Analysis of customized Convolutional Neural Network

Submitted by **: S Akash IIT2015067**

Introduction

Convolutional neural networks are deep artificial neural networks that are used primarily to classify images (e.g. name what they see), cluster them by similarity (photo search), and perform object recognition within scenes. They are algorithms that can identify faces, individuals, street signs, tumors, platypuses and many other aspects of visual data.

Convolutional networks perform optical character recognition (OCR) to digitize text and make natural-language processing possible on analog and handwritten documents, where the images are symbols to be transcribed. CNNs can also be applied to sound when it is represented visually as a spectrogram. More recently, convolutional networks have been applied directly to text analytics as well as graph data with graph convolutional networks.

The efficacy of convolutional nets (ConvNets or CNNs) in image recognition is one of the main reasons why the world has woken up to the efficacy of deep learning. They are powering major advances in computer vision (CV), which has obvious applications for self-driving cars, robotics, drones, security, medical diagnoses, and treatments for the visually impaired.

Model selection is of high significance when using images as much information should be extracted from smaller operations and smaller inputs as possible. The images are already huge data points and to get accurate results by huge computations done on small scale is required.

Problem Statement

This report demonstrates how changing the hyperparameters cause change in accuracy of the model. We start with taking a simple architecture and expand and change hyperparameters like strides, optimization function, padding and size of filters and analyze the outcome.

Dataset

We use the MNIST dataset of Keras. The dataset contains 60,000 images in training dataset and 10,000 images in test set. To train entire dataset would take at least 1 hour when run on CPU
Architecture, so we take 10 percent of the dataset. This we have :
6000 training set examples
1000 test ste examples
The images are 28 x 28 images with only one channel.

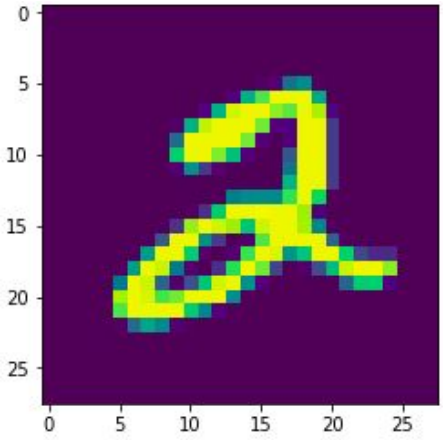


Figure 1 : MNIST data sample

The labels for this dataset are numbers from 0 to 9.
The distribution of the classes is as shown:

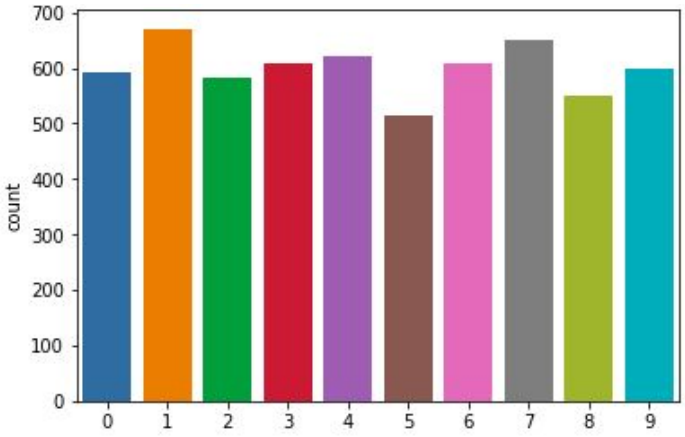


Figure 2 : MNIST data classes distribution

Methodology

Architecture 1

We start with a basic architecture, two convolutional layers, followed by a max pool layer and a fully connected layer. The Conv layers used 8 filters each of size 5 x 5 and the max pooling layer had a filter of size 2 by 2 and stride as 2.

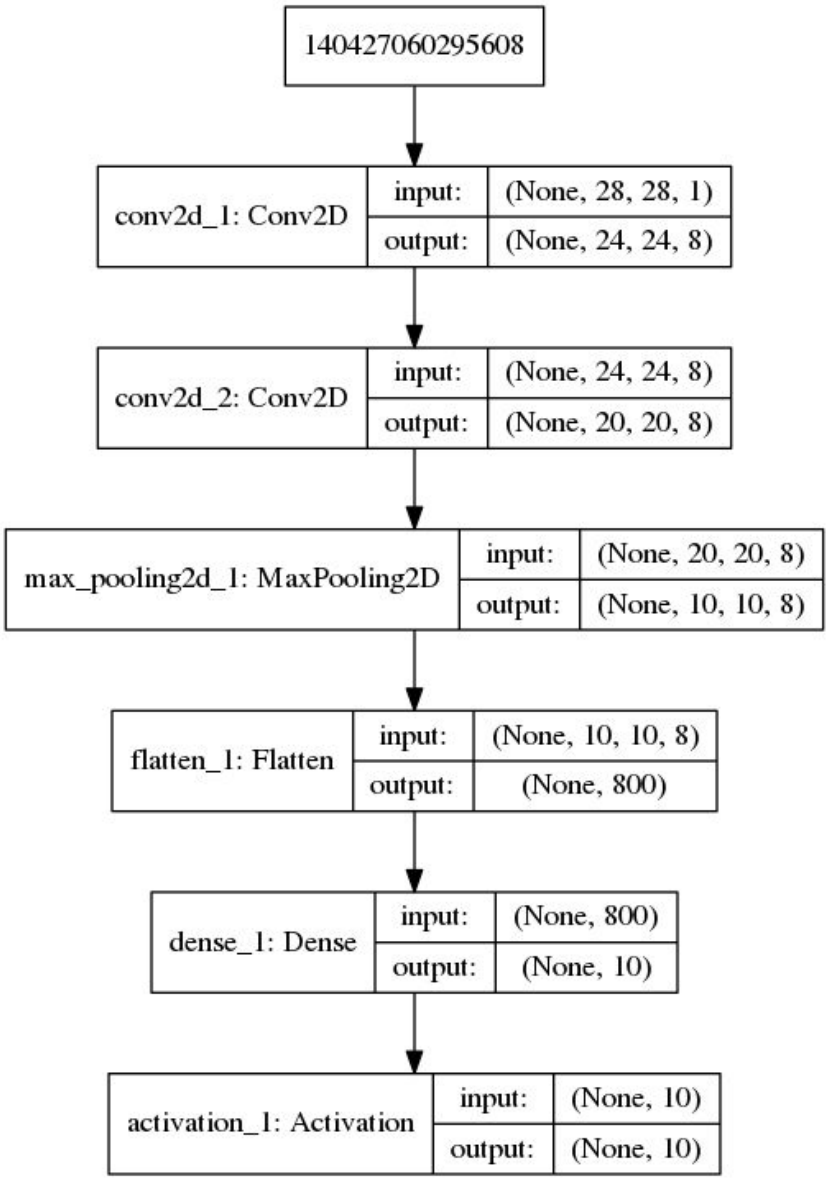


Figure 2 : Model 1 skeleton

Architecture 2

The architecture for model 2 was the same with the optimization function changed from Adadelta to Stochastic Gradient Descent.

Architecture 3

The architecture for model 3 is addendum to architecture for model 1 and model 2. We change the padding from 0 to same for the first layer. The output size would now be equal to input size with respect to first conv layer. We change the stride from 1 to 2 for the second Conv layer and use Adam optimization function.

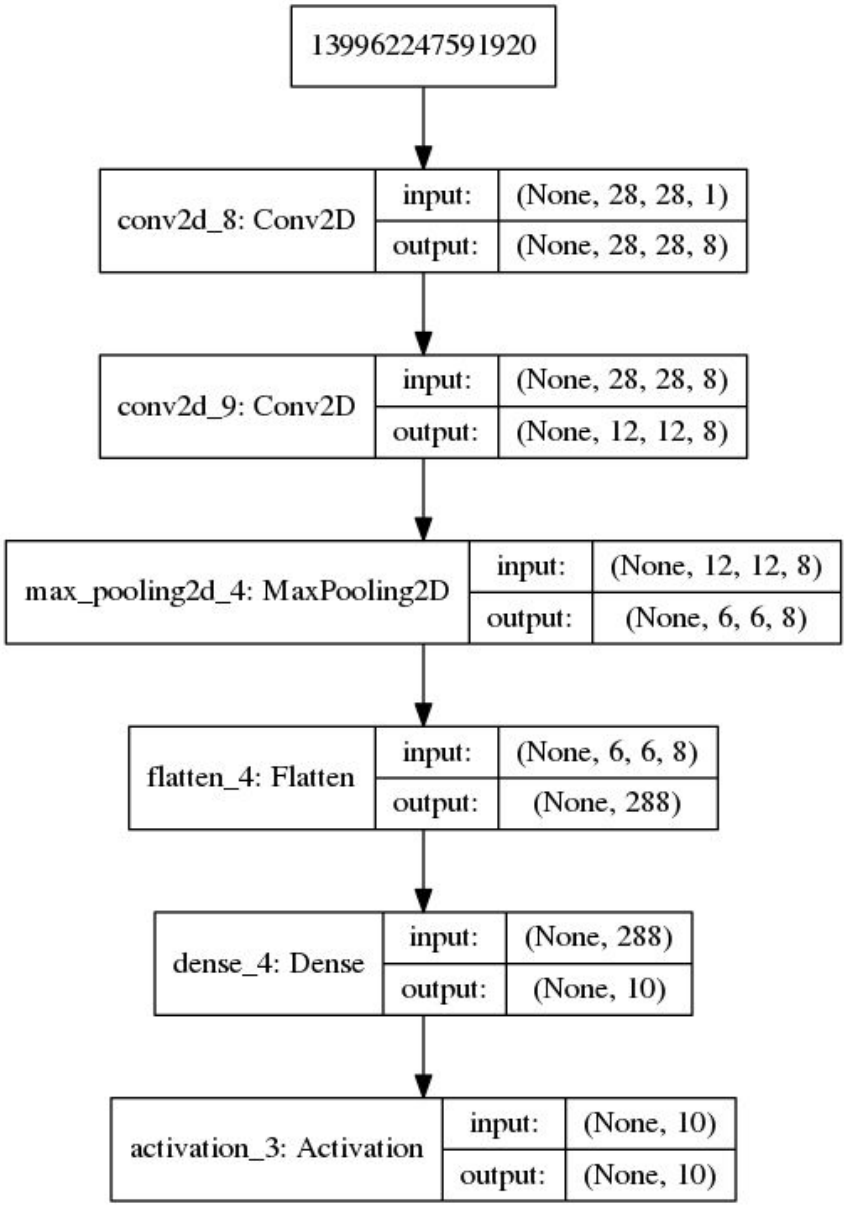


Figure 4 : Model 3 architecture

Architecture 4

The fourth model is again an addendum to the previously discussed model architectures. The filter size for convolution layers change from (5,5) to (3,3) and

increase the number of filters from 8 to 32 for the first convolution layer and to 64 in second convolution layer. We add a dropout layer with a dropout layer of 0.25 after the max pooling layer and use Adadelta as our optimization function.

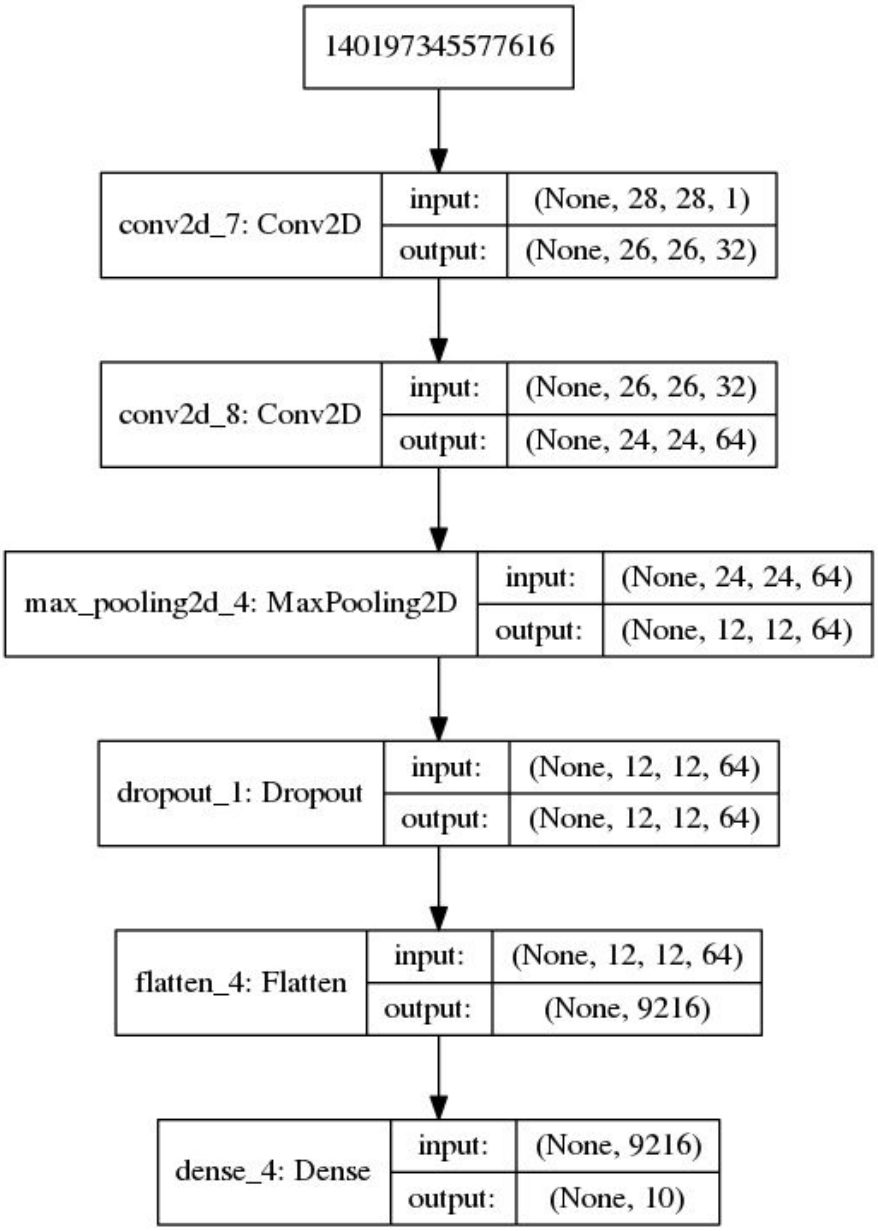


Figure 5 : Model 4 architecture

Architecture 5

We add another Dropout layer to model 4 with a dropout rate of 0.5.

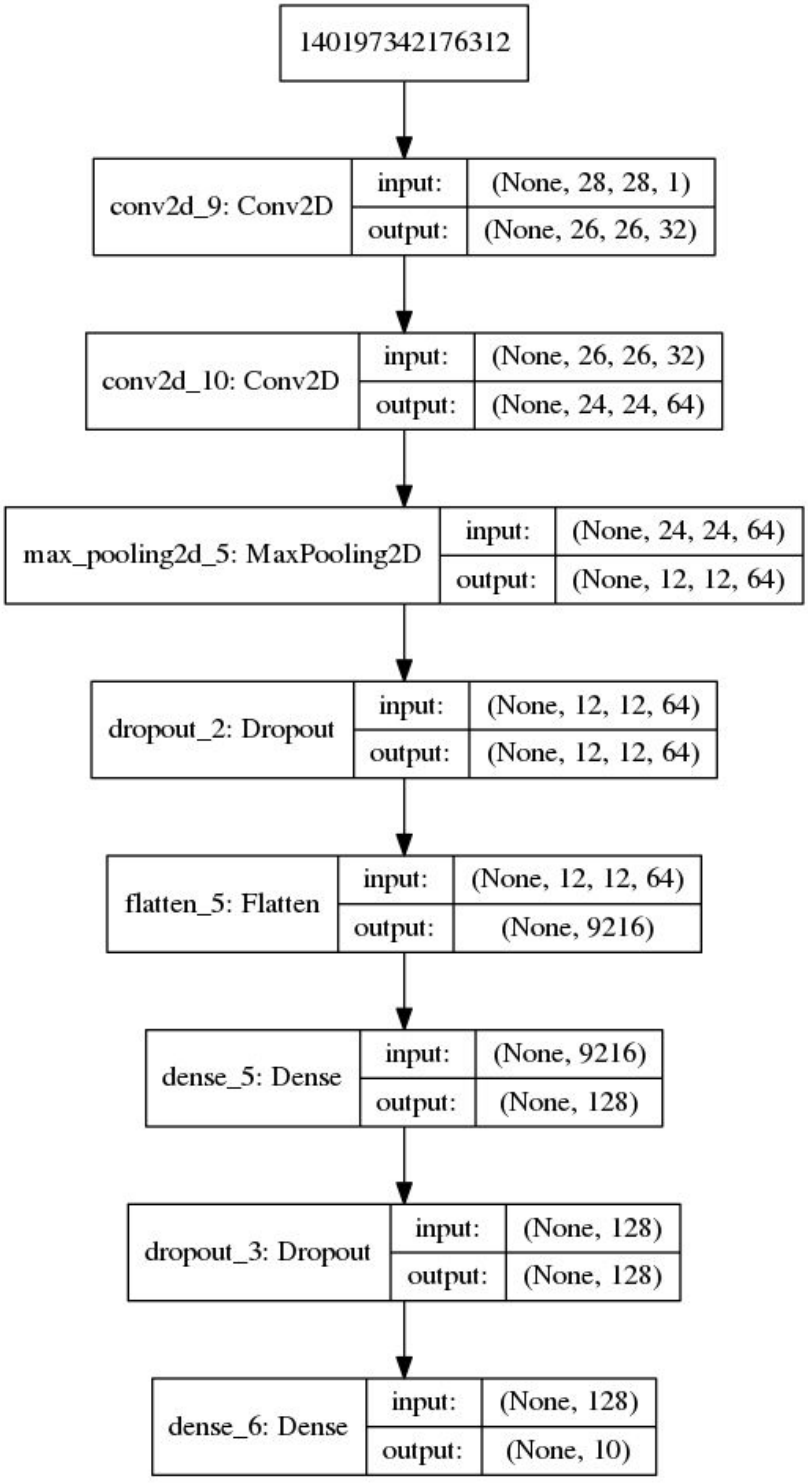


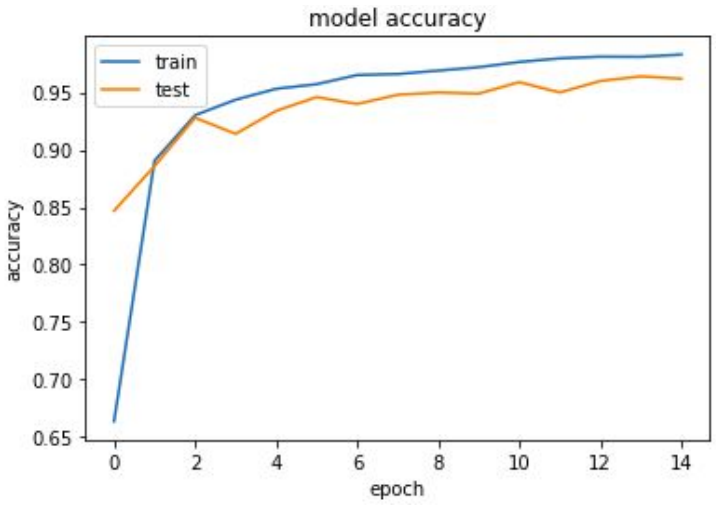
Figure 6 : Model 5 architecture

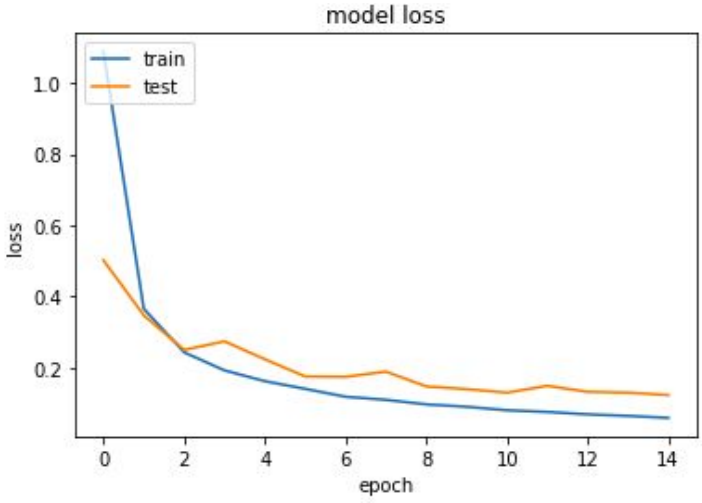
Analysis and results

When we start of with the first model proposed, we find to acquire the following metric results:

Test loss: 0.12365462710368666
Accuracy on test set: 0.9619619619619619

With the model accuracy and loss as shown:



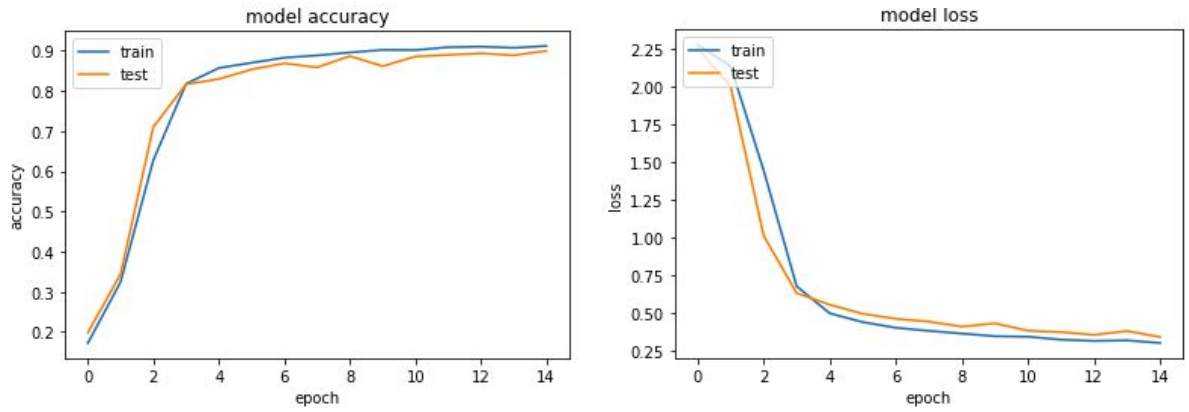


Which is pretty good for a start. As the Adadelta optimization considers changing learning rate as well when training, this solves problem of overshooting when the learning rate is high or slow convergence when learning rate is low. Adaptive learning causes dynamic shift in the rate of convergence, that lead to take time but results in good performance.

In model 2, we change the learning rate to SGD. This results in decrease in accuracy and significant increase in model loss. Consider the results:

Test loss: 0.3382477292099276
Test accuracy: 0.8998998998998999

With model accuracy and loss as shown:

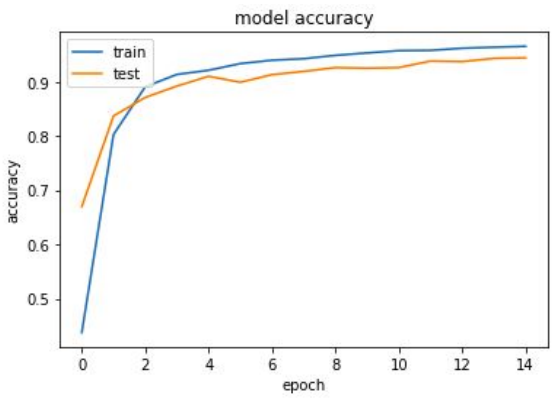


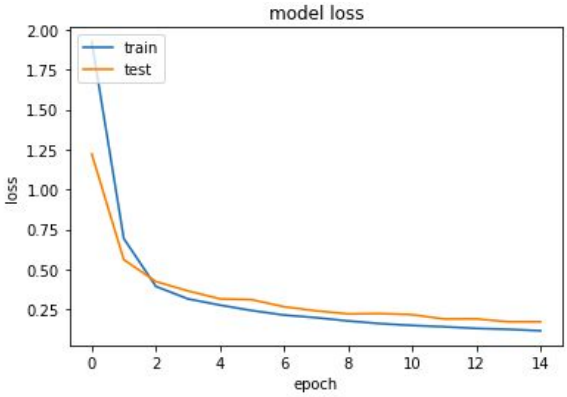
This occurs as now the model is not adjusting the learning rate but uses the same learning rate throughout the training. This leads to overshooting the convergence point if learning rate is high and slow convergence in case of low learning rate, resulting in decrease in accuracy.

In model 3, we see increase in accuracy, thanks to Adam optimization function that uses momentum and learning rate decay into consideration. Though increasing the stride leads to missing some features and hamper the learning. Same padding results in non-shrinking of intermediate input in subsequent layers thus preserving the features as required. The result for this model is:

Test loss: 0.17176926206868212
Test accuracy: 0.943943943943944

With the accuracy and loss as shown varying through iterations:

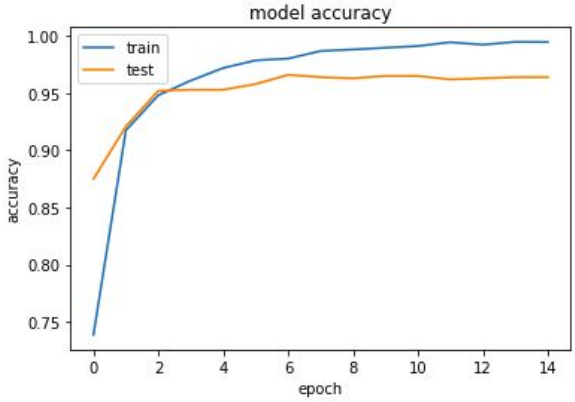


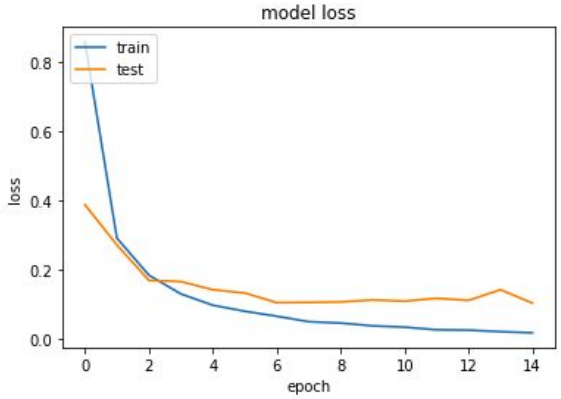


Drastically changing the model results in significant change. The smaller the filter size the more information is gathered by them, for the same information, we would need a considerate amount of filters bundled together. Dropout helps in regularization and decreases chances of overfitting. Thus, the results shown as evidence:

Test loss: 0.10288116631615027
Test accuracy: 0.963963963963964

With the accuracy and loss of the model through iterations as shown:

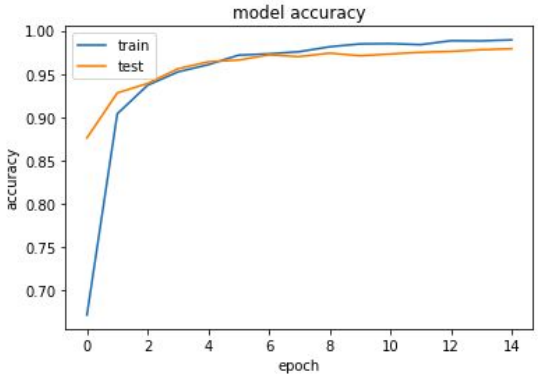


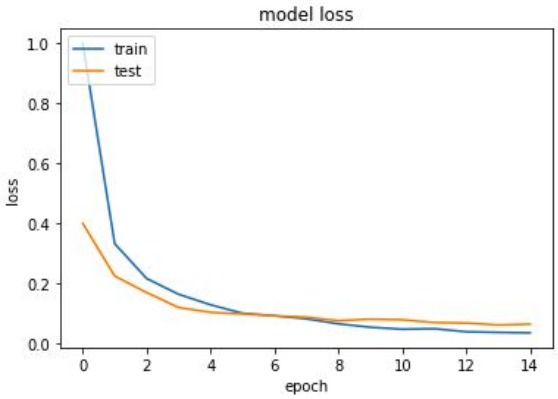


In the final model, we increase dropout layers, thus leading to more regularization of the model and we find the accuracy to be highest so far:

Test loss: 0.0641570260954645
Test accuracy: 0.978978978978979

With the model accuracy and loss through iterations shown as follows:





Thus we gather the summary of the results:

Model	Architectural change	Accuracy
1	optimization function (Adadelta)	0.9619619619619619
2	optimization function (SGD)	0.8998998998998999
3	optimization function (Adam), Padding, stride	0.943943943943944
4	Dropout Layer	0.963963963963964
5	2 Dropout Layers	0.978978978978979