**Hand Written Digits Recognition Using Neural Networks (Tensor Flow) and Tensor Board**

|  |  |  |
| --- | --- | --- |
| **INDEX** | | |
| **SNO** | **CONTENT** | **Page No** |
| **01** | **Introduction** | **3** |
| **02** | **Getting DATA** | **3** |
| **03** | **Exploring Data** | **5** |
| **04** | **Data Pre processing** | **6** |
| **05** | **Setting Tensor flow Graph** | **7** |
| **06** | **Starting Session, Batching & Training** | **9** |
| **07** | **Tensor Board Setup And Summaries & File writer:** | **11** |
| **08** | **Tensor Flow Graph** | **13** |
| **09** | **Name Scopes** | **14** |
| **10** | **Accuracy Test** | **15** |
| **11** | **Conclusion** | **15** |

**INTRODUCTION:**

**Python Deep Learning**

To make machines more intelligent, the developers are diving into machine learning and deep learning techniques. A human learns to perform a task by practicing and repeating it again and again so that it memorizes how to perform the tasks. Then the neurons in his brain automatically trigger and they can quickly perform the task they have learned. Deep learning is also very similar to this. It uses different types of neural network architectures for different types of problems.

**For example –** object recognition, image and sound classification, object detection, image segmentation, etc.

**Hand Written Digits Recognition:**

The handwritten digit recognition is the ability of computers to recognize human handwritten digits. It is a hard task for the machine because handwritten digits are not perfect and can be made with many different flavours. The handwritten digit recognition is the solution to this problem which uses the image of a digit and recognizes the digit present in the image.

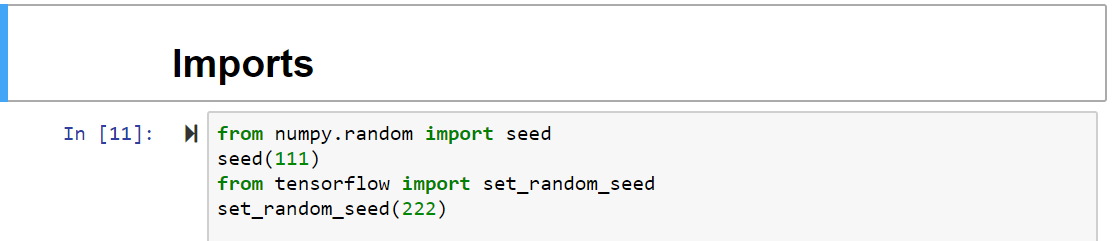
In this project I am going to predict the hand written digit images using tensor flow.

1. **GETTING DATA:**

NIST [national institute of science and technology]-we are having a MINST data.

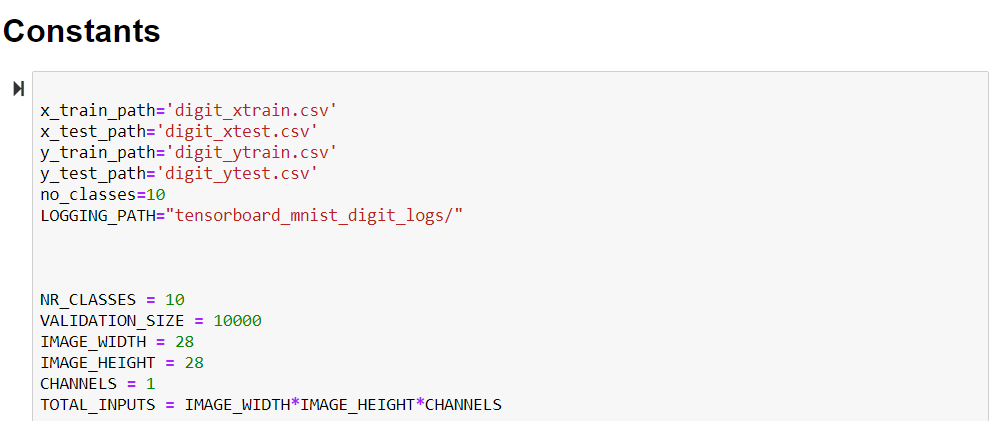
The [MNIST dataset](http://yann.lecun.com/exdb/mnist/) contains 60,000 training images of handwritten digits from zero to nine and 10,000 images for testing. So, the MNIST dataset has 10 different classes. The handwritten digits images are represented as a 28×28 matrix where each cell contains greyscale pixel value. which is single colour ranges from (0-255).

Before getting data we are going start with the below seed code.



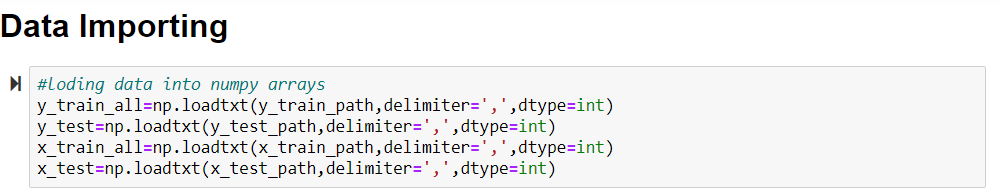
The above piece of code is very necessary when we are going to do a tensor flow because in tensor flow weights for our model are first initialized with random number. So every time we run there will be different so in order to maintain it constant we will use seed and set\_seed\_random.

Then making the constants and loading the data,



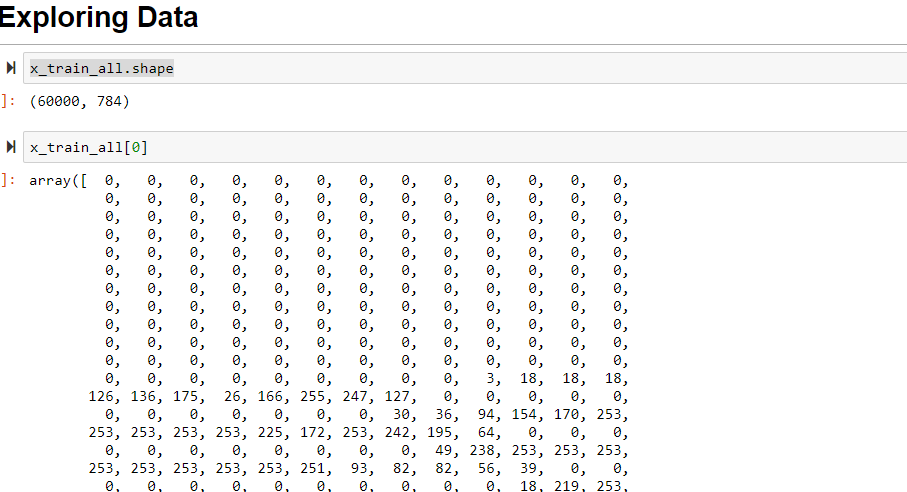
In the above code we are going to provide values to some constant which are going to use them repeatedly.

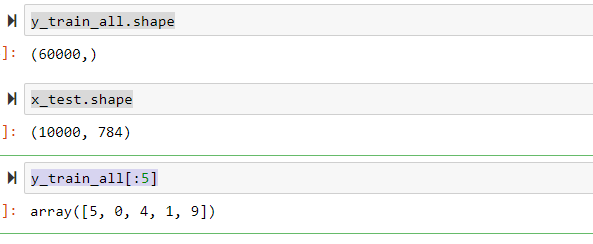
Loading data,



So we imported the data into numpy arrays, delimiter is comma since the file is CSV and data type is integer variables and we saved them in some variables.

1. **EXPLORING DATA:**

The next step is exploring data in this session we see how our data is arranged is proper or not.



1. **DATA PREPROCESSING:**

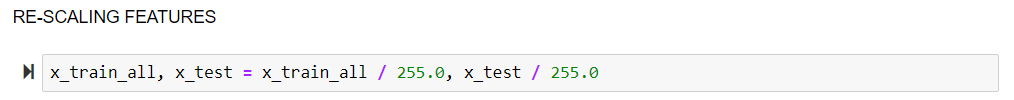
In this we are going to do three things,

**a**) RESCALING FEATURES.

**b)** ONE HOT ENCODING LABELS.

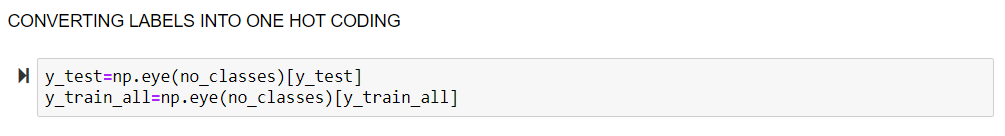
**c)** VALIDATION DATASET FROM TRAING DATASET.

Rescaling data, while performing tensor flow model the learning rate for optimizer are very small values. It helps the neural network when the features data are small between zero and one. Since our colour ranges from 0-255 if we divide with 255.0 will get values between 0-1 and convert it into float values.

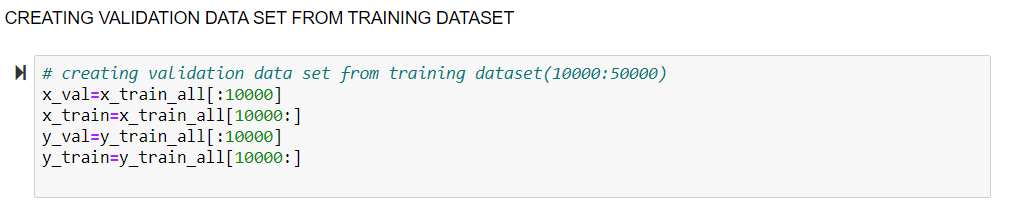


One hot encoding, in this we will convert all the label/target vales into an array consisting of one in the position of values of target and rest all are zero.

For this we will use np.eye: It returns a 2D-array with ones in the diagonal and zero elsewhere.



Creating validation dataset, since our training dataset is having 60000 rows we will Split the training dataset into a smaller training dataset and a validation dataset for the features and the labels. Create four arrays: x\_val, y\_val, x\_train, and y\_train from x\_train\_all and y\_train\_all. Use the validation size of 10,000.



1. **SETING TENSORFLOW GRAPH:**

**Tensor:**

 Tensor Flow programs use a data structure called tensor to represent all the data. Any type of data you plan to use for your model can be stored in Tensors. Simply put, a Tensor is a multi-dimensional array (0-D tensor: scalar, 1-D tensor: vector, 2-D tensor: matrix, and so on). Hence, Tensor Flow is simply referring to the flow of the Tensors in the computational graph.

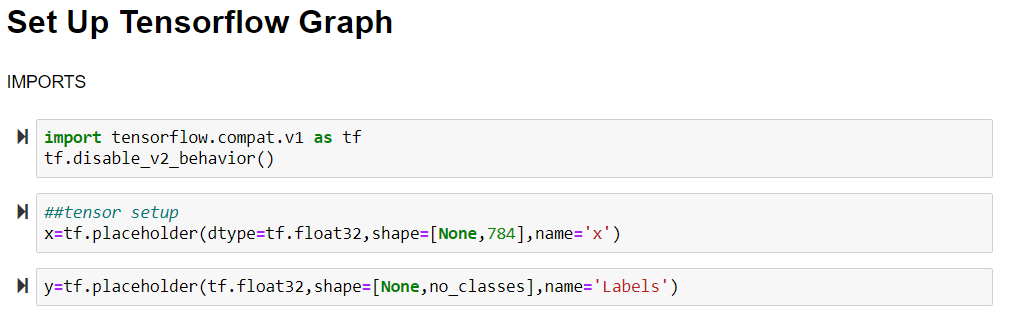
**Tensor flow Graph:**

Building the **GRAPH**, it represents the data flow of the computations.

The biggest idea about Tensor flow is that all the numerical computations are expressed as a computational graph. In other words, the backbone of any Tensor flow program is a Graph. Anything that happens in your model is represented by the computational graph. This makes it, the to go place for anything related to your model. Quoted from the Tensor Flow website, "A computational graph (or graph in short) is a series of Tensor Flow operations arranged into a graph of nodes". Basically, it means a graph is just an arrangement of nodes that represent the operations in your model.

The graph is composed of a series of nodes connected to each other by edges (from the image above). Each **node** in the graph is called **op** (short for operation). So we'll have one node for each operation; either for operations on tensors (like math operations) or generating tensors (like variables and constants). Each node takes zero or more tensors as inputs and produces a tensor as an output.

**NOTE**:  One of the biggest advantages of Tensor Flow is its visualizations of the computation graph. It’s called Tensor Board.

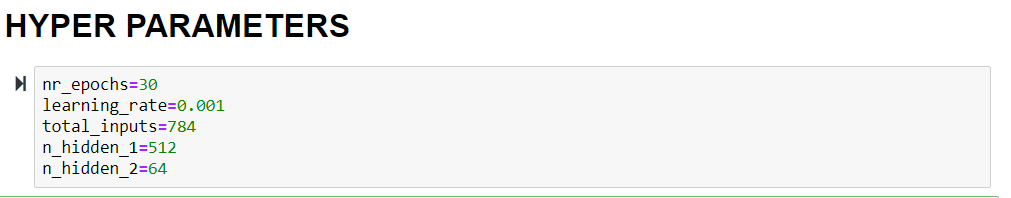


\*\*\*Inserts a placeholder for a sparse tensor that will be always fed.

\*\*\* **Important**: This sparse tensor will produce an error if evaluated. Its value must be fed using the feed\_dict optional argument to Session.run()

Setting hyper parameters,

Hyper parameters are basically determined by us like number of epochs, number of layers, learning rates, etc., which can be adjusted depending on the output.



Next step is to give some weights and bias to our neural networks layers.

So we will be creating two layers each having 512 and 64 neurons in it.

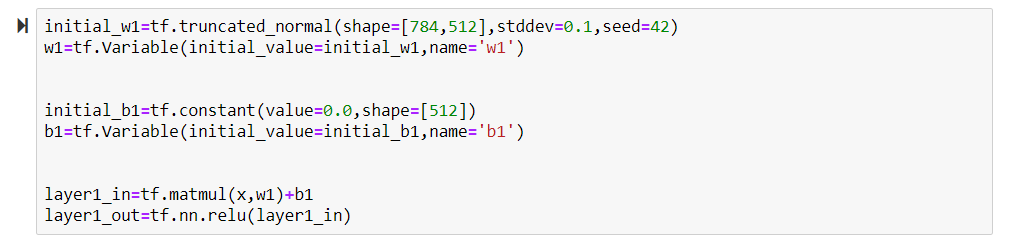
In the below code we initialized weights and bias of hidden layer one and saved those in the variables. Here we used ReLu activation function.

Truncated normal will consider the values of normal distribution and eliminates the extreme highs and lows.

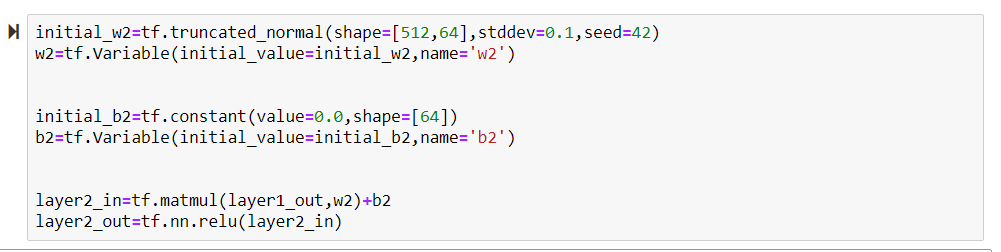
The layer one in will be input multiplied by weights plus bias which can be done by using tensor flow matrix multiplication.

Layer one is activated using relu function and out is given and this will be continued to the second hidden layer.

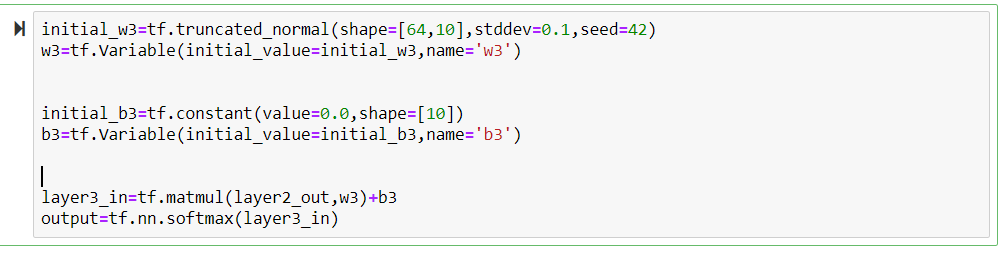
Weights, bias, layer one in and out code



Weights, bias, layer two in and out code



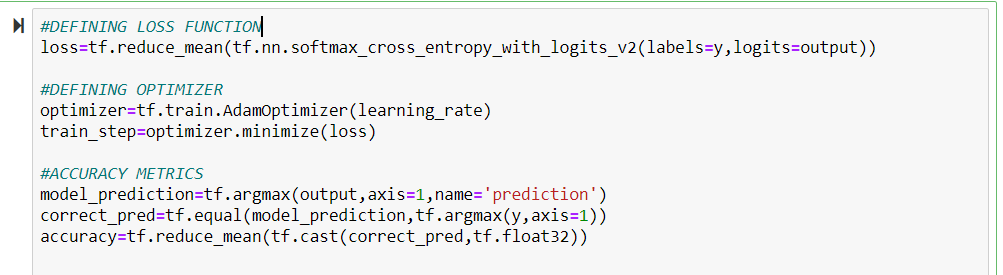
Weights, bias, output layer in and out code. Output layer activation function will be softmax.



Know we have setup our Loss function, Optimizer and Accuracy.

We are going to use cross entropy loss function, Adam optimizer.

Accuracy we will compare the output value (output layer) and ‘y’ label (output data) dataset.



**Loss function:** In machine learning you develop a model, which is a hypothesis, to predict a value given a set of input values. The model has a set of weights which you tune based on a set of training data. The training data has several pairs of input values as well as predicted values. We use a loss function to determine how far the predicted values deviate from the actual values in the training data. We change the model weights to make the loss minimum, and that is what training is all about.

**Optimizer:** Optimizers are the extended class, which include added information to train a specific model. The optimizer class is initialized with given parameters but it is important to remember that no Tensor is needed. The optimizers are used for improving speed and performance for training a specific model.

**STARTING SESSION, BATCHING & TRAINING:**

**Session:**

tf.Graph and tf.Session in TensorFlow. It’s simple:

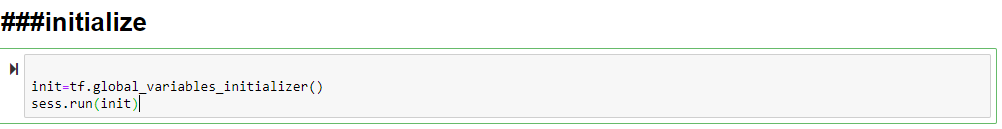
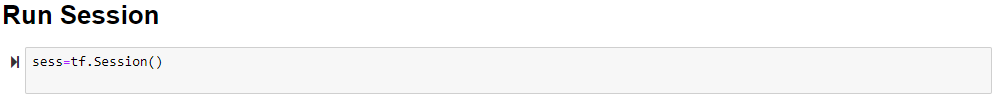
A graph defines the computation. It doesn’t compute anything, it doesn’t hold any values, it just defines the operations that you specified in your code.

A session allows executing graphs or part of graphs. It allocates resources (on one or more machines) for that and holds the actual values of intermediate results and variables.

Simply session object encapsulate environment under which all the operations and calculations are performed and executed.

**\*\*\*Note:** After the completion of session the session should be closed.

Below is the code for created session object and after creating it we have to initialize all the variables.



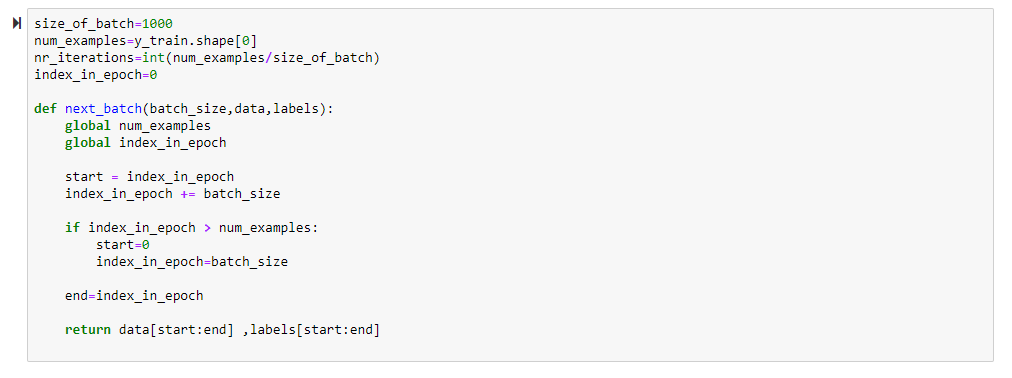
As we run the code the data flows into the session object.

**Batching:**

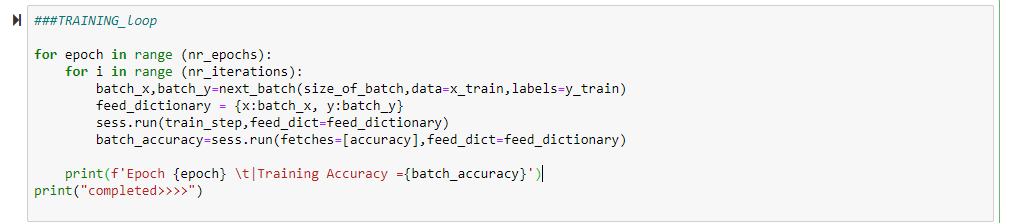
Since our dataset huge we will convert all the data set into small batches.

So that the refining will be good.

Below is the function created to convert the whole training dataset into small batch of 1000.



**Training Loop:**



In the above code snippet the sess.run accepts two arguments the one is the data to be feed into the place holders this must be feed in the dictionary type.

The other is train\_step, this is the optimizer to minimize the loss functions.

**TENSOR BOARD SETUP and SUMMARIES & FILEWRITER:**

**Tensor board:** Tensor Board is a suite of web applications for inspecting and understanding your Tensor Flow runs and graphs. Tensor Board currently supports five visualizations: scalars, images, audio, histograms, and graphs. The computations you will use in Tensor Flow for things such as training a massive deep neural network, can be fairly complex and confusing, Tensor Board will make this a lot easier to understand, debug, and optimize your Tensor Flow programs.

Simply, tensor board is the interface used to visualize the graph and other tools to understand, debug, and optimize the model.

**Summaries**:

The way tensor flow gets hold its calculations is through summaries.this will generate charts and graphs

**File writer**:

File writer will take the summaries and write them to the files created in the tensor flow setup.

Without a default writer, summary ops become silent no-ops

Code: Now in order to setup tensor board we first create a path with the required name you can see in the constants snippet

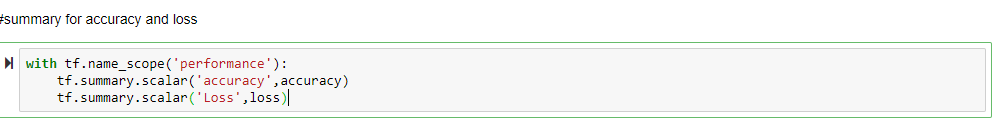
Then we will create a folder in the same directory. Below is the code for it.



So he above code will give us confirmation message if everything was done correct or gives an error message if something gone wrong.

The code will create a root folder and sub folder.

Now we will create summaries,



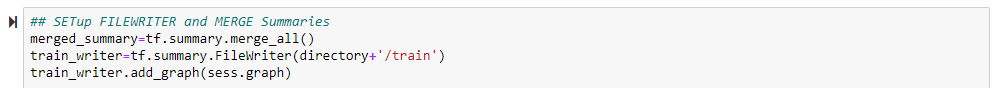
Any value that we want to visuvalize in tensorboard is tracked by the thing summary we generated two summaries one is for loss and other is for accuracy.

Now we will use our file writer to add this summaries and then we wrote our summaries to our desk

The main thing to give to the file writer is where to dump the written file. Then we will add the graph.

The tf.summary.merge\_all() will write all the summaries at the same time.

Below is the code for it.

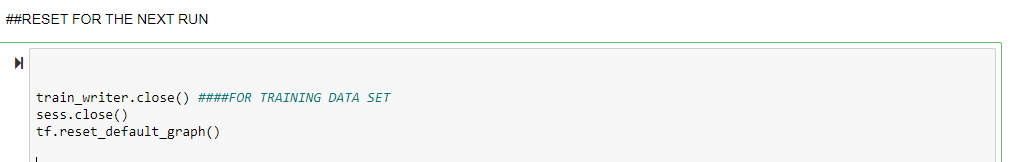


Next is we will add the summaries and file writer to our training loop



So after completion of training the session object should be closed that means reset.

For that we have to write the below code.

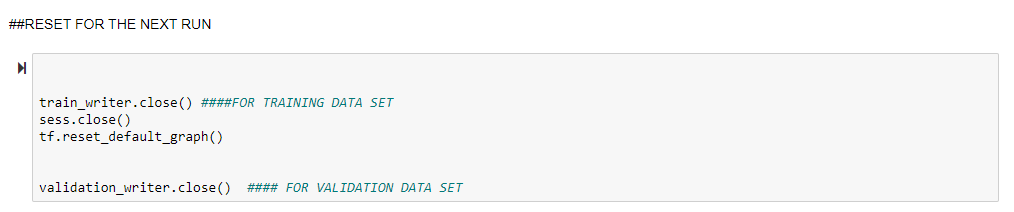
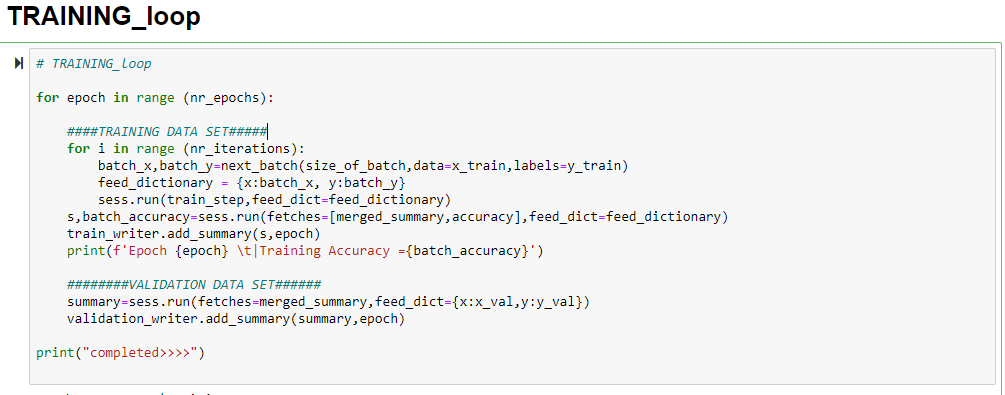


This will close the session, train\_writer, graph.

Know we have included the same thing for the validation dataset and add it to the training loop.

Below are the code snippets,





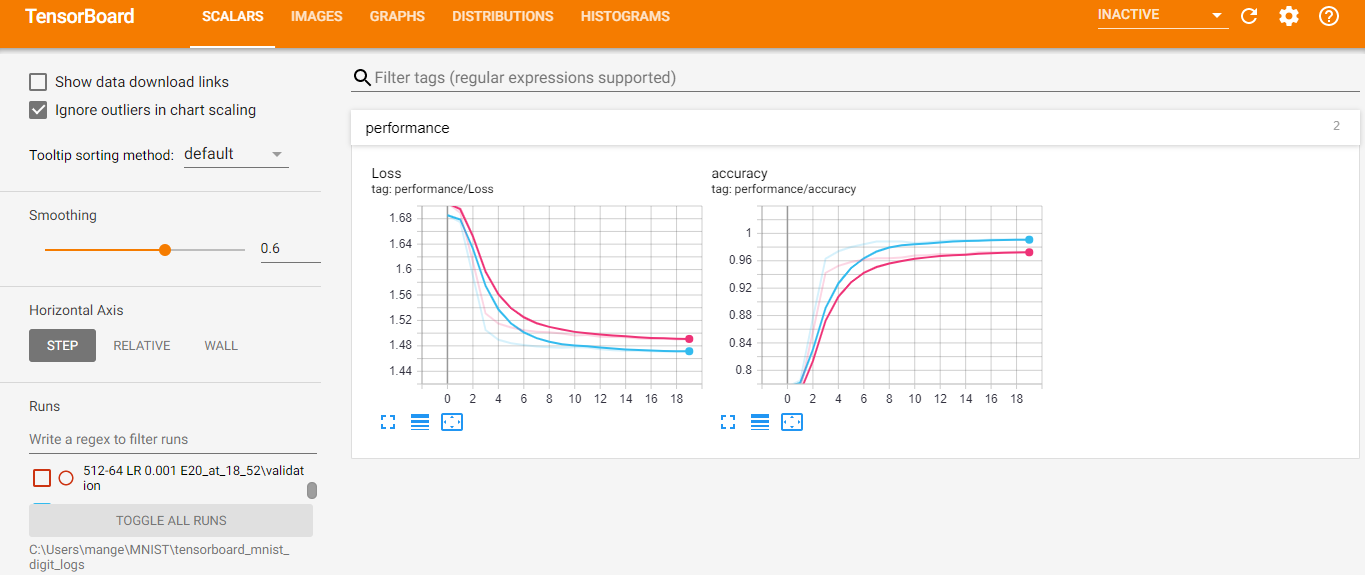
**TENSORFLOW GRAPHS:**

We can open the tensorboard by opening the anaconda prompt

And type : tensorboard –logdir=”path of the file where directory is created”.

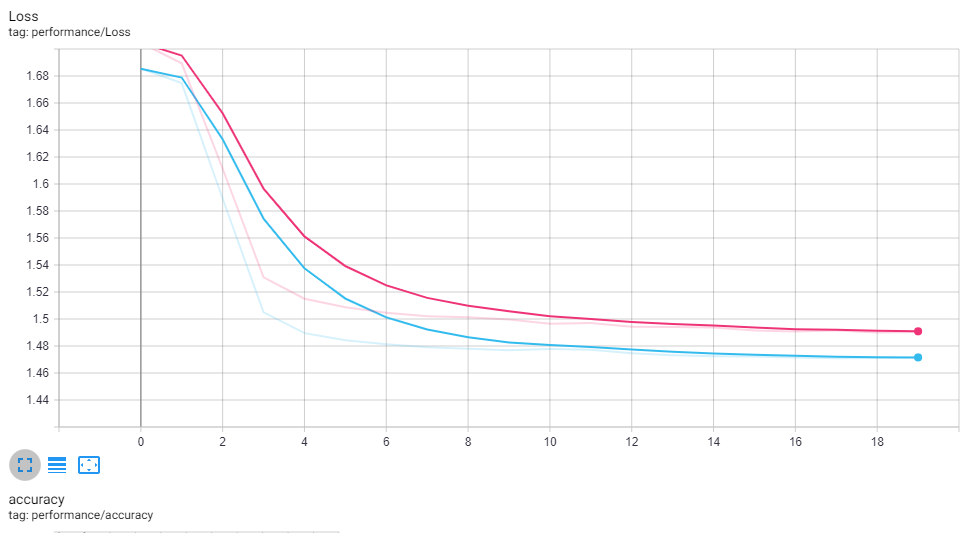
This will give us a link like--- <http://DESKTOP-5UK4THK:6006/> in my case.

Then in the we get the GUI like below one.



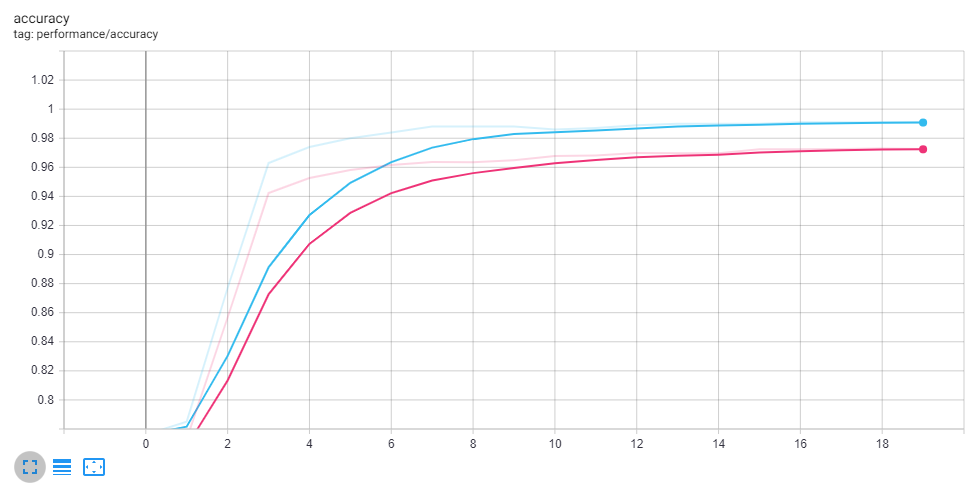
LOSS function (scalar):

x-axis represents no of epochs and y axis loss



Accuracy function (scalar):

x-axis represents no of epochs and y axis accuracy.



The blue line represents the training dataset and red line represents the validation dataset.

**NAME SCOPES:**

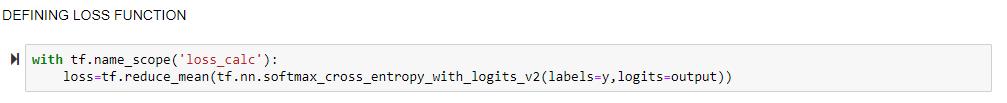
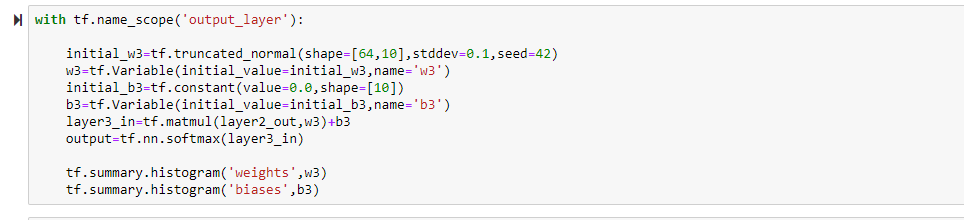
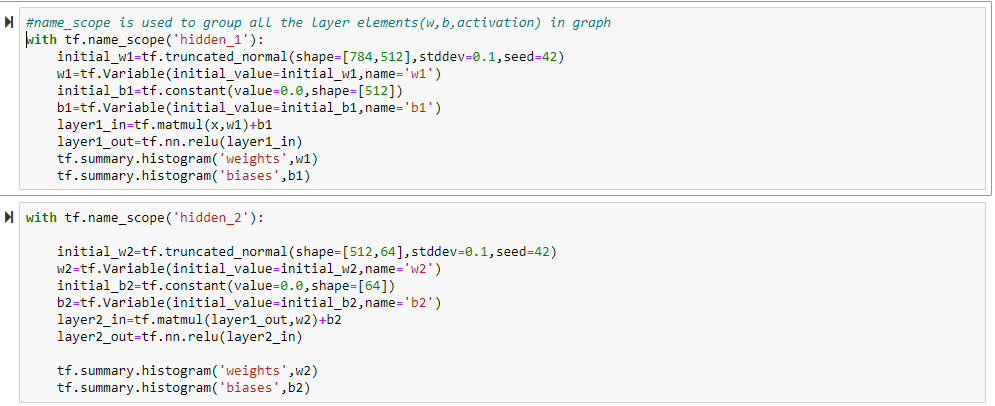
Returns a context manager that creates hierarchical names for operations.

A graph maintains a stack of name scopes. A with name\_scope(...): statement pushes a new name onto the stack for the lifetime of the context.

Tensor flow, however, uses namespaces to organize tensors/variables and operations. Tensor board group operations according namespaces they belong to, and generate a nice visual representation of the graph for you:

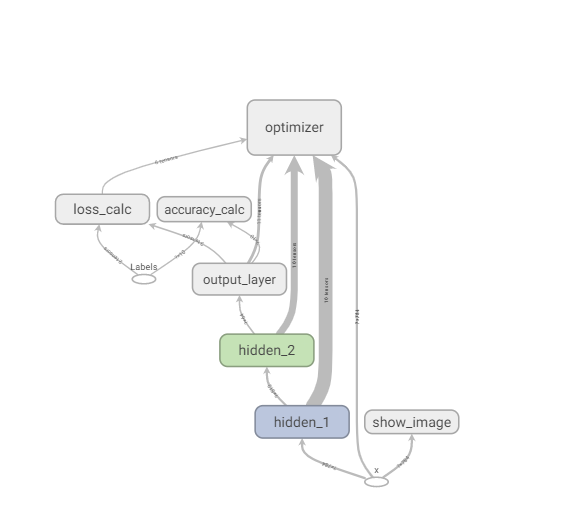
So we will do name\_scope() for required elements.

Below is the code snippet



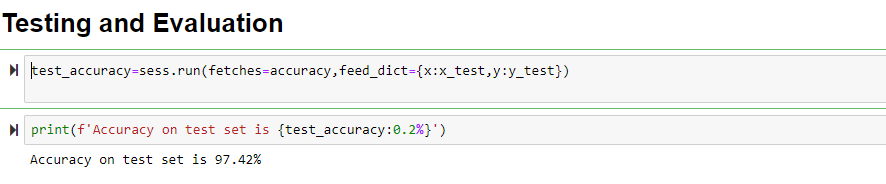
So the above code will create a graph which out much elements and dataset flow will be seen clearly as shown below.

Graphs-this is the graph representing our calculated values flow between the edge to the different nodes



**ACCURACY OF TEST:**

The test accuracy is found by the below code snippet



**Conclusion:**

In this project we successfully completed thehandwritten digit recognition with proper tensor flow by using tensor board (a visualization tool) and model is evaluated and tested and got an accuracy of 97.42%.