Chandawad Saurav Balaji

IEC2020119

IIIT Allahabad

04 September 20XX

AI Major Project

Project Overview

From a long time the term named AI has fascinated us a lot.So here's an AI project built by me Which identifies the given handwritten(scribbled) image and Displays it.

This project is built with the help of tensorflow and Keras.TensorFlow is a very popular machine learning tool.By leveraging TensorFlow, we can get up and running with machine learning as fast as within 15 minutes.

So let's get started with the amazing process of Making The project

Problem Statement

1. Find a dataset online (see the "Where to Find Datasets" section below)
2. Understand and describe the modeling objective clearly

    1. What type of data is it? (images, text, audio, etc.)
    2. What type of problem is it? (regression, classification, generative modeling, etc.)
3. Clean the data if required and perform exploratory analysis (plot graphs, ask questions)
4. Modeling
    1. Define a model (network architecture)
    2. Pick some hyperparameters
    3. Train the model
    4. Make predictions on samples
    5. Evaluate the test dataset
    6. Save the model weights
    7. Record the metrics (loss, accuracy per epochs)
    8. Try different hyperparameters & regularization
5. Conclusions - summarize your learning & identify opportunities for future work
6. Publish and submit your Jupyter notebook
7. Write a report to describe your experiments and summarize your work.

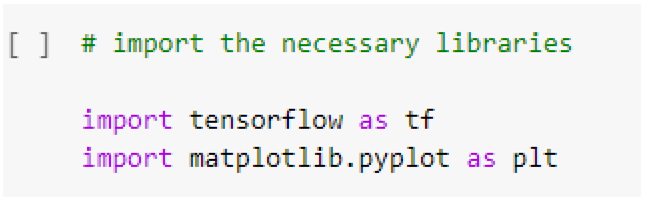
The process

Importing Necessary Libraries

First thing, importing the necessary libraries. We will import two libraries for this Project.

First one is TensorFlow, which is the major library we will use. We will import tensorflow as tf, meaning that, for us to use tensorflow in our code, we won't need to write 'tensorflow' each time, but we will simply use the alias of 'tf'. This is kind of the default way to import tensorflow.

We will also import matplotlib, which is a very popular data visualization library.

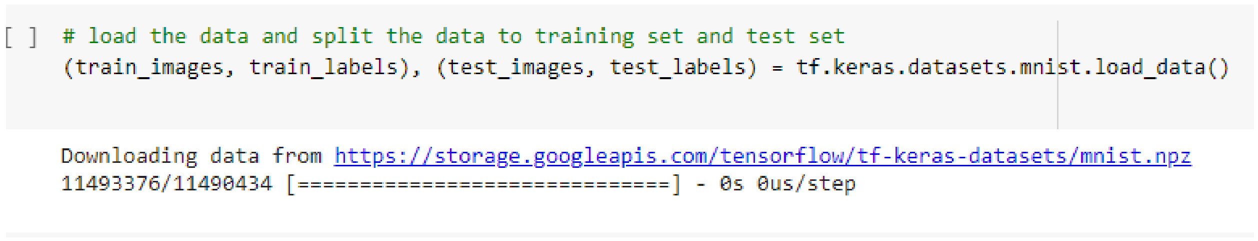


```
[ ]  # import the necessary libraries

     import tensorflow as tf
     import matplotlib.pyplot as plt
```

Load and Splitting the Data

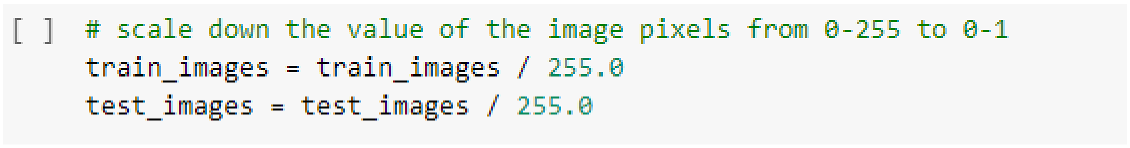
Next step is to get the data we are going to work with and split it to training and test datasets.

Training set is the part of the data we will train our neural network on. Once training is complete, we will test its performance on the test dataset. In both of the datasets, we will have two things, the actual images of the hand written digits, and the corresponding labels for that digit.



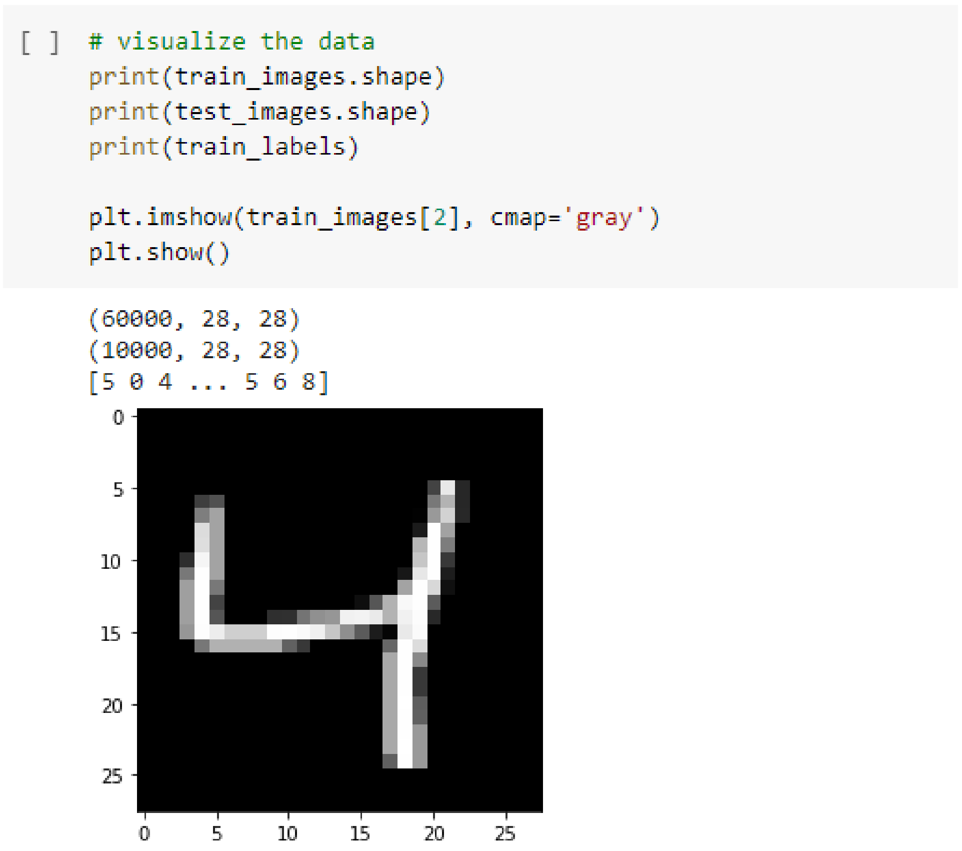
```
[ ]  # load the data and split the data to training set and test set
     (train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()

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

Next step is to scale down the pixel values of the images from 0–255 range to 0–1 range. Neural networks work much better with numbers that are close to zero. One of the most common ranges to convert is 0–1.



```
[ ]  # scale down the value of the image pixels from 0-255 to 0-1
     train_images = train_images / 255.0
     test_images = test_images / 255.0
```

Visualizing the Data



```
[ ]  # visualize the data
     print(train_images.shape)
     print(test_images.shape)
     print(train_labels)

     plt.imshow(train_images[2], cmap='gray')
     plt.show()
```

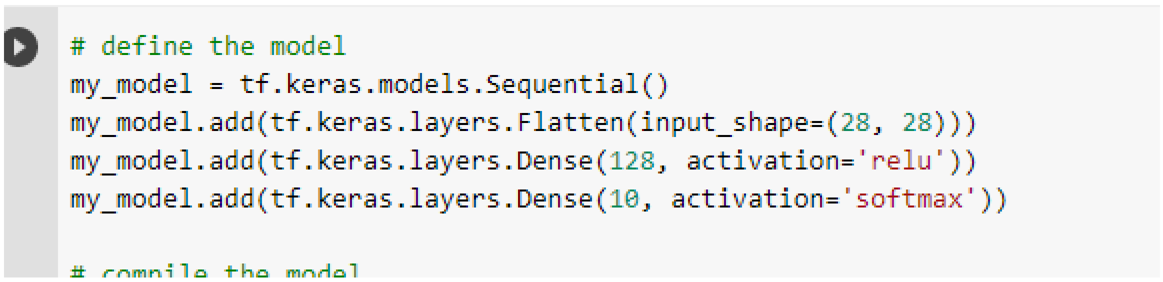
```
(60000, 28, 28)
(10000, 28, 28)
[5 0 4 ... 5 6 8]
```

Running this code cell should give you something like this :

(60000, 28, 28)(10000, 28, 28)[5 0 4 ... 5 6 8]

What this output tells us is that we have 60,000 images in the training set that are grayscale and are 28 x 28 pixels. If we were working with colored images, the shape of our training images would look like this: (60000, 28, 28, 3). The 3 here, would have come from 3 color channels (most likely: red, green, blue) that we would have needed to represent the image.

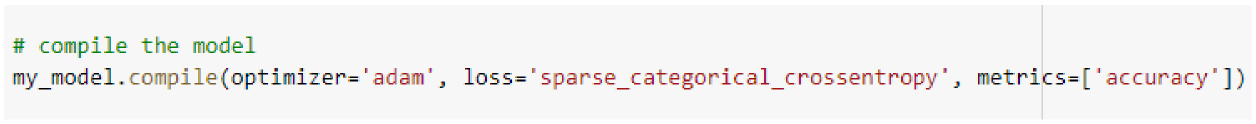
Defining the Machine Learning Model



```python
# define the model
my_model = tf.keras.models.Sequential()
my_model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))
my_model.add(tf.keras.layers.Dense(128, activation='relu'))
my_model.add(tf.keras.layers.Dense(10, activation='softmax'))

# compile the model
```

Compiling the Model

Next step is to compile the model. When compiling the model, we will use one of the most popular optimizers and loss functions.This function is highly useful when we are classifying more than 2 classes. Then, in order to track the accuracy of the model, we are going to use the metrics.

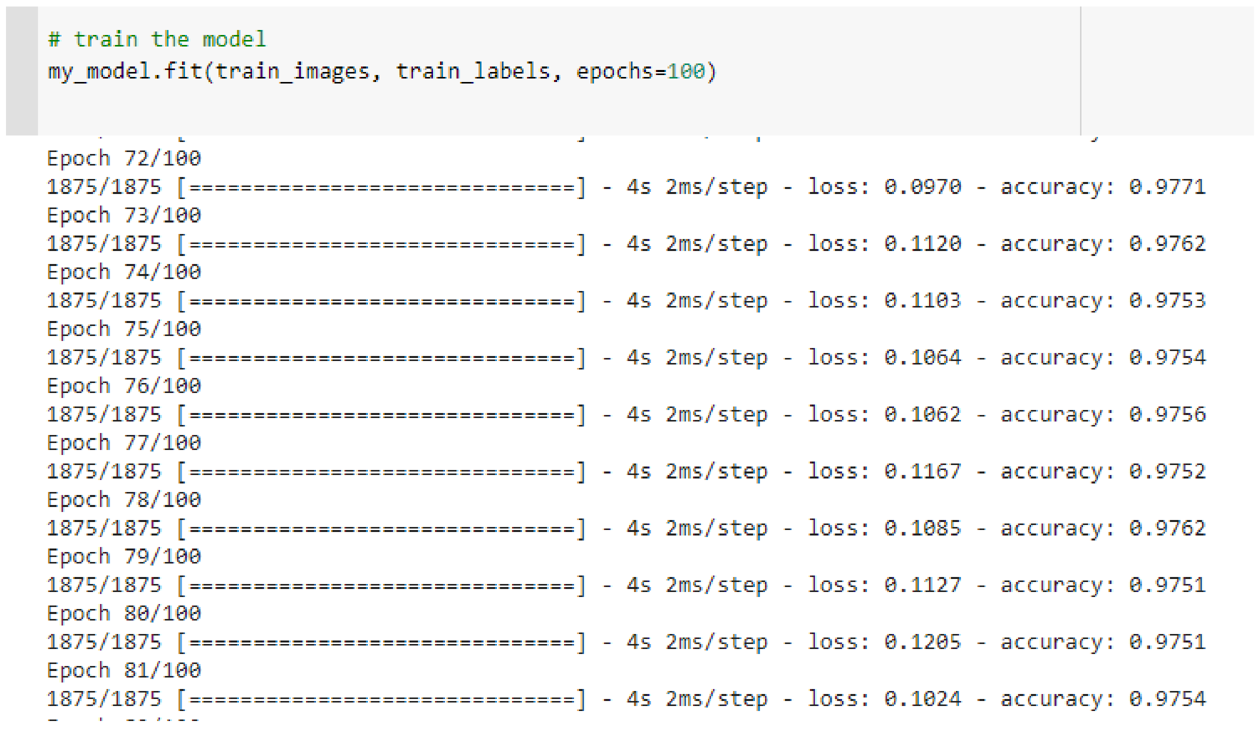


```python
# compile the model
my_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Training the Model

Next step is to train the model. Until now, our model had never seen the data before. In the training step, we will train the neural network model with the images of the hand written digits, as well as the corresponding labels for those images.

We have 60,000 training images and we will show those images 100 times to our model. To communicate that, we will enter the “epochs” value as 100.

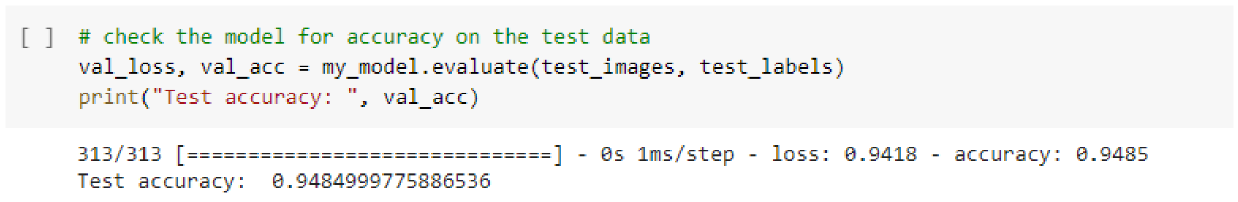


```
# train the model
my_model.fit(train_images, train_labels, epochs=100)
```

```
Epoch 72/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0970 - accuracy: 0.9771
Epoch 73/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1120 - accuracy: 0.9762
Epoch 74/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1103 - accuracy: 0.9753
Epoch 75/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1064 - accuracy: 0.9754
Epoch 76/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1062 - accuracy: 0.9756
Epoch 77/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1167 - accuracy: 0.9752
Epoch 78/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1085 - accuracy: 0.9762
Epoch 79/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1127 - accuracy: 0.9751
Epoch 80/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1205 - accuracy: 0.9751
Epoch 81/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1024 - accuracy: 0.9754
```

Checking the Model for Accuracy on the Test Data

To get a more realistic view on the model’s performance, we should measure its accuracy on data it hasn’t seen before. In order to achieve that, we will use the test dataset. To test the model, we will write my_model.evaluate() and we will give it the testing images, as well as the corresponding labels for those images. We will also print the test accuracy on a separate line.

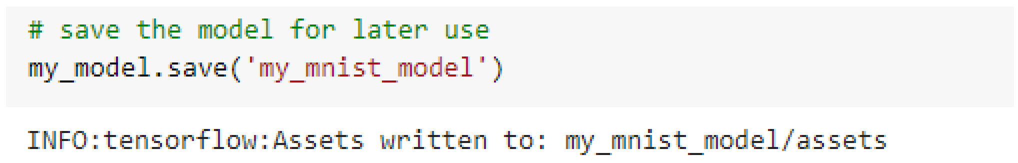


```
[ ]  # check the model for accuracy on the test data
     val_loss, val_acc = my_model.evaluate(test_images, test_labels)
     print("Test accuracy: ", val_acc)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.9418 - accuracy: 0.9485
Test accuracy:  0.9484999775886536
```

Saving the Model

Saving the model. If we do not save the model, we will need to recreate it every time want to use it.To save our model, we will write my_model.save() and we will also give it a file directory to save the file.

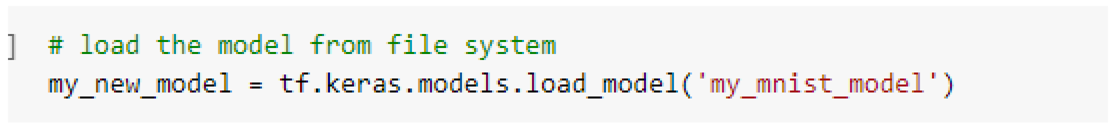


```
# save the model for later use
my_model.save('my_mnist_model')
```

```
INFO:tensorflow:Assets written to: my_mnist_model/assets
```

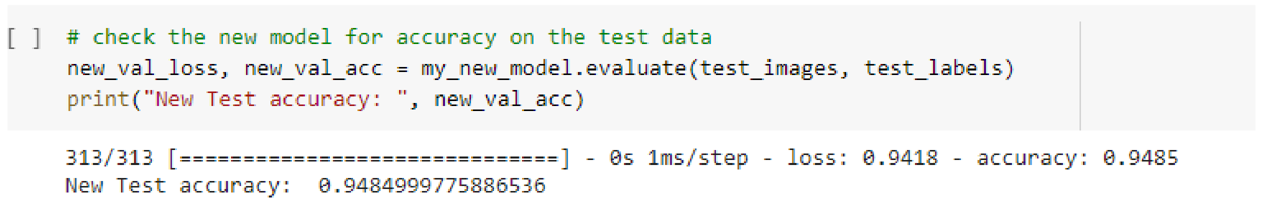
Retrieving the Model

Let’s also retrieve the model from the file system to make sure that it works. To load the model, we need to locate the model in the file system and pass it to the .load_model() method. Since we are in the same notebook, we can just copy and paste the file address from the previous code cell, where we saved the model.



```
] # load the model from file system
  my_new_model = tf.keras.models.load_model('my_mnist_model')
```

Checking the new the New Model for Accuracy:



```
[ ]  # check the new model for accuracy on the test data
     new_val_loss, new_val_acc = my_new_model.evaluate(test_images, test_labels)
     print("New Test accuracy: ", new_val_acc)

     313/313 [==============================] - 0s 1ms/step - loss: 0.9418 - accuracy: 0.9485
     New Test accuracy:  0.9484999775886536
```

The Scope And Opportunities for future work:

Game Programmer

Video games are widely loved. However, in the current scenario, games require highly intelligent enemies to keep the players challenged. This is where a software engineer or a game programmer comes into play. Companies are on the lookout for candidates who are well-educated, thorough with the basics of AI and can design games that keep the user engaged.

Robotic Scientist

Not only are robots cool, they are also gradually taking over the industrial world. This field needs engineers or programmers that can program these robots to solve problems like a human would.

Having a master's in robotic engineers and a license from the state can help a great deal for a career in this field.

Software Engineer (Face Recognition Software)

Many companies, including security companies, police departments, casinos, and even Google are utilising face recognition to understand the people benefiting from their service.

Search Engine Manager

One of the highest paying companies from face recognition, Google also uses AI in other interesting ways. The organisation hires people with artificial intelligence degrees to manage their massive search engine. Users search for a variety of things and Google Search should be able to predict what the users are searching for despite the spelling and grammatical errors for the searched phrase. This is where knowledge and study of artificial intelligence come into play.

THE END