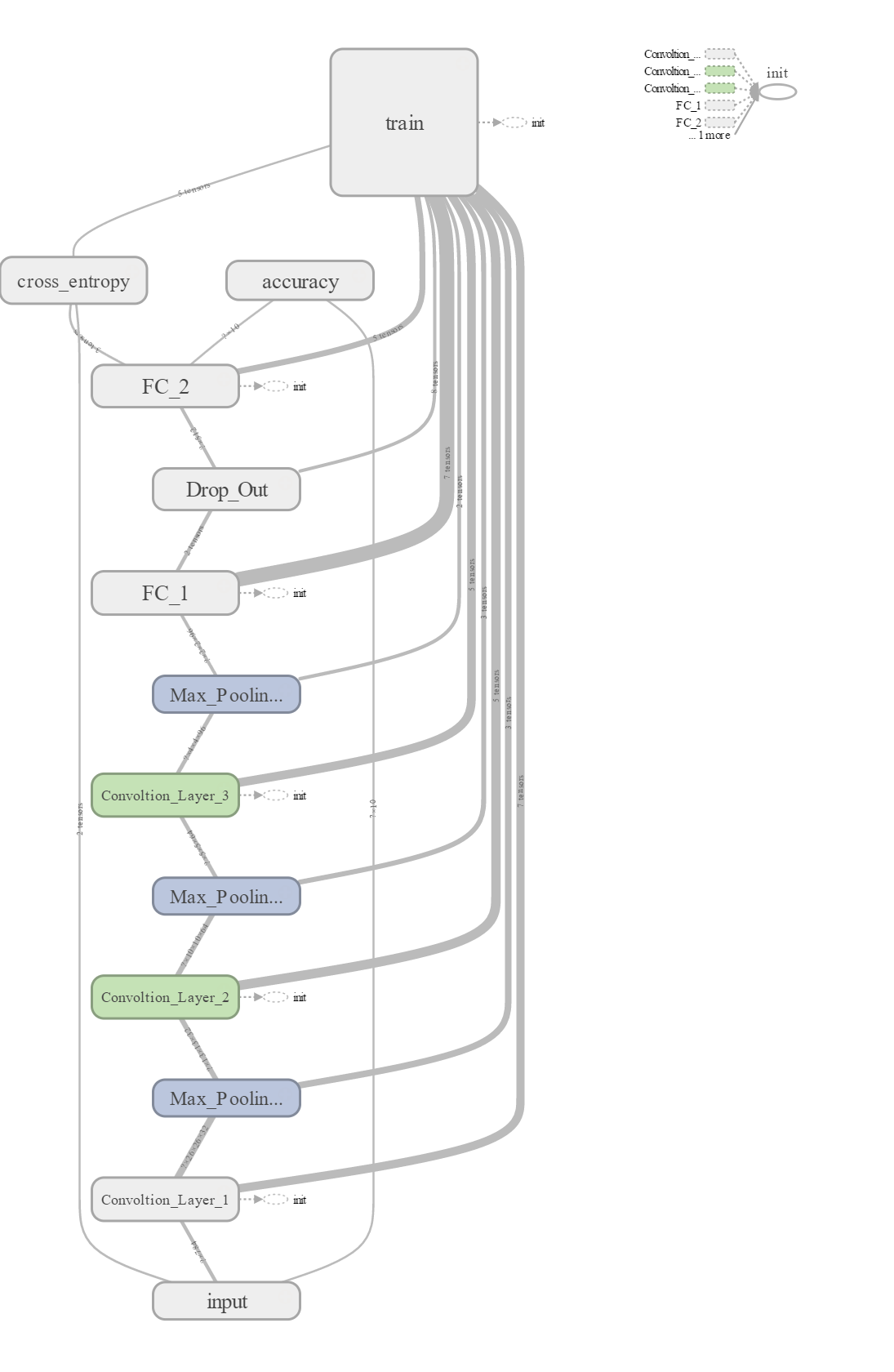
|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Input | Output | Filter |
| Conv\_1 | (?, 28, 28, 1) | (?, 26, 26, 32) | 3\*3 |
| Pool\_1 | (?, 26, 26, 32) | (?, 13, 13, 32) | 2\*2 |
| Conv\_2 | (?, 13, 13, 32) | (?, 10, 10, 64) | 4\*4 |
| Pool\_2 | (?, 10, 10, 64) | (?, 5, 5, 64) | 5\*5 |
| Conv\_3 | (?, 5, 5, 64) | (?, 4, 4, 96) | 2\*2 |
| Pool\_3 | (?, 4, 4, 96) | (?, 2, 2, 96) | 2\*2 |
| Flatten | (?, 2, 2, 96) | (?, 384) | N/A |
| FA\_1 | (?, 384) | (?, 512) | N/A |
| FA\_2 | (?, 512) | (?, 10) | N/A |

****

**Default**

****

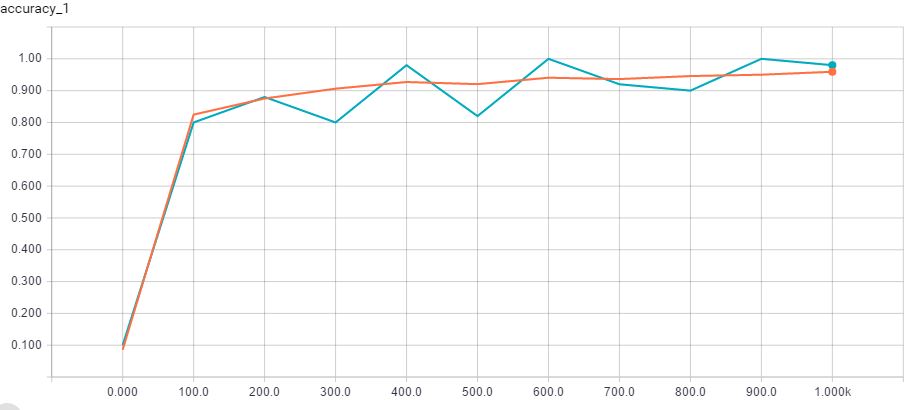
**LeNet**

****

**Modified**

**Accuracy**

1. Default

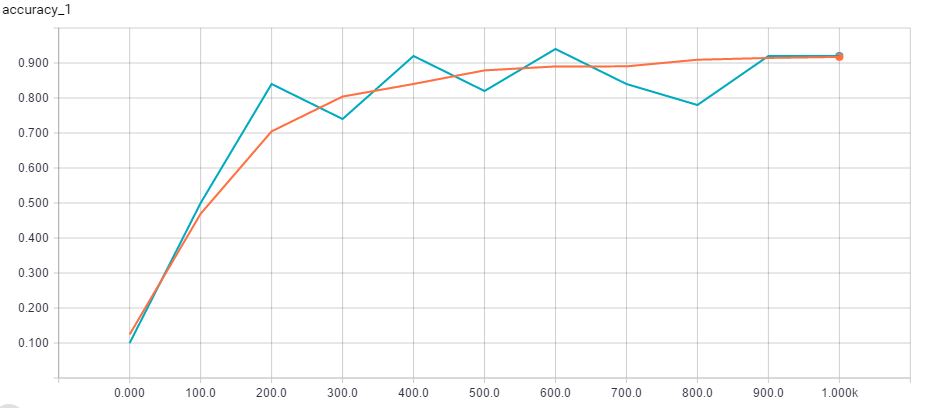
****

Test

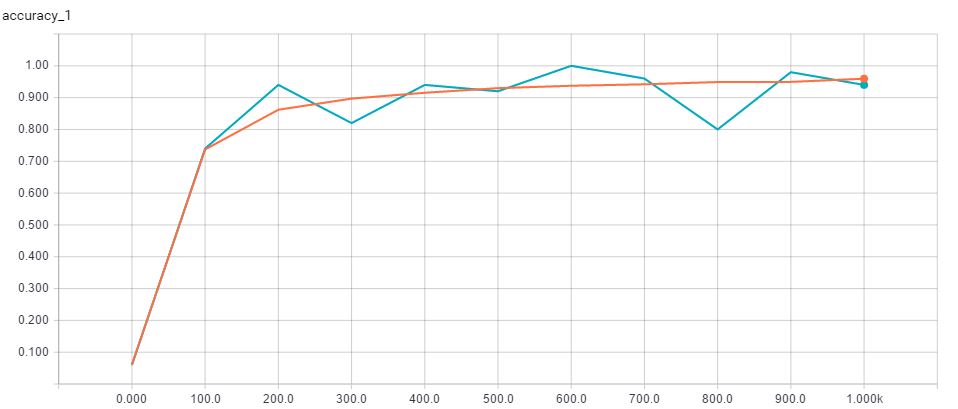
Training

Dropout = 0.1

1. **LeNet**

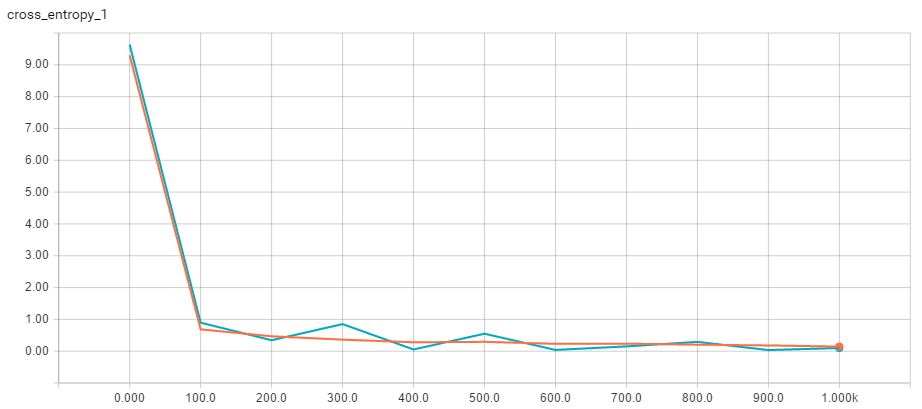
****

1. **Modified**

****

**Cross Entropy**

1. Default

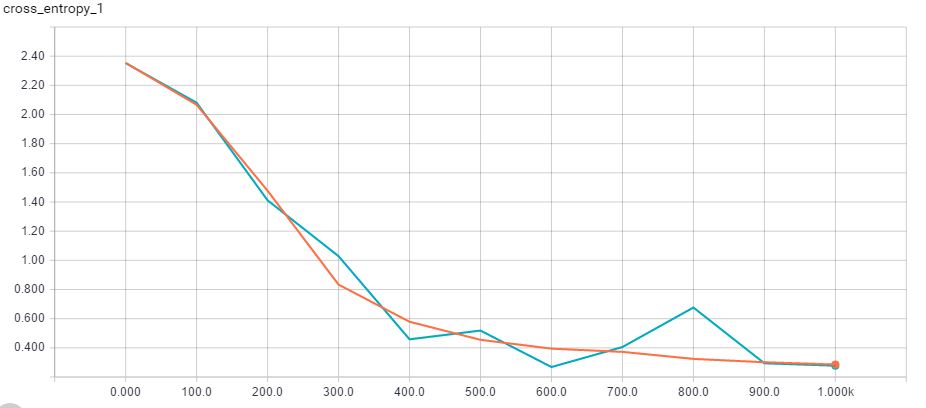
****

Test

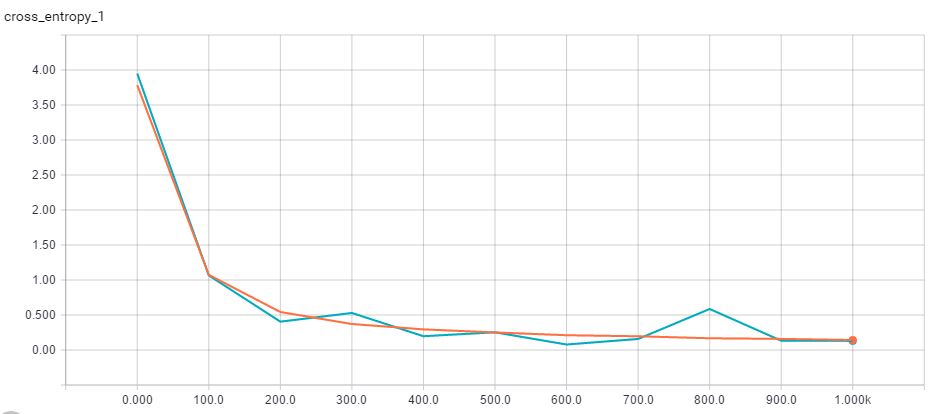
Training

Dropout = 0.1

1. LeNet

****

1. Modified



**Results**

1. Dropout = 0.0

## Accuracy

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Iteration | Default | | LeNet | | Modified | |
| Train | Test | Train | Test | Train | Test |
| 0 | 0.1 | 0.1136 | 0.14 | 0.0958 | 0.12 | 0.0855 |
| 100 | 0.88 | 0.8118 | 0.62 | 0.5424 | 0.72 | 0.777 |
| 200 | 0.88 | 0.8735 | 0.84 | 0.7531 | 0.94 | 0.8926 |
| 300 | 0.8 | 0.8984 | 0.82 | 0.8288 | 0.92 | 0.9127 |
| 400 | 0.96 | 0.9136 | 0.82 | 0.851 | 0.96 | 0.9315 |
| 500 | 0.88 | 0.9186 | 0.86 | 0.8617 | 0.9 | 0.9375 |
| 600 | 1 | 0.9269 | 0.96 | 0.8862 | 1 | 0.9407 |
| 700 | 0.96 | 0.9298 | 0.88 | 0.8904 | 0.92 | 0.9448 |
| 800 | 0.84 | 0.9404 | 0.76 | 0.8985 | 0.84 | 0.9512 |
| 900 | 0.98 | 0.9456 | 0.86 | 0.9065 | 0.94 | 0.9467 |
| 1000 | 0.92 | 0.9498 | 0.92 | 0.9109 | 0.94 | 0.9615 |

1. Dropout = 0.1

## Accuracy

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Iteration | Default | | LeNet | | Modified | |
| Train | Test | Train | Test | Train | Test |
| 0 | 0.1 | 0.0857 | 0.1 | 0.1245 | 0.06 | 0.0612 |
| 100 | 0.8 | 0.8247 | 0.5 | 0.4693 | 0.74 | 0.7372 |
| 200 | 0.88 | 0.8752 | 0.84 | 0.7047 | 0.94 | 0.862 |
| 300 | 0.8 | 0.9062 | 0.74 | 0.8041 | 0.82 | 0.8969 |
| 400 | 0.98 | 0.9271 | 0.92 | 0.8401 | 0.94 | 0.9153 |
| 500 | 0.82 | 0.9204 | 0.82 | 0.8789 | 0.92 | 0.9298 |
| 600 | 1 | 0.9406 | 0.94 | 0.8902 | 1 | 0.9373 |
| 700 | 0.92 | 0.9363 | 0.84 | 0.8904 | 0.96 | 0.9419 |
| 800 | 0.9 | 0.9458 | 0.78 | 0.9093 | 0.8 | 0.949 |
| 900 | 1 | 0.9501 | 0.92 | 0.9144 | 0.98 | 0.9495 |
| 1000 | 0.98 | 0.9593 | 0.92 | 0.9174 | 0.94 | 0.9595 |

1. Dropout = 0.5

## Accuracy

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Iteration | Default | | LeNet | | Modified | |
| Training | Test | Training | Test | Training | Test |
| 0 | 0.12 | 0.0866 | 0.16 | 0.1374 | 0.1 | 0.119 |
| 100 | 0.84 | 0.8175 | 0.6 | 0.5079 | 0.68 | 0.7068 |
| 200 | 0.88 | 0.8853 | 0.78 | 0.7023 | 0.94 | 0.8438 |
| 300 | 0.82 | 0.9024 | 0.7865 | 0.7865 | 0.78 | 0.8873 |
| 400 | 0.96 | 0.9181 | 0.821 | 0.821 | 0.92 | 0.9084 |
| 500 | 0.88 | 0.9221 | 0.8518 | 0.8518 | 0.84 | 0.9234 |
| 600 | 1 | 0.9315 | 0.873 | 0.873 | 0.98 | 0.9316 |
| 700 | 0.98 | 0.934 | 0.882 | 0.882 | 0.94 | 0.9353 |
| 800 | 0.86 | 0.9433 | 0.8915 | 0.8915 | 0.86 | 0.943 |
| 900 | 1 | 0.9471 | 0.8992 | 0.8992 | 0.98 | 0.9463 |
| 1000 | 0.94 | 0.9519 | 0.9035 | 0.9035 | 0.94 | 0.9522 |

**Observations**

* The time needed to train the network increases with increase in depth of the model.
* The lower value of dropout probabilities results in increased accuracy of the model. While extreme values of dropout probability, decrease the accuracy of the model.
* The accuracy of the model is highest for the architecture found on TensorFlow tutorial (Default), while its lowest for the one that is inspired by LeNet model. This is because of the filter mappings between LeNet layers is not same as that of the original LeNet model. The accuracy of the Modified version of Default CNN model is considerably better. The reason for this is the presence of an additional convolution and sampling layer that captures additional information of the image to be classified.