Neural net is a computational learning system. It uses a network of functions to understand and translate a data input of one from into a desired output. This model concept was inspired by human biology and the way neurons of human brain function together to understand inputs from human senses.

Neural networks are just one of many tools and approaches used in machine learning algorithms. The neural network itself may be used as a piece in many different machine learning algorithms to process complex data inputs into a space that computers can understand.

It is being applied to many real-life problems today including speech and image recognition, spam email filtering, finance, and medical diagnosis etc.

A brief discussion of working principle used in Neural Network is given below,

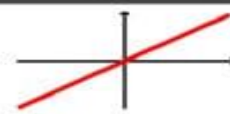
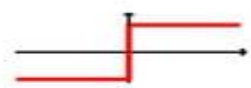
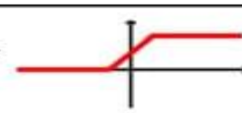
- Fully Connected Layer:
  Fully Connected layers in a neural network are those layers where all the inputs from one layer are connected to every activation unit of the next layer. In most popular machine learning models, the last few layers are full connected layers which compiles the data extracted by previous layers to form the final output.
- Convolution Layer:
  A convolution is the simple application of a filter to an input that results in an activation. Repeated application of the same filter to an input results in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input, such as an image.

  Convolutional layers are the major building blocks used in convolutional neural networks.

- Pooling Layer:
  Pooling layers reduce the dimensions of data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, tiling sizes such as 2 x 2 are commonly used. Global pooling acts on all the neurons of the feature map. There are two common types of pooling in popular use: max and average. *Max pooling* uses the maximum value of each local cluster of neurons in the feature map, while *average pooling* takes the average value.

Convolutional networks may include local and/or global pooling layers along with traditional convolutional layers.

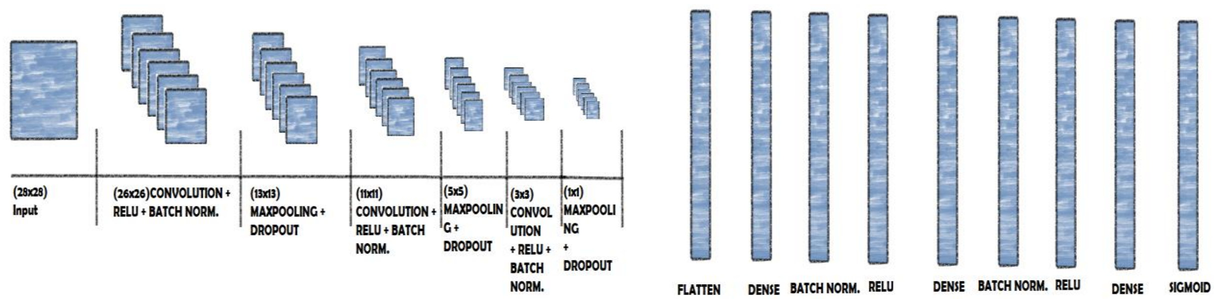
- Activation Function:
  In artificial neural networks, the activation function of a node defines the output of that node given an input or set of inputs. A standard integrated circuit can be seen as a digital network of activation functions that can be "ON" (1) or "OFF" (0), depending on input. This is similar to the behavior of the linear perceptron in neural networks. However, only *nonlinear* activation functions allow such networks to compute nontrivial problems using only a small number of nodes, and such activation functions are called nonlinearities.

Activation Function	Equation	Example	1D Graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Unit Step (Heaviside Function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Sign (signum)	$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Piece-wise Linear	$\phi(z) = \begin{cases} 0 & z \le -\frac{1}{2} \\ z + \frac{1}{2} & -\frac{1}{2} \le z \le \frac{1}{2} \\ 1 & z \ge \frac{1}{2} \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multilayer NN	
Hyperbolic Tangent (tanh)	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multilayer NN, RNNs	
ReLU	$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	Multilayer NN, CNNs	

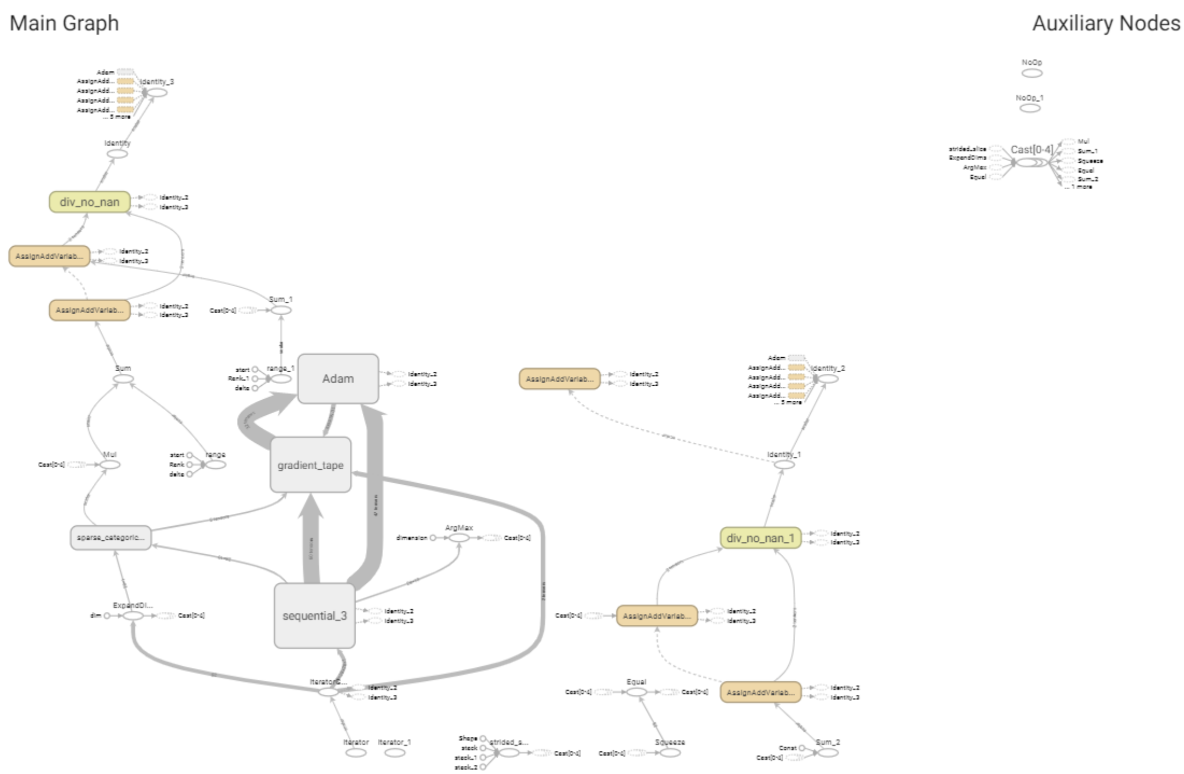
Proposed Model

```
3-conv-256-nodes-2-dense-1619596652
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 26, 26, 256)       2560
_________________________________________________________________
activation_18 (Activation)   (None, 26, 26, 256)       0
_________________________________________________________________
batch_normalization_15 (Batc (None, 26, 26, 256)       1024
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 13, 13, 256)       0
_________________________________________________________________
dropout_9 (Dropout)          (None, 13, 13, 256)       0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 11, 11, 256)       590080
_________________________________________________________________
activation_19 (Activation)   (None, 11, 11, 256)       0
_________________________________________________________________
batch_normalization_16 (Batc (None, 11, 11, 256)       1024
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 5, 5, 256)         0
_________________________________________________________________
dropout_10 (Dropout)         (None, 5, 5, 256)         0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 3, 3, 256)         590080
_________________________________________________________________
activation_20 (Activation)   (None, 3, 3, 256)         0
_________________________________________________________________
batch_normalization_17 (Batc (None, 3, 3, 256)         1024
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 1, 1, 256)         0
_________________________________________________________________
dropout_11 (Dropout)         (None, 1, 1, 256)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_9 (Dense)              (None, 256)               65792
_________________________________________________________________
batch_normalization_18 (Batc (None, 256)               1024
_________________________________________________________________
activation_21 (Activation)   (None, 256)               0
_________________________________________________________________
dense_10 (Dense)             (None, 256)               65792
_________________________________________________________________
batch_normalization_19 (Batc (None, 256)               1024
_________________________________________________________________
activation_22 (Activation)   (None, 256)               0
_________________________________________________________________
dense_11 (Dense)             (None, 10)                2570
_________________________________________________________________
activation_23 (Activation)   (None, 10)                0
=================================================================
Total params: 1,321,994
Trainable params: 1,319,434
Non-trainable params: 2,560
```

Model Design



Model Graph (Using tensorboard)



Process

I have used **mnist** data for this neural network task. This dataset is consisting of 70,000 images of hand written digits. 60,000 images are used as training on the other hand 10000 are used as test set.

```
(train_imgs, train_labels), (test_imgs, test_lebels)=datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

After fetching the dataset, I have normalized them for faster calculation. I also re shaped the images before fitting them inside the model.

```
train_imgs = tf.keras.utils.normalize(train_imgs, axis = 1)
test_imgs = tf.keras.utils.normalize(test_imgs, axis = 1)
train_imgs= train_imgs.reshape(-1,28,28,1)
```

My formatted code is given inside this snap. I have constructed 4 models for finding the optimal solution. Before that I have tested with this model with different dense and convolution layers. They did not give much accuracy what these 4 models have given.

```
dense_layers = [2]
layer_sizes = [32, 64, 128, 256]
conv_layers = [3]

for dense_layer in dense_layers:
    for layer_size in layer_sizes:
        for conv_layer in conv_layers:
            NAME = "{}-conv-{}-nodes-{}-dense-{}".format(conv_layer, layer_size, dense_layer, int(time.time()))
            print(NAME)

            model = Sequential()

            model.add(Conv2D(layer_size, (3, 3), input_shape=(28,28,1)))
            model.add(Activation('relu'))
            model.add(BatchNormalization())
            model.add(MaxPooling2D(pool_size=(2, 2)))
            model.add(Dropout(0.25))



            for l in range(conv_layer-1):
                model.add(Conv2D(layer_size, (3, 3)))
                model.add(Activation('relu'))
                model.add(BatchNormalization())
                model.add(MaxPooling2D(pool_size=(2, 2)))
                model.add(Dropout(0.3))

            model.add(Flatten())

            for _ in range(dense_layer):
                model.add(Dense(layer_size))
                model.add(BatchNormalization())
                model.add(Activation('relu'))

            model.add(Dense(10))
            model.add(Activation('sigmoid'))

            model.summary()

            tensorboard = TensorBoard(log_dir="logs/{}".format(NAME))
            opt = tf.keras.optimizers.SGD(lr=0.001, momentum=0.9)
            model.compile(optimizer='adam',
                          loss='sparse_categorical_crossentropy',
                          metrics=['accuracy'],
                          )

            model.fit(train_imgs, train_labels,
                      batch_size=28,
                      epochs=10,
                      validation_split=0.3,
                      callbacks=[tensorboard])
```

Result and Discussion:

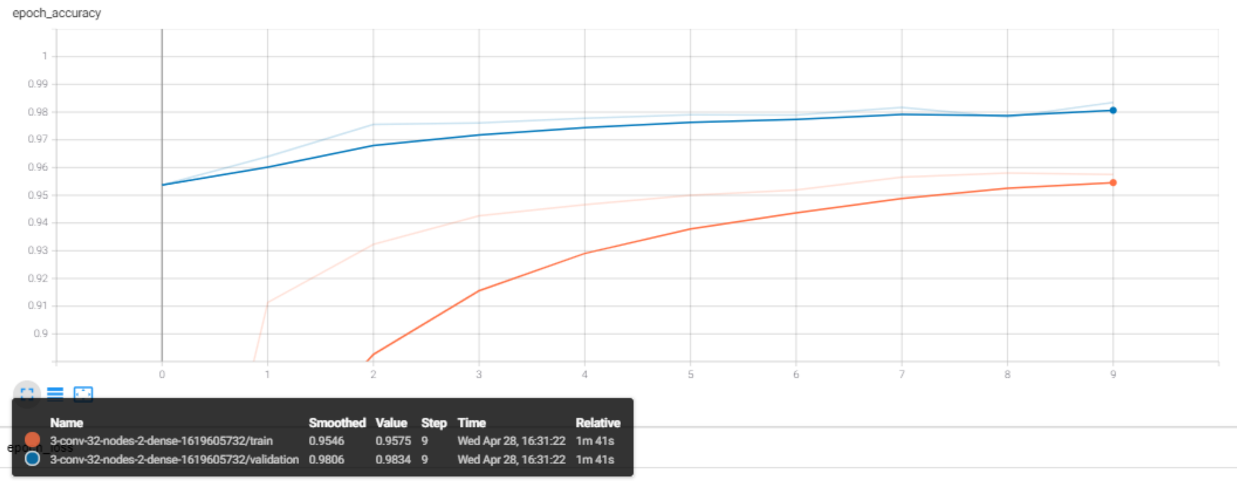


Figure: Accuracy of Model with 3 Convolution Layes with 32 Neurons and 2 Dense Layers.

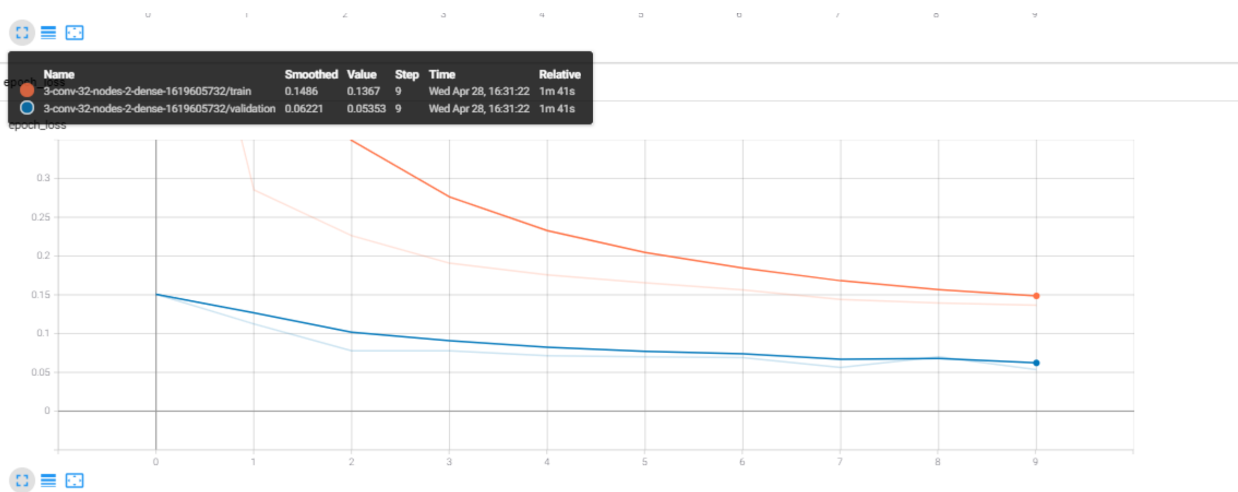


Figure: Loss of Model with 3 Convolution Layes with 32 Neurons and 2 Dense Layer

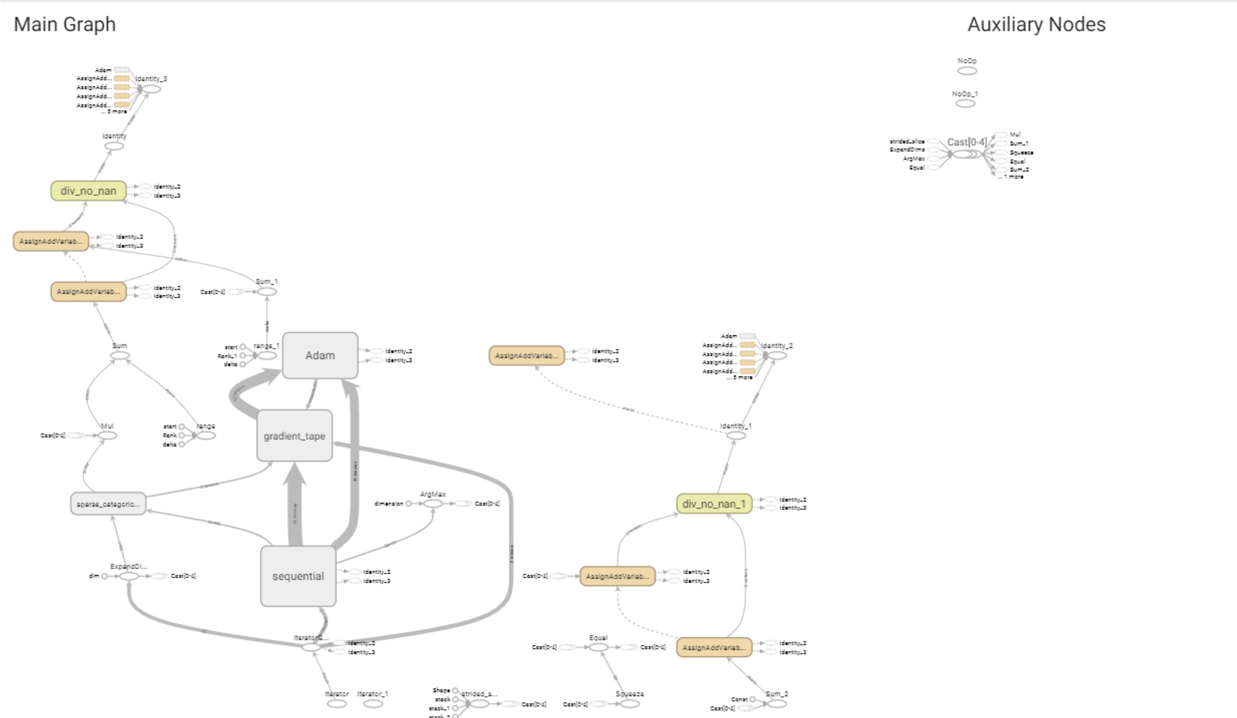


Figure: Graph of Model with 3 Convolution Layes with 32 Neurons and 2 Dense Layers.

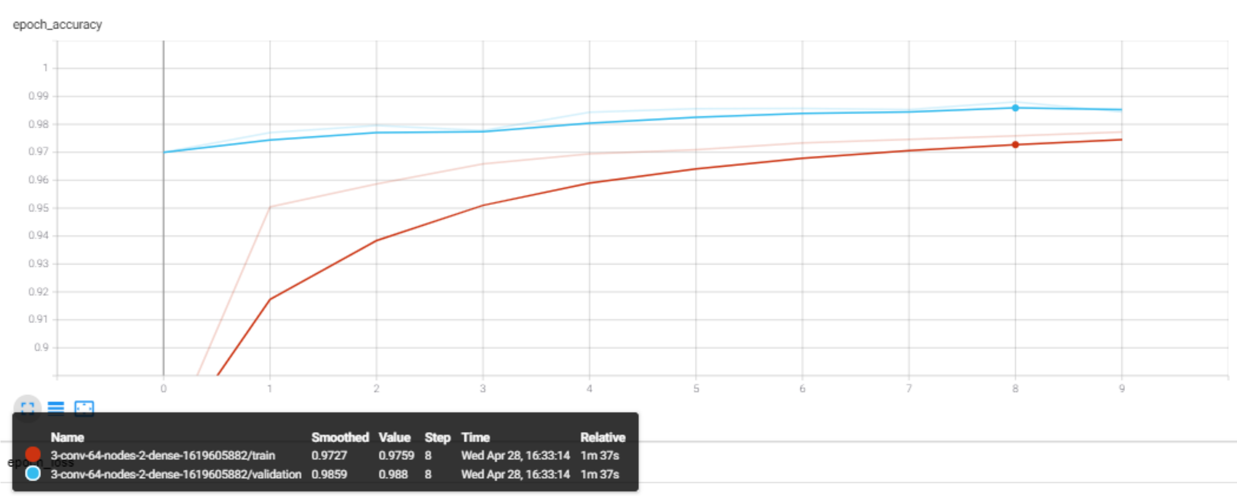


Figure: Accuracy of Model with 3 Convolution Layes with 64 Neurons and 2 Dense Layers

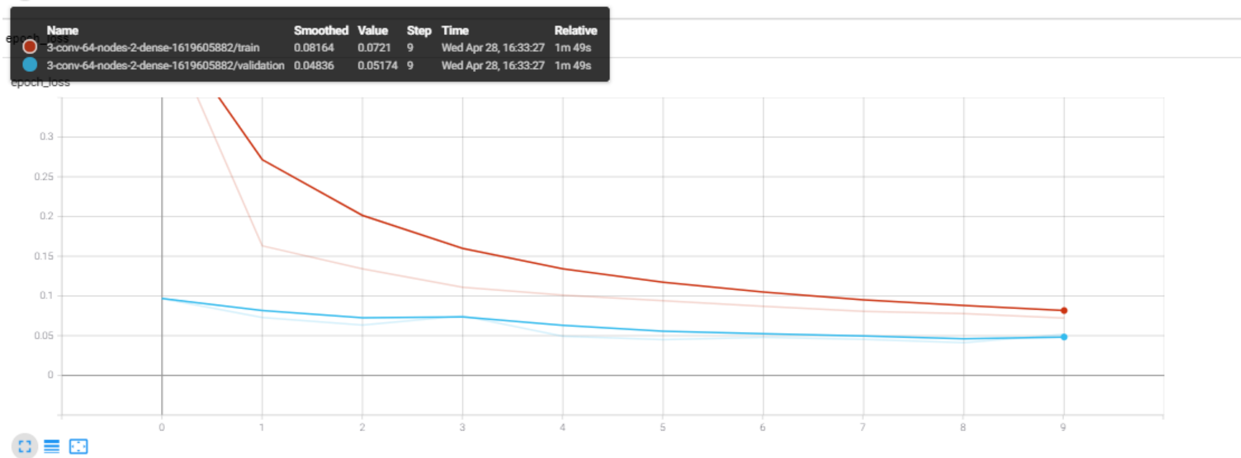


Figure: Loss of Model with 3 Convolution Layes with 64 Neurons and 2 Dense Layers

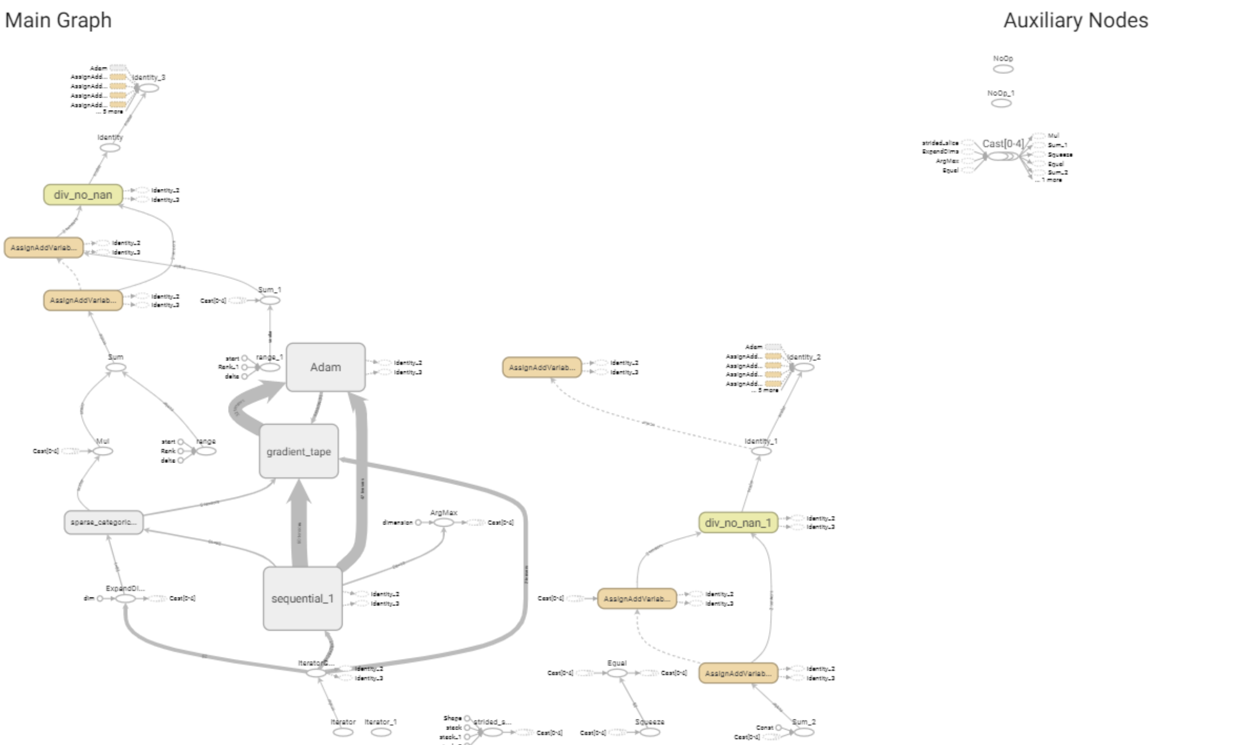


Figure: Graph of Model with 3 Convolution Layes with 64 Neurons and 2 Dense Layers

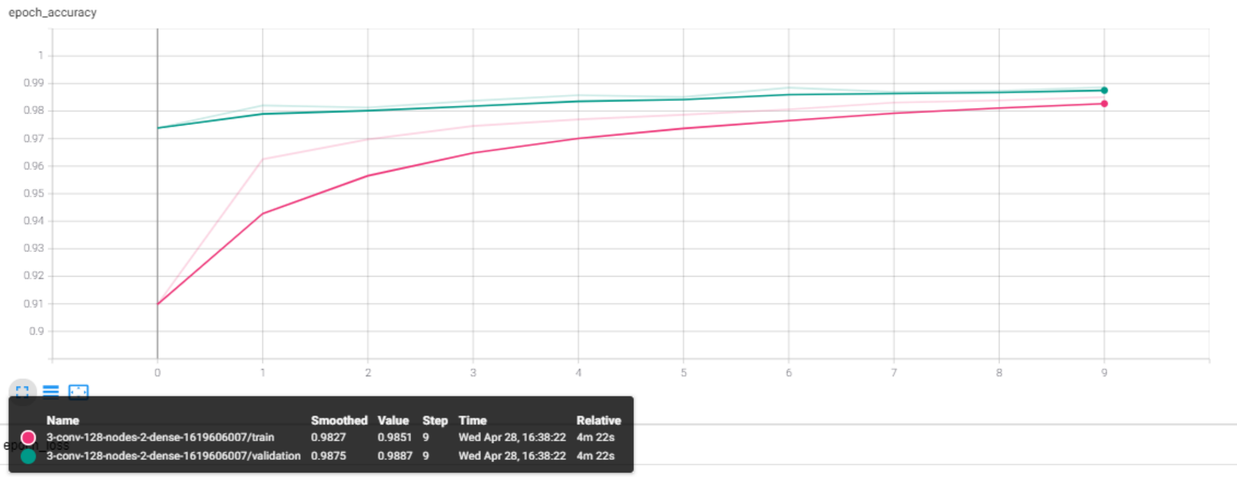


Figure: Accuracy of Model with 3 Convolution Layes with 128 Neurons and 2 Dense Layers

Figure: Loss of Model with 3 Convolution Layes with 128 Neurons and 2 Dense Layers

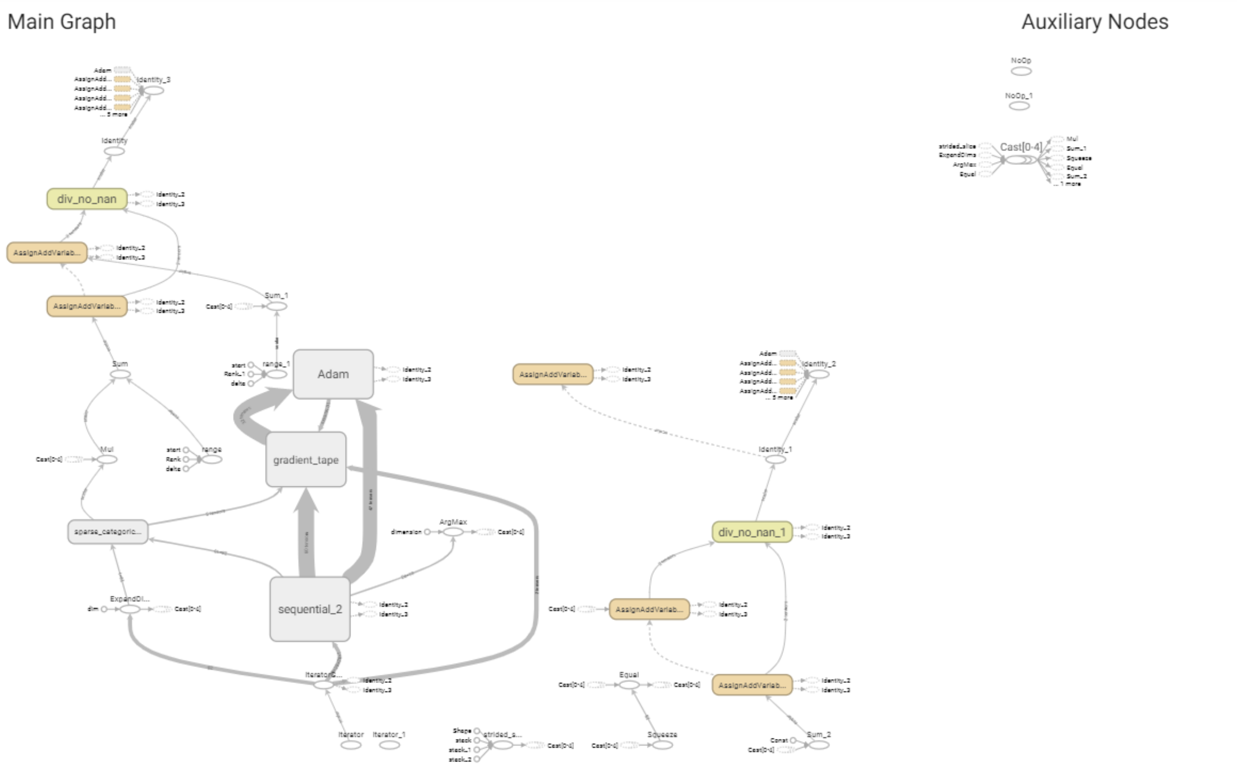


Figure: Graph of Model with 3 Convolution Layes with 128 Neurons and 2 Dense Layers

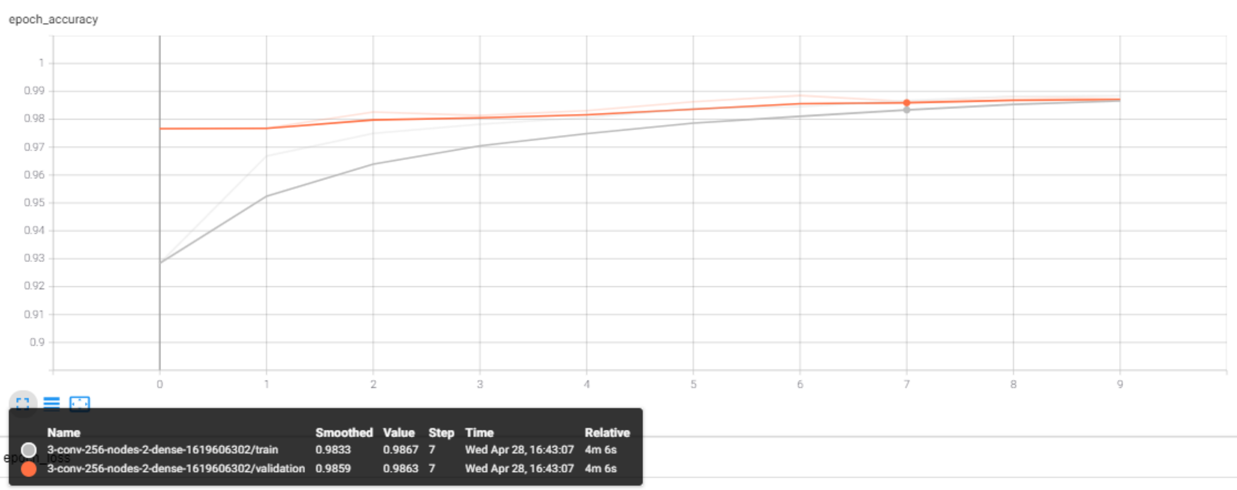


Figure: Accuracy of Model with 3 Convolution Layes with 256 Neurons and 2 Dense Layers

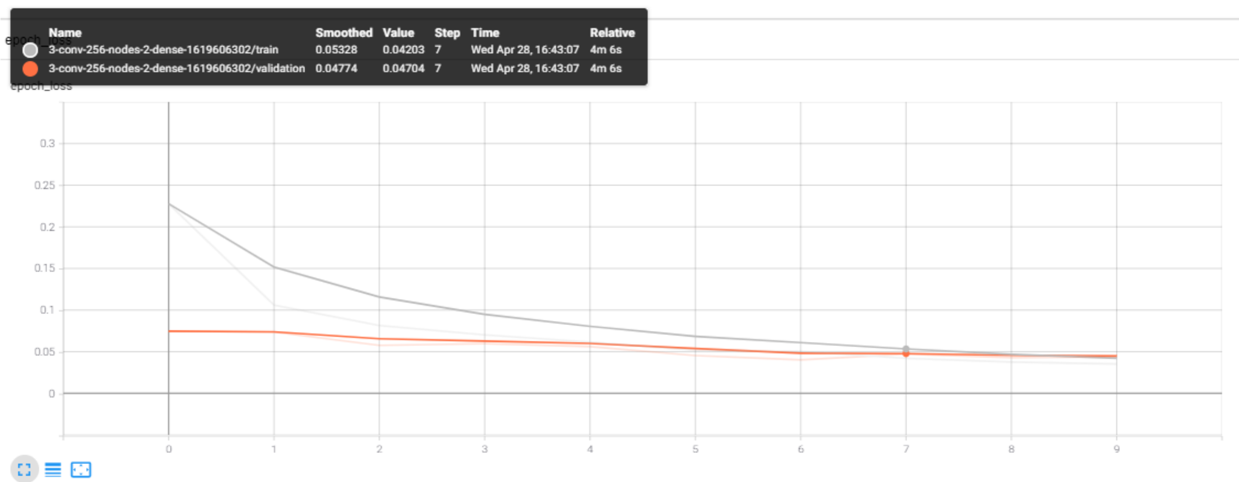


Figure: Loss of Model with 3 Convolution Layes with 256 Neurons and 2 Dense Layers

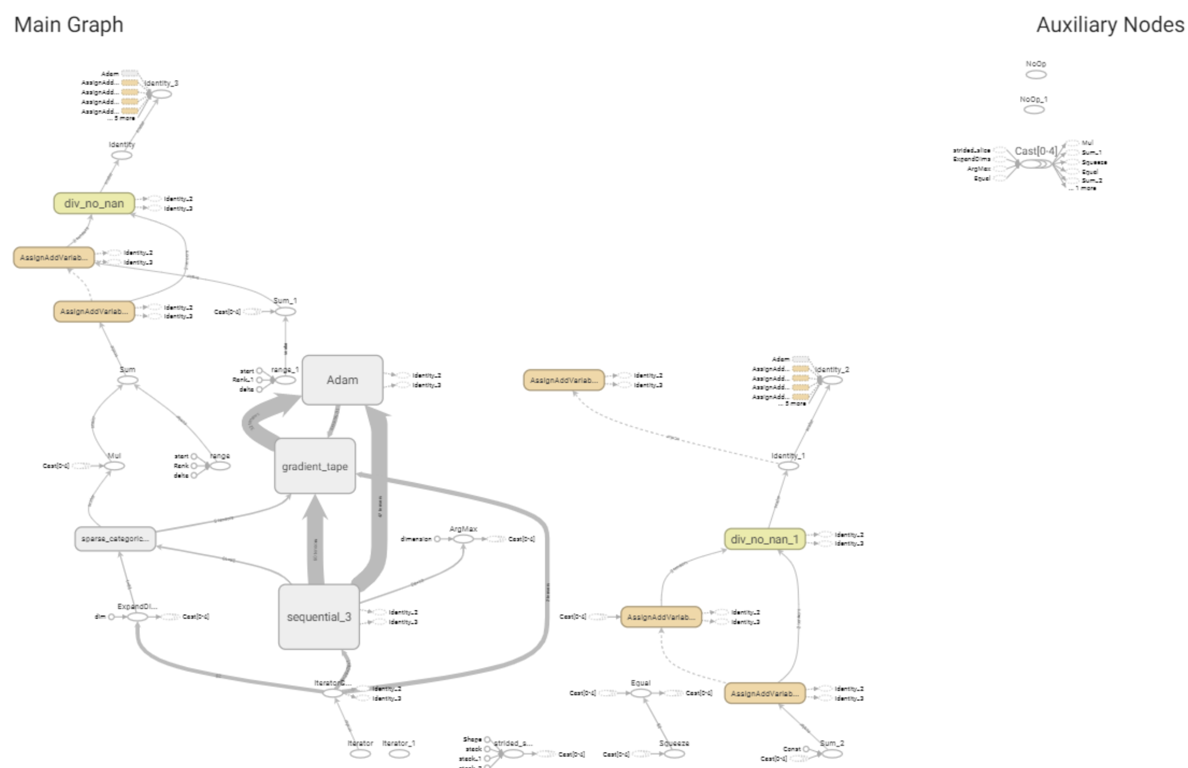


Figure: Graph of Model with 3 Convolution Layes with 256 Neurons and 2 Dense Layers

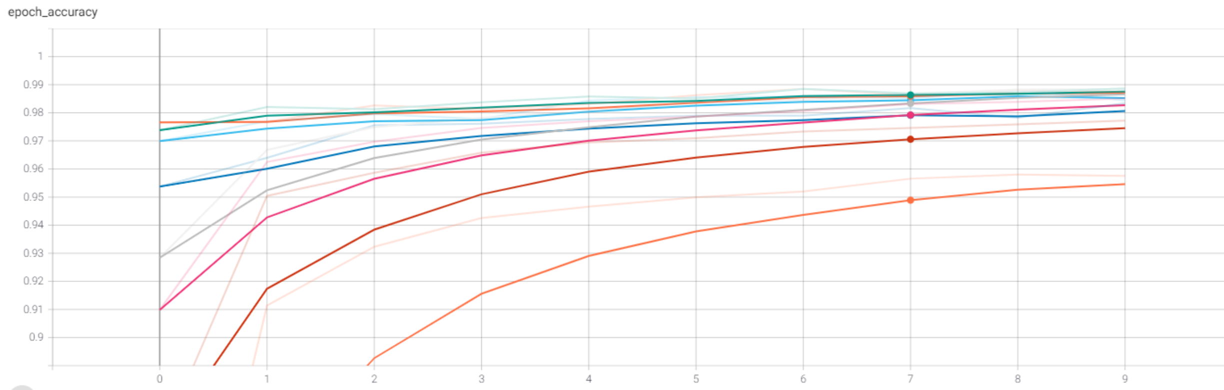


Figure: Accuracy of Models

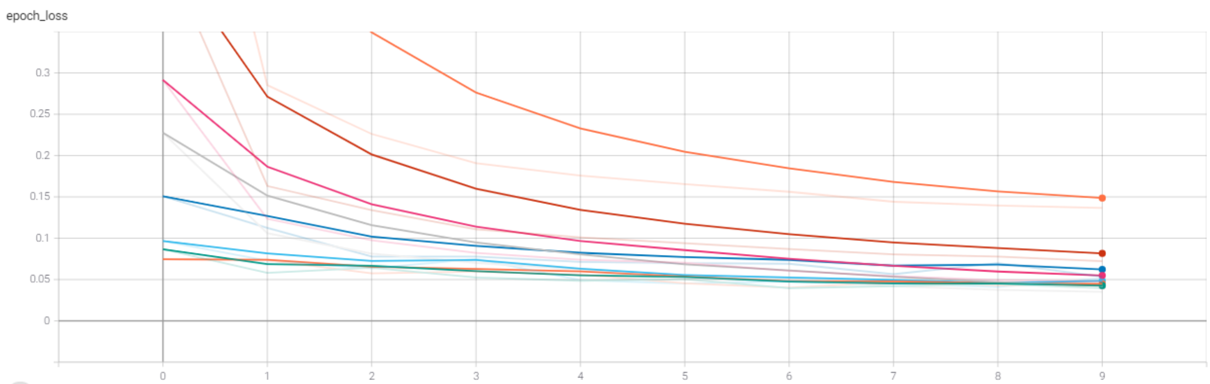


Figure: Loss of Models

```
Epoch 1/10
1500/1500 [==============================] - 38s 24ms/step - loss: 0.4173 - accuracy: 0.8678 - val_loss: 0.0746 - val_accuracy: 0.9766
Epoch 2/10
1500/1500 [==============================] - 35s 24ms/step - loss: 0.1115 - accuracy: 0.9641 - val_loss: 0.0735 - val_accuracy: 0.9768
Epoch 3/10
1500/1500 [==============================] - 35s 23ms/step - loss: 0.0791 - accuracy: 0.9755 - val_loss: 0.0575 - val_accuracy: 0.9826
Epoch 4/10
1500/1500 [==============================] - 35s 23ms/step - loss: 0.0708 - accuracy: 0.9781 - val_loss: 0.0596 - val_accuracy: 0.9813
Epoch 5/10
1500/1500 [==============================] - 35s 23ms/step - loss: 0.0637 - accuracy: 0.9794 - val_loss: 0.0561 - val_accuracy: 0.9831
Epoch 6/10
1500/1500 [==============================] - 35s 23ms/step - loss: 0.0539 - accuracy: 0.9831 - val_loss: 0.0454 - val_accuracy: 0.9863
Epoch 7/10
1500/1500 [==============================] - 35s 23ms/step - loss: 0.0487 - accuracy: 0.9849 - val_loss: 0.0402 - val_accuracy: 0.9885
Epoch 8/10
1500/1500 [==============================] - 35s 23ms/step - loss: 0.0439 - accuracy: 0.9868 - val_loss: 0.0470 - val_accuracy: 0.9863
Epoch 9/10
1500/1500 [==============================] - 35s 23ms/step - loss: 0.0362 - accuracy: 0.9890 - val_loss: 0.0429 - val_accuracy: 0.9881
Epoch 10/10
1500/1500 [==============================] - 35s 23ms/step - loss: 0.0380 - accuracy: 0.9878 - val_loss: 0.0436 - val_accuracy: 0.9874
```

Form this mentioned figure and other figures it is clear that the model with 256 Neurons out performed others. With Accuracy of 98.78% and loss of 3.8%. So that I would prefer this model for this **mnist** Dataset.