Python Implementation using Keras and Tensorflow:

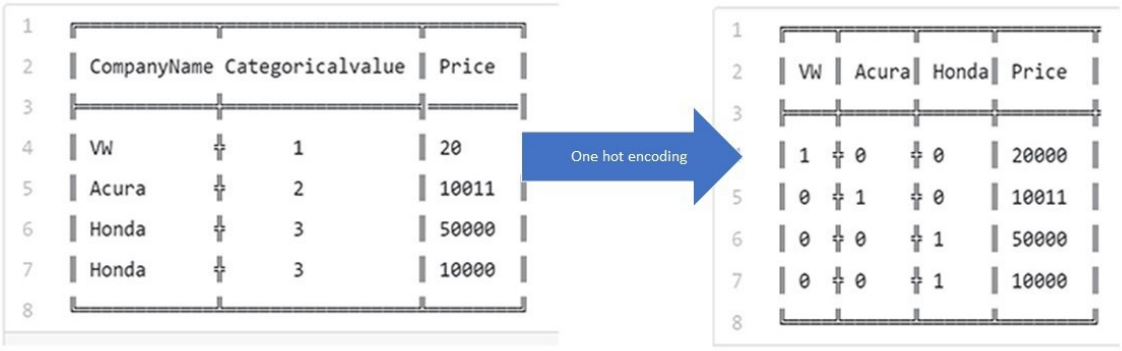
First let us just see what is one hot encoding:

Suppose you want to work with binary data and particularly classify it into the given set of classes then you use this technique. With one hot encoding what happens is the value on the class where a given input is present gets the value 1 and all other classes goes to 0. Logically only activating that class for where the input belongs and not any other classes.

In this way all the other classes gets 0 boolean value and the activated class gets the value 1.

Let us consider an example for this :

Suppose that we have three classes of cars :Volkswagon as VW, Acura and Honda and all these types of cars have categorical values as 1, 2, 3 respectively. Suppose all these cars have different costs. Even cars with the same type have different costs. How are we representing these costs in the form of one hot encoding?



So in one hot encoding we form a vector of 1s and 0s and put 1s in their categorical positions accordingly.

Now after training in the neural network supposing we want to predict the categorical class of such data their output that we get is in the form of probability vector of these classes. Suppose for a particular input we have a probability of a particular class as the highest, then the input is classified into that class.

This is a similar procedure we have applied to out code below.

Experiments:

These experiments were carried out in python. We built an Artificial Neural Network and used one hot encoding for classifying out binary data into one of the three classes. The libraries we included were :

1.) Keras with Tensorflow
2.) Matplotlib to plot the graphs
3.) And pandas along with numpy to read the data and convert it into a matrix of values.

Initially we created two types of data sets:
One was with manual splitting of training and testing data and the other was just splitting the data into input features and output lables.

First type :

```
#splitting the training and testing data on your own
x_train = data.iloc[1:451, 0:10]
x_train= x_train.values

x_train= np.append(x_train, data.iloc[-350:, 0:10].values, axis=0)

x_train.shape
# x_train.transpose().shape

# x_actual_train = x_train.transpose()
# x_actual_train.shape

y_train= data.iloc[1:451, 11:12].values
y_train= np.append(y_train, data.iloc[-350:, 11:12].values, axis=0)

# y_train=y_train.values


##creating testing data set


x_test = data.iloc[451:675, 0:10].values
y_test = data.iloc[451:675, 11:12].values


print("Shapes, of training ", x_train.shape, y_train.shape)
print("Shapes, of testing ", x_test.shape, y_test.shape)
```

```
Shapes, of training  (800, 10) (800, 1)
Shapes, of testing  (224, 10) (224, 1)
```

Second type :

```
#splitting the validation data from the fit function

x_data_full = data.iloc[1:, 0:10].values
y_data_full = data.iloc[1:, 11:12].values

print("Shapes of input and output matrix vectors : ", x_data_full.shape, y_data_full.shape)
```

```
Shapes of input and output matrix vectors :  (1023, 10) (1023, 1)
```

The second type of data is generally used for all the comparisons and validations and allowing the Keras's fit function to directly split data into the given splitting and testing ratio by itself.

First experiment:

```
#building the model :

#( you can run this block of code again and again
# to reintitialize the weights and matrices)

#building with one hidden layer and starting off with ten neurons

model_for_training = tf.keras.models.Sequential()
model_for_training.add(tf.keras.layers.Flatten())
model_for_training.add(tf.keras.layers.Dense(10, activation=tf.nn.relu))
model_for_training.add(tf.keras.layers.Dense(3, activation=tf.nn.softmax))


#specifying and training to build the model

model_for_training.compile(optimizer= 'adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])

history_of_training=model_for_training.fit(x_data_full, y_data_full, validation_split=0.20, epochs=8
0)

#setting the epochs as 80

#here the split of testing to validation is 80:20
```

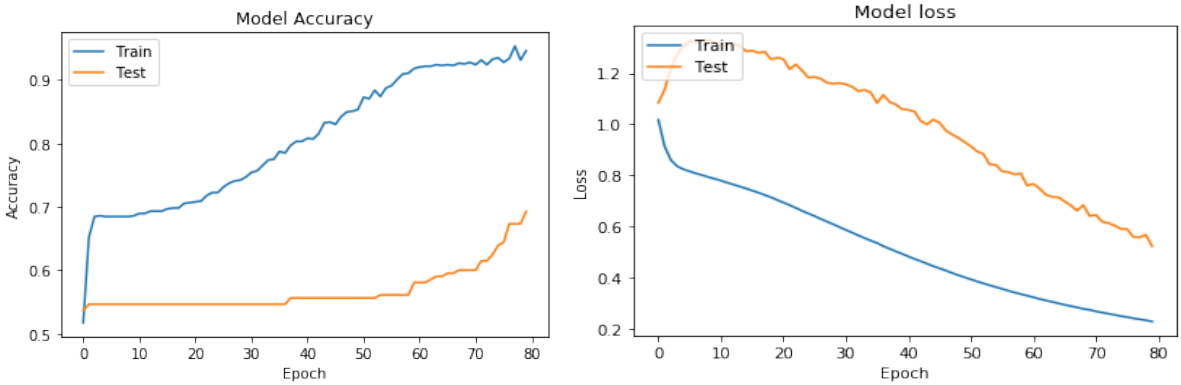
Last few lines of Results:

```
Epoch 76/80
818/818 [==============================] - 0s 30us/sample - loss: 0.2462 - acc: 0.9279 - val_los
s: 0.5903 - val_acc: 0.6439
Epoch 77/80
818/818 [==============================] - 0s 30us/sample - loss: 0.2414 - acc: 0.9340 - val_los
s: 0.5594 - val_acc: 0.6732
Epoch 78/80
818/818 [==============================] - 0s 30us/sample - loss: 0.2378 - acc: 0.9535 - val_los
s: 0.5583 - val_acc: 0.6732
Epoch 79/80
818/818 [==============================] - 0s 30us/sample - loss: 0.2337 - acc: 0.9315 - val_los
s: 0.5678 - val_acc: 0.6732
Epoch 80/80
818/818 [==============================] - 0s 30us/sample - loss: 0.2290 - acc: 0.9462 - val_los
s: 0.5238 - val_acc: 0.6927
```

Where : val_loss and val_acc are the validation loss and validation accuracy.
Visualising the experiment:

```
import matplotlib.pyplot as plt
```

```
#plotting graphs:
#import matplotlib as plt

#plotting training and validation accuracy values
plt.plot(history_of_training.history['acc'])
plt.plot(history_of_training.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()


#plotting training and validation loss values

plt.plot(history_of_training.history['loss'])
plt.plot(history_of_training.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
```



Second Experiment:

```
#now building the model with 15 neurons in the hidden layer and increasing the epochs to upto 90


model_for_training = tf.keras.models.Sequential()
model_for_training.add(tf.keras.layers.Flatten())
model_for_training.add(tf.keras.layers.Dense(15, activation=tf.nn.relu))
model_for_training.add(tf.keras.layers.Dense(3, activation=tf.nn.softmax))


#compiling the model by increamenting the epochs to 90
```

```
#specifying and training to build the model

model_for_training.compile(optimizer= 'adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])

history_of_training=model_for_training.fit(x_data_full, y_data_full, validation_split=0.20, epochs=9
0)
```

#here the split of testing to validation is 80:20

Last few lines of Results:

```
Epoch 86/90
818/818 [==============================] - 0s 38us/sample - loss: 0.1328 - acc: 0.9890 - val_los
s: 0.3252 - val_acc: 0.7561
Epoch 87/90
818/818 [==============================] - 0s 36us/sample - loss: 0.1301 - acc: 0.9963 - val_los
s: 0.2995 - val_acc: 0.7805
Epoch 88/90
818/818 [==============================] - 0s 37us/sample - loss: 0.1283 - acc: 0.9927 - val_los
s: 0.2820 - val_acc: 0.8293
Epoch 89/90
818/818 [==============================] - 0s 38us/sample - loss: 0.1254 - acc: 0.9988 - val_los
s: 0.2957 - val_acc: 0.7854
Epoch 90/90
818/818 [==============================] - 0s 38us/sample - loss: 0.1230 - acc: 0.9963 - val_los
s: 0.3140 - val_acc: 0.7659
```

Here accuracy : 99.6% loss is or error is 0.12.

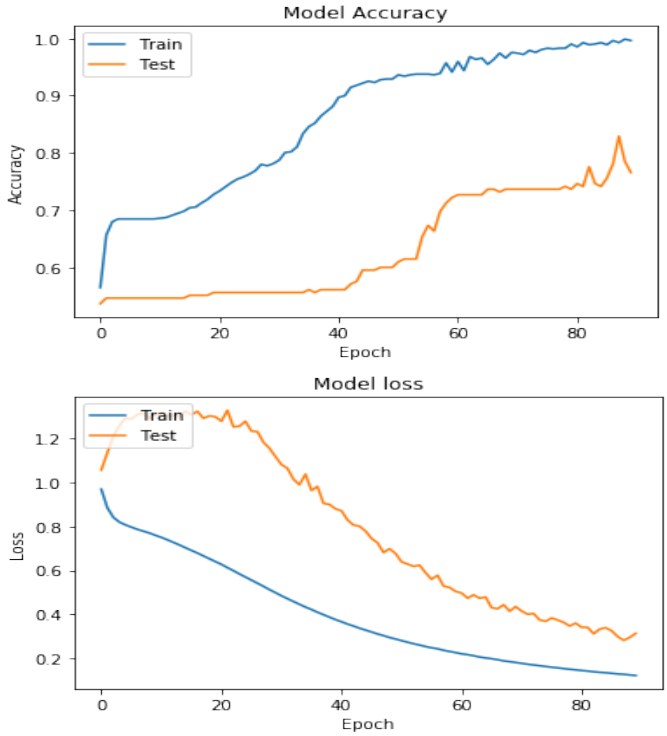
Validation loss is 0.31 and Validation Accuracy is : 76 percent.

Visualising the results:

```
## visualizing the the graph plots ##

#plotting training and validation accuracy values
plt.plot(history_of_training.history['acc'])
plt.plot(history_of_training.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

#plotting training and validation loss values

```
plt.plot(history_of_training.history['loss'])
plt.plot(history_of_training.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
```



Experiment 3:

 # now finally incrementing the epochs to about 198 and neurons 25 checking out the visualisation

```
model_for_training = tf.keras.models.Sequential()
model_for_training.add(tf.keras.layers.Flatten())
model_for_training.add(tf.keras.layers.Dense(25, activation=tf.nn.relu))
model_for_training.add(tf.keras.layers.Dense(3, activation=tf.nn.softmax))
```

#compiling the model by increamenting the epochs to 98

```
#specifying and training to build the model

model_for_training.compile(optimizer= 'adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])

history_of_training=model_for_training.fit(x_data_full, y_data_full, validation_split=0.20, epochs=1
98)
```

Last few lines of the Results:

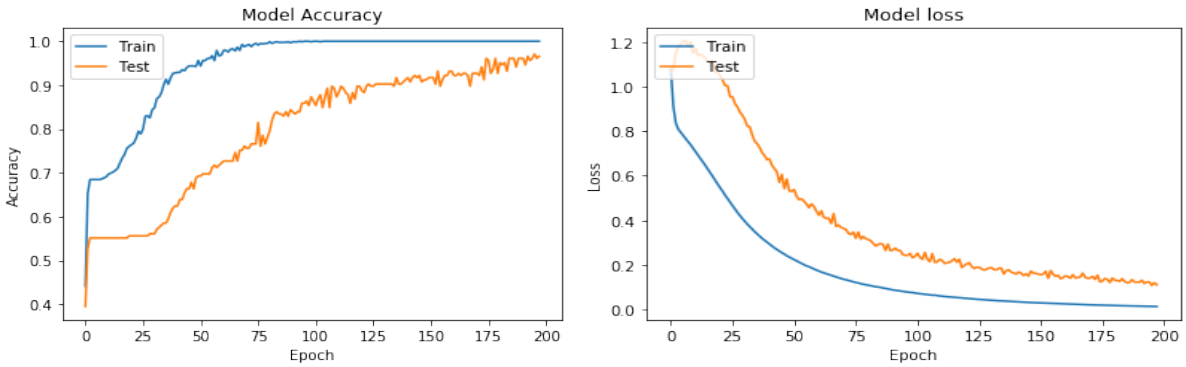
```
Epoch 194/198
818/818 [==============================] - 0s 81us/sample - loss: 2.3305e-04 - acc: 0.97 - val_l
oss: 0.1161 - val_acc: 0.9463
Epoch 195/198
818/818 [==============================] - 0s 97us/sample - loss: 2.3276e-04 - acc: 0.97 - val_l
oss: 0.1284 - val_acc: 0.9366
Epoch 196/198
818/818 [==============================] - 0s 83us/sample - loss: 2.2288e-04 - acc: 0.97 - val_l
oss: 0.1299 - val_acc: 0.9366
Epoch 197/198
818/818 [==============================] - 0s 85us/sample - loss: 2.1949e-04 - acc: 0.99160 - v
al_loss: 0.1271 - val_acc: 0.9366
Epoch 198/198
818/818 [==============================] - 0s 79us/sample - loss: 2.1648e-04 - acc: 0.99160 - v
al_loss: 0.1196 - val_acc: 0.9415
```

Here training accuracy: 99.26%, training loss : 0.0002,

Validation accuracy: 94.15%, Validation loss: 0.111

```
## visualizing the the graph plots ##

#plotting training and validation accuracy values
plt.plot(history_of_training.history['acc'])
plt.plot(history_of_training.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()


#plotting training and validation loss values

plt.plot(history_of_training.history['loss'])
plt.plot(history_of_training.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
```

plt.legend(['Train', 'Test'], loc='upper left')



Experiment 4:

#building a multi layered perceptron with first hidden layer : 25 neurons second hidden layer 35 neur ons

```
model_for_training = tf.keras.models.Sequential()
model_for_training.add(tf.keras.layers.Flatten())
model_for_training.add(tf.keras.layers.Dense(25, activation=tf.nn.relu))
model_for_training.add(tf.keras.layers.Dense(35, activation=tf.nn.relu))
model_for_training.add(tf.keras.layers.Dense(3, activation=tf.nn.softmax))
```

#compiling the model by increamenting the epochs to 198

#specifying and training to build the model

```
model_for_training.compile(optimizer= 'adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])

history_of_training=model_for_training.fit(x_data_full, y_data_full, validation_split=0.20, epochs=1
98)
```

o/p:

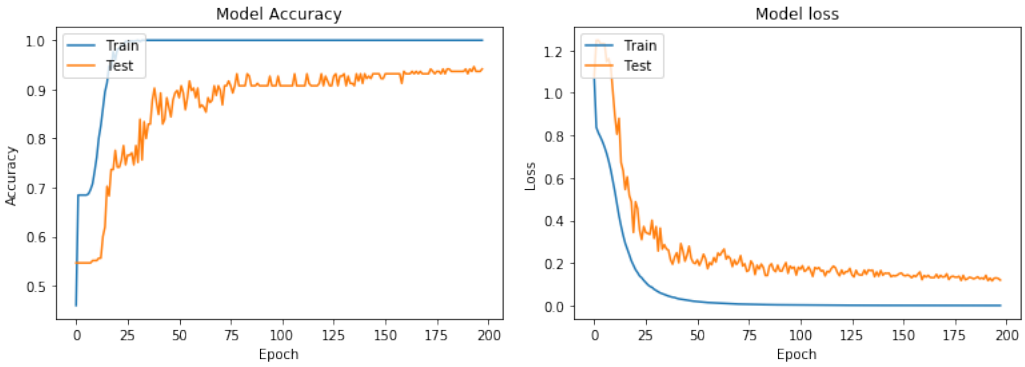
```
Epoch 194/198
818/818 [==============================] - 0s 81us/sample - loss: 2.3305e-04 - acc: .9908 - val_
loss: 0.1161 - val_acc: 0.9463
Epoch 195/198
818/818 [==============================] - 0s 97us/sample - loss: 2.3276e-04 - acc: .9958 - val_
loss: 0.1284 - val_acc: 0.9366
Epoch 196/198
818/818 [==============================] - 0s 83us/sample - loss: 2.2288e-04 - acc: .9958 - val_
loss: 0.1299 - val_acc: 0.9366
Epoch 197/198
```

818/818 [==============================] - 0s 85us/sample - loss: 2.1949e-04 - acc: .9998 - val_loss: 0.1271 - val_acc: 0.9366
Epoch 198/198
818/818 [==============================] - 0s 79us/sample - loss: 2.1648e-04 - acc: .9998 - val_loss: 0.1196 - val_acc: 0.9906

Validation_accuracy: 99.06%
Validation_loss: 0.1196

Visualization:

```
## visualizing the the graph plots ##

#plotting training and validation accuracy values
plt.plot(history_of_training.history['acc'])
plt.plot(history_of_training.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()


#plotting training and validation loss values

plt.plot(history_of_training.history['loss'])
plt.plot(history_of_training.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
```



Evaluation of all the experiments:

Exp No.	Validation Accuracy	Validation Loss	Training Accuracy	Training Loss	Hidden layers(Neurons)	Epochs
1	69.27%	0.52	94%	0.229	1(10)	80
2	76.6%	0.31	99%	0.123	1(15)	90
3	94.15%	0.11	99.16%	0.0002	1(25)	198
4	99.06%	0.1196	99.98%	0.002	2(25, 35)	198

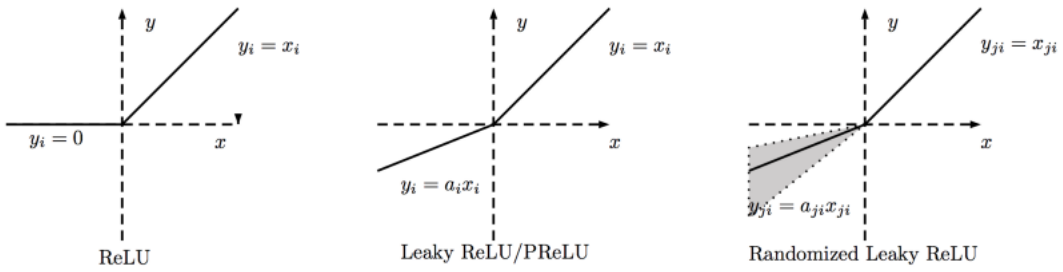
Experiment for testing the results and outputs:

This experiment was done in order to test on manually splitted data and check for its accuracy and error for the training and testing data. And also predict the unknown data that the algorithm has never seen. In this case the output that we get for the predictions is the probability distribution between three classes. The class having the highest probability distribution is the predicted class that we will be having.

This special experiment is done with a single layered 15 perceptrons and 98 epochs. We use ReLu (Rectified Linear Unit) on the first layer and softmax activation function for the output layer.

ReLu (activation function) is generally used to avoid the gradient vanishing problem. With this the network never trains itself and keeps on getting adjusted to just one value.

ReLu- Rectified Linear units : It has become very popular in the past couple of years. It was recently proved that it had *6 times improvement in convergence* from other activation functions. The graph of a ReLu is as follows:




```
# working with our custom split data
#single hidden layer with 15 neurons

model_for_training = tf.keras.models.Sequential()
model_for_training.add(tf.keras.layers.Flatten())
model_for_training.add(tf.keras.layers.Dense(15, activation=tf.nn.relu)
)
#model_for_training.add(tf.keras.layers.Dense(5, activation=tf.nn.relu)
)
```

```
model_for_training.add(tf.keras.layers.Dense(3, activation=tf.nn.softma
x))
```

```
model_for_training.compile(optimizer= 'adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history_of_training=model_for_training.fit(x_train, y_train, epochs=98)
```

Last few lines of results:

```
Epoch 95/98
800/800 [==============================] - 0s 49us/sample - loss: 0.125
7 - acc: 0.9962
Epoch 96/98
800/800 [==============================] - 0s 44us/sample - loss: 0.123
3 - acc: 0.9962
Epoch 97/98
800/800 [==============================] - 0s 42us/sample - loss: 0.121
7 - acc: 0.9962
Epoch 98/98
800/800 [==============================] - 0s 46us/sample - loss: 0.118
8 - acc: 0.9975
```

Now we do the predictions for the testing data:

```
predictions = model_for_training.predict(x_test)
print("The predictions for the test matrix is as follows ! ")
predictions
```

Last few lines of output:

```
    [1.42024388e-03, 9.62157547e-01, 3.64222378e-02],
       [2.19982485e-05, 6.19970143e-01, 3.80007803e-01],
       [4.82479408e-02, 9.50352848e-01, 1.39918609e-03],
       [1.56404148e-03, 9.82586265e-01, 1.58496182e-02],
       [1.92731875e-03, 9.77278173e-01, 2.07945686e-02],
       [3.59663718e-05, 7.23713756e-01, 2.76250333e-01],
       [1.48988573e-03, 9.74905014e-01, 2.36051362e-02],
       [2.13602434e-05, 6.42776668e-01, 3.57202023e-01],
       [4.36497721e-05, 7.47431159e-01, 2.52525240e-01],
       [9.54495576e-08, 8.05952623e-02, 9.19404566e-01],
       [6.94394767e-01, 3.05596262e-01, 8.97603240e-06],
       [3.94176468e-02, 9.59647417e-01, 9.34938260e-04],
       [5.10180853e-02, 9.48016226e-01, 9.65652580e-04]], dtype=float32
)
```

As you can see these are lists of lists of numpy arrays and it contains three values in each inner most list. These values are basically probability distribution for each class. The one with the highest probability is the class that is considered.

Now we test this testing data with the original output labels of its own and check for the actuall loss and accuracy:

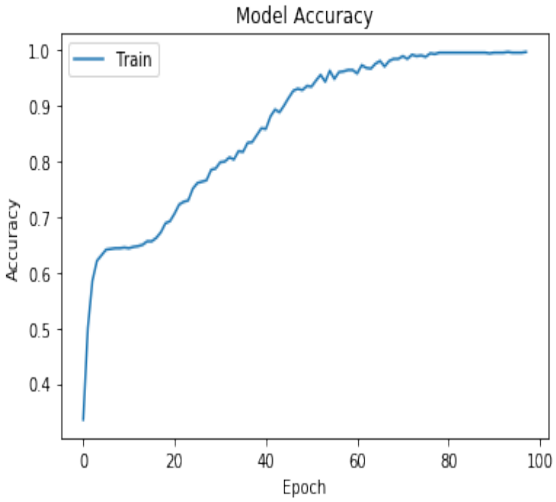
```
#evaluating the percent of accuracy for the testingd data
val_loss_for_testing, val_accuracy_for_testing = model_for_training.eva
luate(x_test, y_test)

print("Testing loss and accuracy",val_loss_for_testing, val_accuracy_fo
r_testing)
```

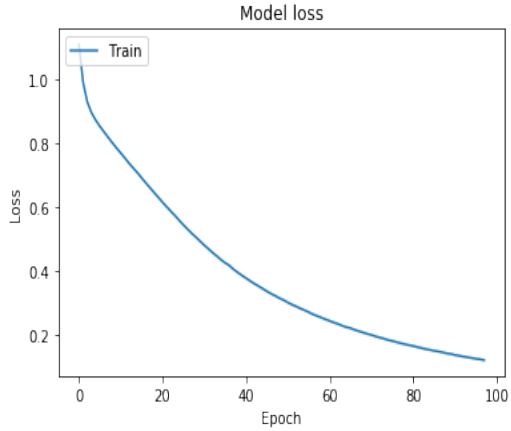
o/p:

```
224/224 [==============================] - 0s 765us/sample - loss: 0.18
26 - acc: 0.9911
Testing loas and accuracy 0.1825783678463527 0.9910714
```

As you can see our accuracy is 0.9910 ie : 99.10% which is really better.
And the loss is : 0.1825.

Let us plot the graph for the visualization with the code same as mentioned above:





Steps to run this code:

I have attached both the ipython notebook and the python files for the same. However below are the steps to run ipython notebook (ipynb). You need same libraries for python code implementation as well. But prefer using ipynb:

1. Install Keras based on tensorflow (You can use CPU Version of tensorflow)
2. Install Numpy and Pandas.
3. Install Matplotlib for the visualization.
4. Install jupyter on your PC or any machine you are working with.
5. Go to the directory where you have saved your ipython notebook from terminal (for mac or linux or from your command prompt incase of a windows machine).
6. Type jupyter notebook. Open the saved ipython notebook and start running each block of code.