Implementing Feed forward Neural Network with keras and tensorflow

Assignment No. 2

Problem Statement

Implementing Feed forward Neural Network with keras and tensorflow.

1. Import the necessary packages
2. Load the training and testing data(MNIST)
3. Define the network architecture using keras
4. Train the model using SGD
5. Evaluate the network
6. Plot the training loss and accuracy

Solution Expected

Implement and train a feed-forward neural network (also known as an "MLP" for "multi-layer perceptron") on a dataset called MNIST and improve model generalisation by achieving increased accuracy and descresesd loss where model gains good confidence with the prediction.

Objectives to be achieved

1. Understand how to use Tensorflow Eager and Keras Layers to build a neural network architecture.
2. Understand how a model is trained and evaluated.
3. Identify digits from images.
4. Our main goal is to train a neural network (using Keras) to obtain > *90%* accuracy on MNIST dataset.
5. Research at least 1 technique that can be used to improve model generalization.

Methodology to be used

- Deep Learning
- Feed Forward Neural Network

Justification with Theory/Literature

Deep learning has revolutionized the world of machine learning as more and more ML practitioners have adopted deep learning networks to solve real-world problems. Compared to the more traditional ML models, deep learning networks have been shown superior performance for many applications.

The first step toward using deep learning networks is to understand the working of a simple feedforward neural network we get started with how we can build our first neural network model using **Keras** running on top of the **Tensorflow** library.

TensorFlow is an open-source platform for machine learning. Keras is the high-level application programming interface (API) of TensorFlow. Using Keras, we can rapidly develop a prototype system and test it out. This is the first in a three-part series on using TensorFlow for supervised classification tasks.

A Conceptual Diagram of the Neural Network

we'll build a supervised classification model that learns to identify digits from images. We'll use the well-known MNIST dataset to train and test our model. The MNIST dataset consists of 28-by-28 images of handwritten digits along with their corresponding labels.

We'll construct a neural network with one hidden layer to classify each digit image into its corresponding label. The figure below shows a conceptual diagram of the neural network we are about to build. The output layer consists of 10 units, where each unit corresponds to a different digit. Each unit computes a value that can be interpreted as the confidence value of its respective digit. The final classification is the digit with the maximum confidence value.

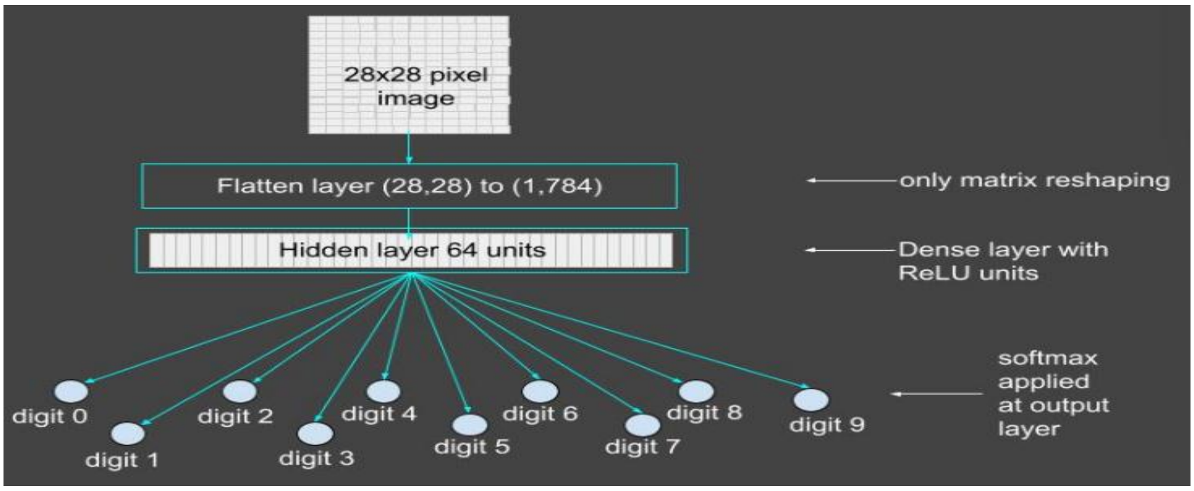


Figure 1: Conceptual diagram of the neural network. Each output unit corresponds to a digit and has its confidence value. The final classification is the digit with the maximum confidence value.

Implementing Feed forward Neural Network with keras and tensorflow

TensorFlow and Keras Libraries

If Keras and TensorFlow are not installed on your system, you can easily do so using pip or conda depending upon your Python environment.

```
pip install tensorflow
```

In the context of ML, a tensor is a multidimensional array, which in its simplest form is a scalar. Vectors and matrices are special cases of tensors. In TensorFlow, a tensor is a data structure. It is a multidimensional array composed of elements of the same type. Tensors are used to encapsulate all inputs and outputs to a deep learning network. The training dataset and each test example has to be cast as a tensor. All operations within the layers of the network are also performed on tensors.

Layers in TensorFlow?

You can build a fully connected feedforward neural network by stacking layers sequentially so that the output of one layer becomes the input to the next. In TensorFlow, layers are callable objects, which take tensors as input and generate outputs that are also tensors. Layers can contain weights and biases, which are both tuned during the training phase. We'll create a simple neural network from two layers:

1. Flatten layer
2. Dense layer

The Flatten Layer:

This layer flattens an input tensor without changing its values. Given a tensor of rank n, the Flatten layer reshapes it to a tensor of rank 2. The number of elements on the first axis remains unchanged. The elements of the input tensor's remaining axes are stacked together to form a single axis. We need this layer to create a vectorized version of each image for further input to the next layer.

The Dense Layer

The dense layer is the fully connected, feedforward layer of a neural network. It computes the weighted sum of the inputs, adds a bias, and passes the output through an activation function. We are using the ReLU activation function for this example. This function does not change any value greater than 0. The rest of the values are all set to 0.

The computations of this layer for the parameters shown in the code above are all illustrated in the figure below.

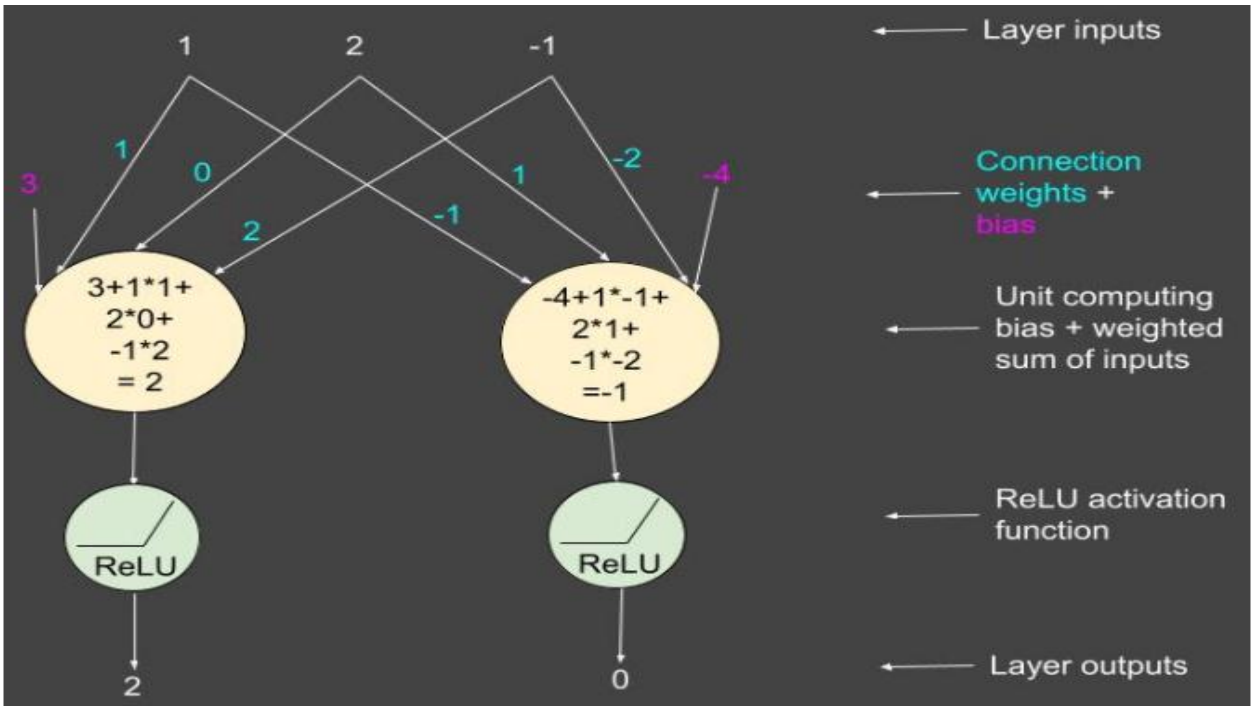


Figure 2: Hidden layer computations.

Creating a Model in TensorFlow

We are now ready to create a model of a simple neural network with one hidden layer. The simplest method is to use Sequential() with a list of all the layers you want to stack together. The code below creates a model and prints its summary. Note the use of relu as the activation function in the first dense layer and a softmax in the output layer. The softmax function normalizes all outputs to sum to 1 and ensures that they are in the range [0, 1].

```
model = tf.keras.Sequential([

   tf.keras.layers.Flatten(input_shape=(28, 28)),

   tf.keras.layers.Dense(64, activation='relu'),

   tf.keras.layers.Dense(10, activation='softmax')

])
```

Compile the Model

Next we compile the model. Here, we specify the optimizer and loss function of our model. The optimization algorithm determines how the connection weights are updated at each training step with respect to the given loss function. Because we have a multiclass classification problem, the loss function we'll use is categorical_crossentropy, coupled with the adam optimizer. You can experiment with other optimizers too. The value of the metrics argument sets the parameter to monitor and record during the training phase.

```
model.compile(optimizer='sgd',

              loss='sparse_categorical_crossentropy',

              metrics=['accuracy'])
```

Train the Neural Network

Now that the model is ready, it's time to train it. We'll load the dataset, train the model, and view the training process. Note that the outputs shown here will vary with every run of the program because of the stochastic nature of the algorithms involved.

Load the Dataset

The following code loads the training set and the test set of the MNIST data. It also prints the statistics of both sets. Because our model has 10 outputs, one for each digit, we need to convert the absolute image labels to categorical ones. The utils module in the Keras library provides the method to_categorical() for this conversion.

Train the Model

The fit() method of the model object trains the neural network. If you want to use a validation set during training, all you have to do is define the percentage of validation examples to be taken from the training set. The splitting of the training set into a train and validation set is automatically taken care of by the fit() method.

In the code below, the fit() method is called in 10 epochs.

View the Training Process

The fit() method returns a history object with detailed information regarding model training. The history attribute is a dictionary object.

To view the learning process, we can plot the accuracy and loss corresponding to different epochs for both the training and validation sets. The following code creates two graphs for these two metrics.

Implementing Feed forward Neural Network with keras and tensorflow

The predict() method

If you want to see the output of the network for one or more train/test examples, use the predict() method. The following example code prints the values of the output layer when the first test image is used as an input. It is a 10-dimensional vector of confidence values corresponding to each digit. The final classification of the image is the argmax() of this vector.

The evaluate() method

The evaluate() method computes the overall accuracy of the model on the dataset passed as an argument. The code snippet below prints the classification accuracy on both the training set and the test set.

Give Answer to following question :

1)What is Feedforward Neural Network ?

2) How the Feedforward Neural Network Works ?

3) Enlist atleast three Real time scenarios where Feedforward Neural Network is used.

4) Explain the components of Feedforward Neural Network.

5) What is costf unction in Feedforward Neural Network.

6) Define mean square error cost function.

7) What is Loss function in Feedforward Neural Network.

8) What is cross entropy loss.

9) What is kernel concept related to Feedforrward Neural Network.

10) Describe MNIST and CIFAR 10 Dataset.

11) Explain use and parameter setting related to feedforward network implementation for following libraries : SKlearn : i) LabelBinarizer (sklearn.preprocessing) ii) classification_report (sklearn.metrics) and tensorflow.keras : models , layers,optimizers,datasets ,baclend and set to respective values.

12) What is mean by flattening the dataset and why it is needed related to standard neural network implementation .

13) Explain difference between Sigmoid and Softmax activation function.

14) What is significance of optimizer in training model.

15) What is Epochs in fit command in training .

Implementing Feed forward Neural Network with keras and tensorflow

Code Snippets

Importing Libraries

```
[1]  #import necessary libraries
     import tensorflow as tf
     from tensorflow import keras
```

```
[2]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import random
     %matplotlib inline
```

Loading & Preparing the data

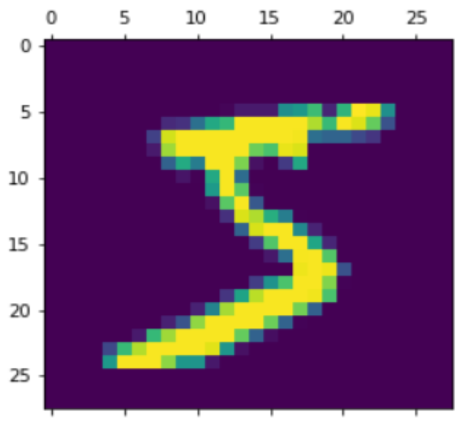
MNIST stands for Modified National Institute of Standards and Technology dataset. It is a dataset of 70,000 handwritten images.Each image is of 28*28 pixel i.e about 784 features. Each feature represent only one pixel intensity i.e from 0(white) to 255(black). This dataset is further devided into 60000 training and 10000 testing images.

```
#import dataset and split into train and test
mnist = tf.keras.datasets.mnist
(x_train,y_train),(x_test,y_test) = mnist.load_data()
```

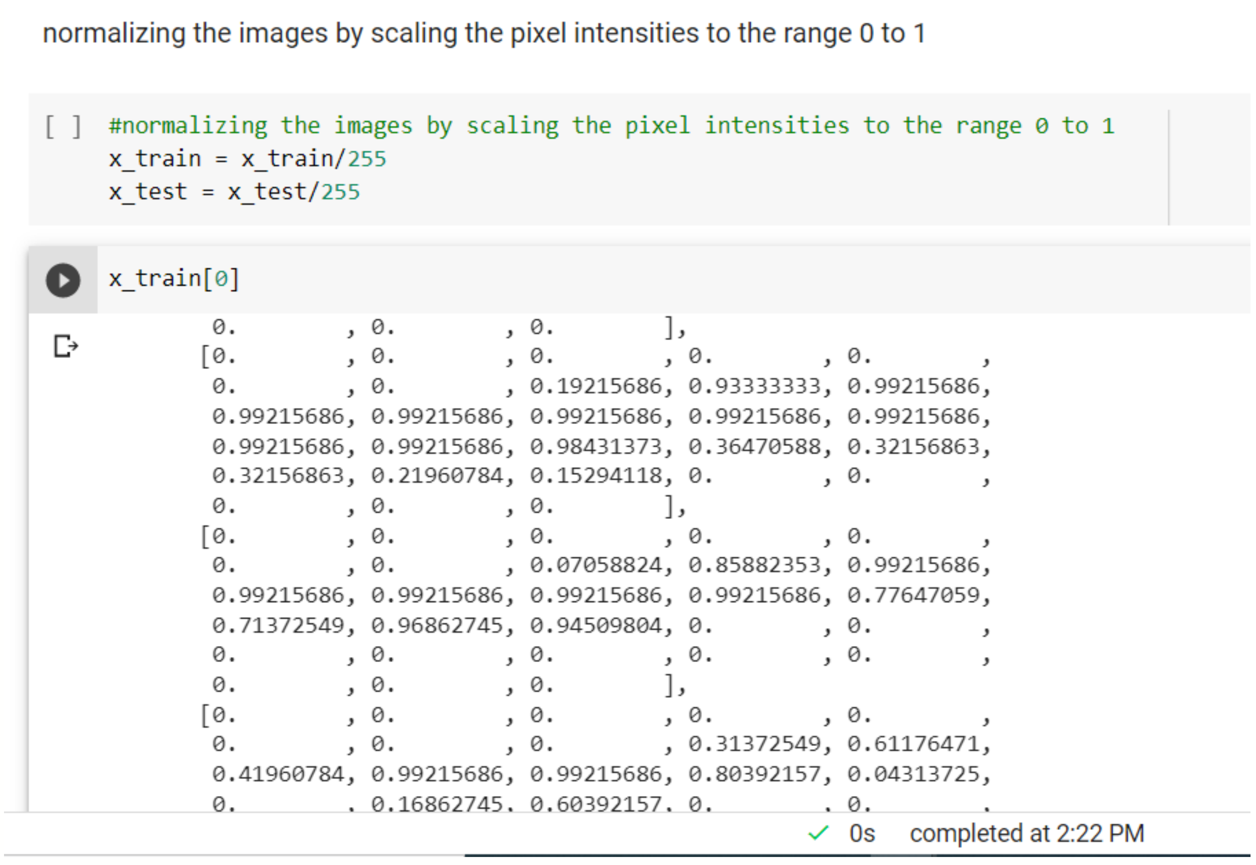
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

```
#to see how first image looks
plt.matshow(x_train[0])
```

```
<matplotlib.image.AxesImage at 0x7f686e32ed10>
```



Implementing Feed forward Neural Network with keras and tensorflow



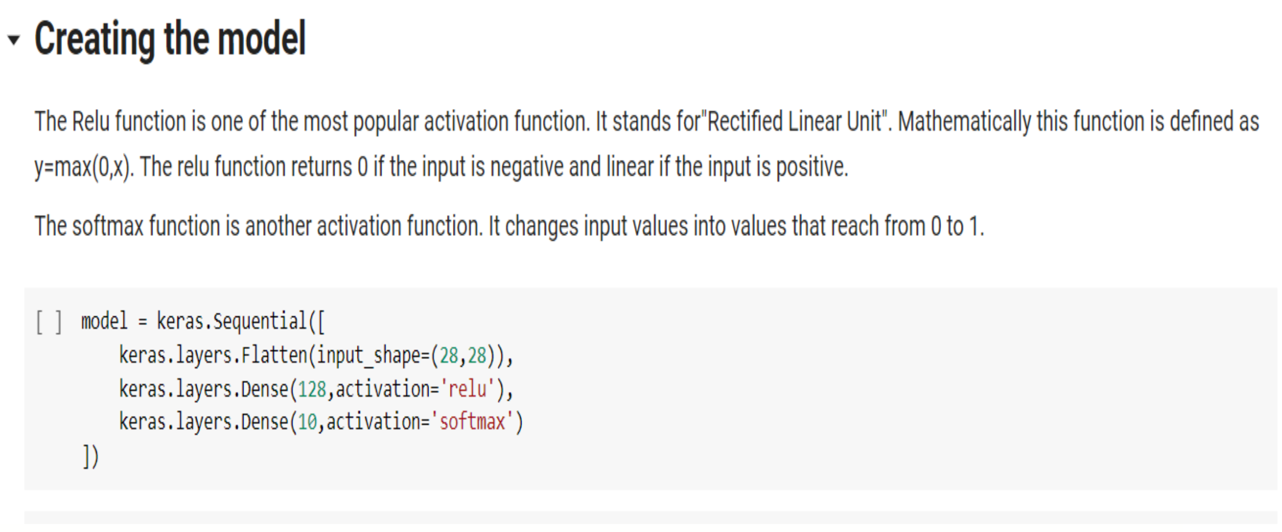
normalizing the images by scaling the pixel intensities to the range 0 to 1

```
[ ]  #normalizing the images by scaling the pixel intensities to the range 0 to 1
     x_train = x_train/255
     x_test = x_test/255
```

```
x_train[0]
```

```
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.19215686, 0.93333333, 0.99215686,
         0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
         0.99215686, 0.99215686, 0.98431373, 0.36470588, 0.32156863,
         0.32156863, 0.21960784, 0.15294118, 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.07058824, 0.85882353, 0.99215686,
         0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.77647059,
         0.71372549, 0.96862745, 0.94509804, 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.31372549, 0.61176471,
         0.41960784, 0.99215686, 0.99215686, 0.80392157, 0.04313725,
         0.        , 0.16862745, 0.60392157, 0.        , 0.        ,
```

✓ 0s completed at 2:22 PM



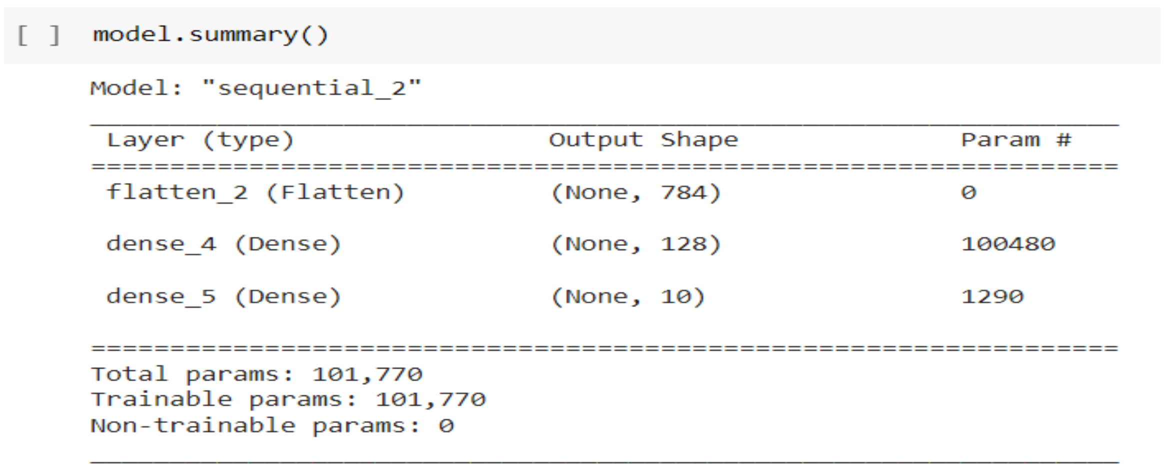
Creating the model

The Relu function is one of the most popular activation function. It stands for"Rectified Linear Unit". Mathematically this function is defined as y=max(0,x). The relu function returns 0 if the input is negative and linear if the input is positive.

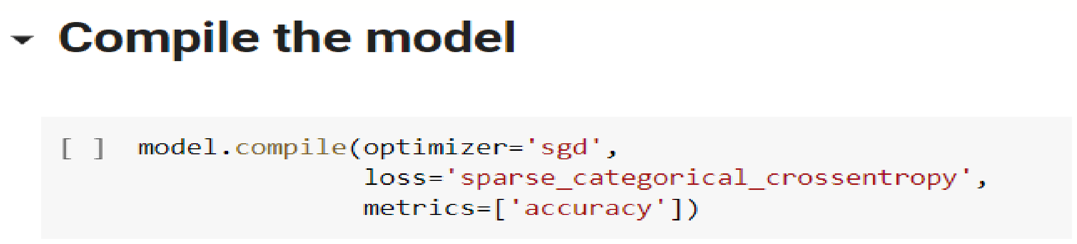
The softmax function is another activation function. It changes input values into values that reach from 0 to 1.

```
[ ]  model = keras.Sequential([
         keras.layers.Flatten(input_shape=(28,28)),
         keras.layers.Dense(128,activation='relu'),
         keras.layers.Dense(10,activation='softmax')
     ])
```

Implementing Feed forward Neural Network with keras and tensorflow

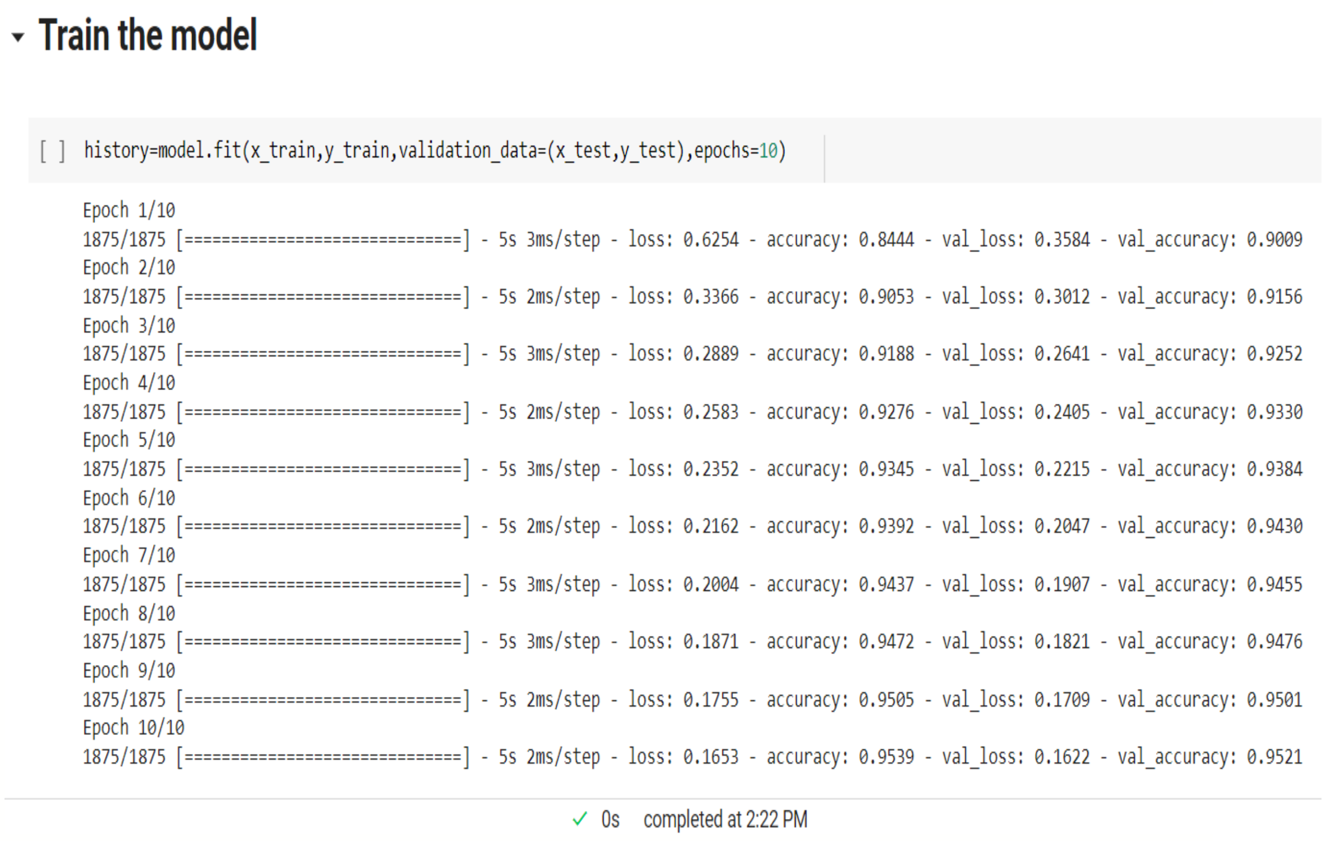


```
[ ]  model.summary()

Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_2 (Flatten)         (None, 784)               0

 dense_4 (Dense)             (None, 128)               100480

 dense_5 (Dense)             (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```



Compile the model

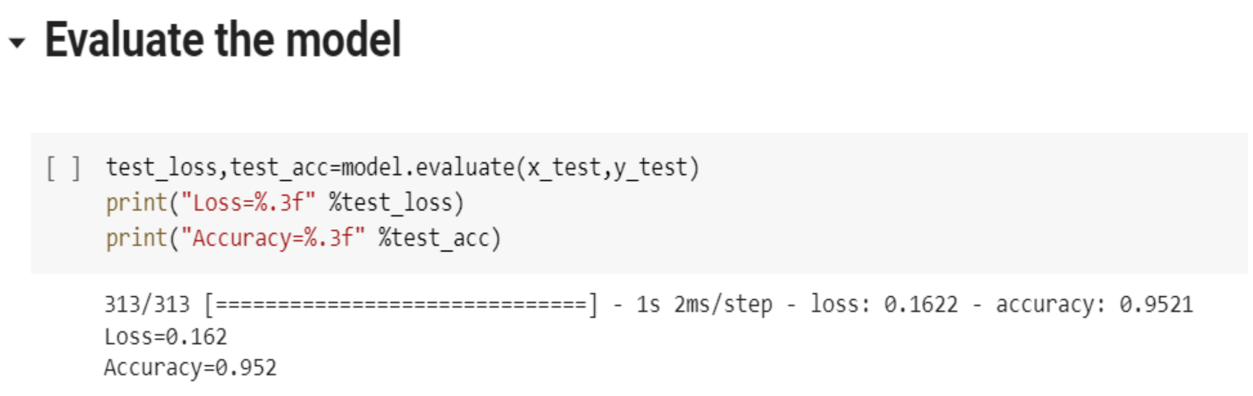
```
[ ]  model.compile(optimizer='sgd',
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])
```



Train the model

```
[ ] history=model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=10)

Epoch 1/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.6254 - accuracy: 0.8444 - val_loss: 0.3584 - val_accuracy: 0.9009
Epoch 2/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.3366 - accuracy: 0.9053 - val_loss: 0.3012 - val_accuracy: 0.9156
Epoch 3/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2889 - accuracy: 0.9188 - val_loss: 0.2641 - val_accuracy: 0.9252
Epoch 4/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.2583 - accuracy: 0.9276 - val_loss: 0.2405 - val_accuracy: 0.9330
Epoch 5/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2352 - accuracy: 0.9345 - val_loss: 0.2215 - val_accuracy: 0.9384
Epoch 6/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.2162 - accuracy: 0.9392 - val_loss: 0.2047 - val_accuracy: 0.9430
Epoch 7/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2004 - accuracy: 0.9437 - val_loss: 0.1907 - val_accuracy: 0.9455
Epoch 8/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1871 - accuracy: 0.9472 - val_loss: 0.1821 - val_accuracy: 0.9476
Epoch 9/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.1755 - accuracy: 0.9505 - val_loss: 0.1709 - val_accuracy: 0.9501
Epoch 10/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.1653 - accuracy: 0.9539 - val_loss: 0.1622 - val_accuracy: 0.9521
```

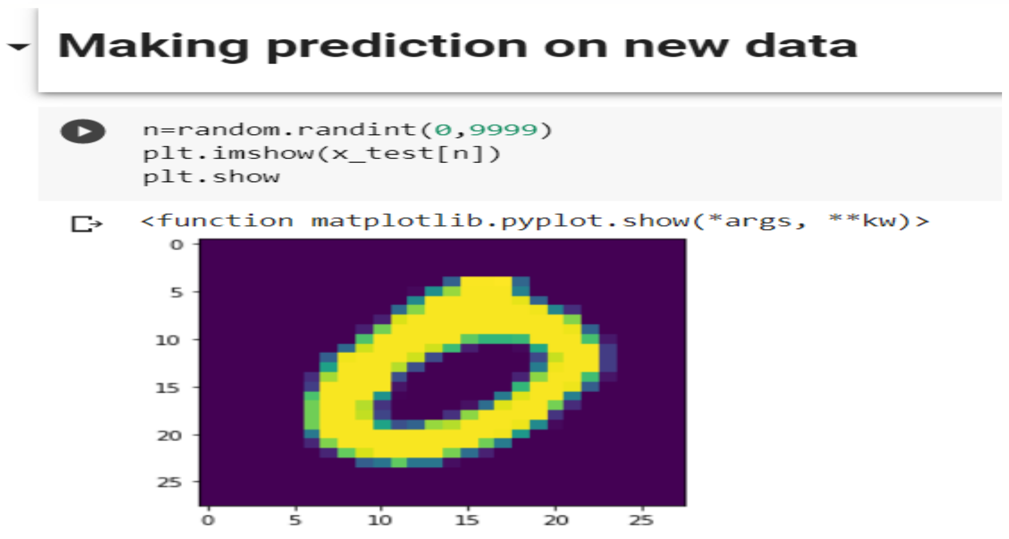
✓ 0s completed at 2:22 PM



Evaluate the model

```
[ ] test_loss,test_acc=model.evaluate(x_test,y_test)
    print("Loss=%.3f" %test_loss)
    print("Accuracy=%.3f" %test_acc)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.1622 - accuracy: 0.9521
Loss=0.162
Accuracy=0.952
```

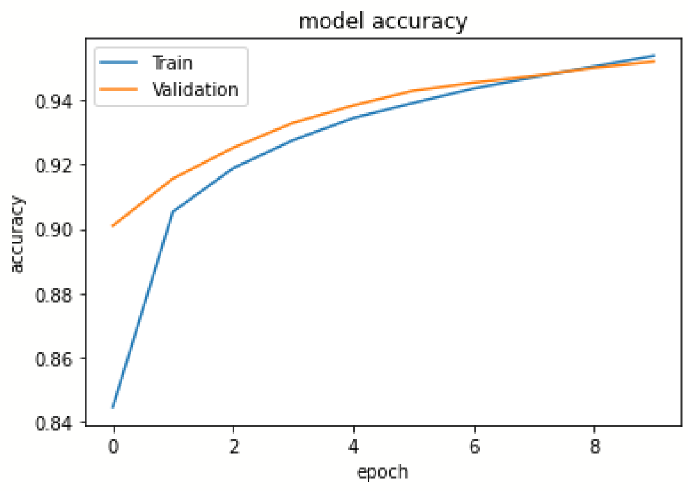


Making prediction on new data

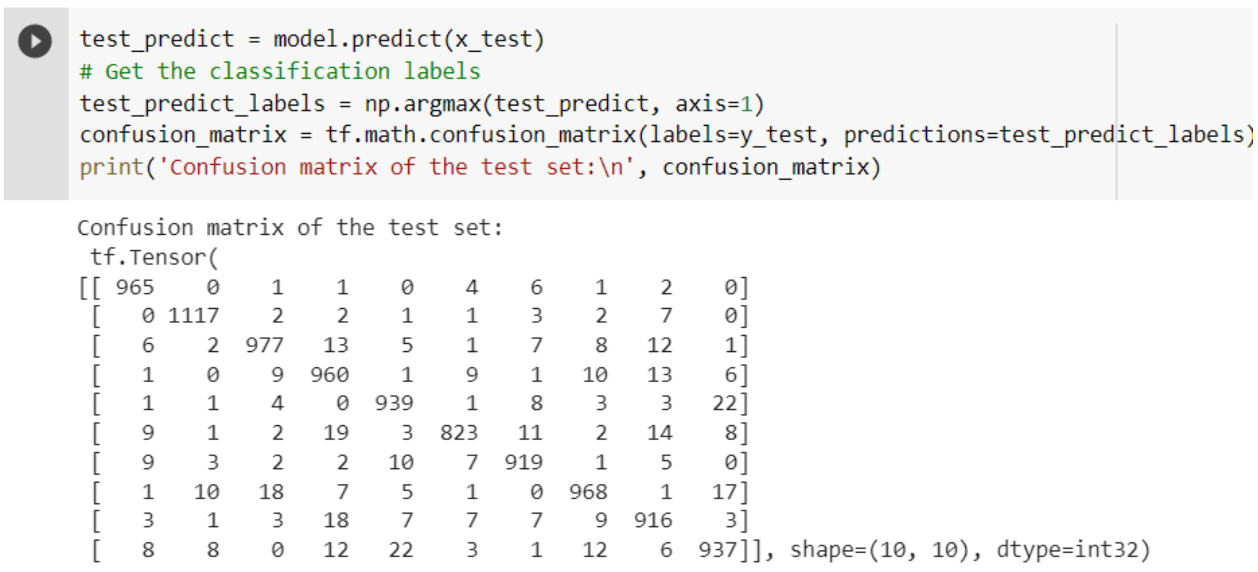
```
n=random.randint(0,9999)
plt.imshow(x_test[n])
plt.show
```

```
<function matplotlib.pyplot.show(*args, **kw)>
```

Plot graph for accuracy and loss



Confusion Matrix



```
test_predict = model.predict(x_test)
# Get the classification labels
test_predict_labels = np.argmax(test_predict, axis=1)
confusion_matrix = tf.math.confusion_matrix(labels=y_test, predictions=test_predict_labels)
print('Confusion matrix of the test set:\n', confusion_matrix)
```

```
Confusion matrix of the test set:
 tf.Tensor(
[[ 965    0    1    1    0    4    6    1    2    0]
 [   0 1117    2    2    1    1    3    2    7    0]
 [   6    2  977   13    5    1    7    8   12    1]
 [   1    0    9  960    1    9    1   10   13    6]
 [   1    1    4    0  939    1    8    3    3   22]
 [   9    1    2   19    3  823   11    2   14    8]
 [   9    3    2    2   10    7  919    1    5    0]
 [   1   10   18    7    5    1    0  968    1   17]
 [   3    1    3   18    7    7    7    9  916    3]
 [   8    8    0   12   22    3    1   12    6  937]], shape=(10, 10), dtype=int32)
```

Conclusion

With above code we can see that, throughout the epochs, our model accuracy increases and loss decreases that is good since our model gains confidence with our prediction

This indicates the model is trained in a good way

1. The two loss(loss and val_loss) are decreasing and the accuracy (accuracy and val_accuracy) increasing.
2. The val_accuracy is the measure of how good the model is predicting so, it is observed that the model is well trained after 10 epochs

References

1. S. Arora and M. P. S. Bhatia, "Handwriting recognition using Deep Learning in Keras," *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, 2018, pp. 142-145, doi: 10.1109/ICACCCN.2018.8748540.
2. https://towardsdatascience.com/feed-forward-neural-networks-how-to-successfully-build-them-in-python-74503409d99a
3. https://pyimagesearch.com/2021/05/06/implementing-feedforward-neural-networks-with-keras-and-tensorflow/

4. https://exchange.scale.com/public/blogs/how-to-build-a-fully-connected-feedforward-neural-network-using-keras-and-tensorflow
5. https://www.kaggle.com/code/prashant111/mnist-deep-neural-network-with-keras/notebook
6. https://sanjayasubedi.com.np/deeplearning/tensorflow-2-first-neural-network-for-fashion-mnist/