Importing the Required Libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras import layers

from tensorflow.keras.layers import Dense,Flatten,MaxPooling2D

from tensorflow.keras.layers import Conv2D

from keras.optimizers import Adam

from keras.utils import np\_utils

from tensorflow.keras.models import load\_model

Loading the Data

(X\_train,y\_train),(X\_test,y\_test) = mnist.load\_data()

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>

11490434/11490434 [==============================] - 0s 0us/step

print(X\_train.shape)

print(X\_test.shape)

(60000, 28, 28)

(10000, 28, 28)

Analyzing the data

X\_train[0]

array([[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3,

18, 18, 18, 126, 136, 175, 26, 166, 255, 247, 127, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 30, 36, 94, 154, 170,

253, 253, 253, 253, 253, 225, 172, 253, 242, 195, 64, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 49, 238, 253, 253, 253, 253,

253, 253, 253, 253, 251, 93, 82, 82, 56, 39, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 18, 219, 253, 253, 253, 253,

253, 198, 182, 247, 241, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 80, 156, 107, 253, 253,

205, 11, 0, 43, 154, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 14, 1, 154, 253,

90, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 139, 253,

190, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 11, 190,

253, 70, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 35,

241, 225, 160, 108, 1, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

81, 240, 253, 253, 119, 25, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 45, 186, 253, 253, 150, 27, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 16, 93, 252, 253, 187, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 249, 253, 249, 64, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 46, 130, 183, 253, 253, 207, 2, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 39,

148, 229, 253, 253, 253, 250, 182, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 24, 114, 221,

253, 253, 253, 253, 201, 78, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 23, 66, 213, 253, 253,

253, 253, 198, 81, 2, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 18, 171, 219, 253, 253, 253, 253,

195, 80, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 55, 172, 226, 253, 253, 253, 253, 244, 133,

11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 136, 253, 253, 253, 212, 135, 132, 16, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

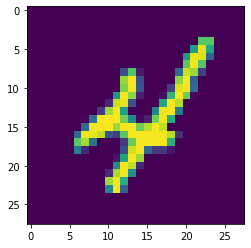
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0]], dtype=uint8)

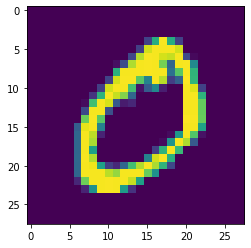
y\_train[2]

4

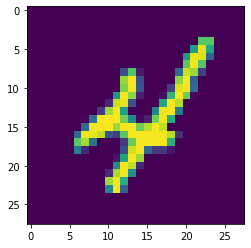
plt.imshow(X\_train[2])



plt.imshow(X\_train[1])



plt.imshow(X\_train[9])



for i in range(9):

plt.subplot(330 + 1 +i)

plt.imshow(X\_train[i], cmap = plt.get\_cmap('gray'))

plt.show()



















reshaping the data

X\_train = X\_train.reshape(60000,28,28,1).astype('float32')

X\_test = X\_test.reshape(10000,28,28,1).astype('float32')

X\_train

array([[[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

...,

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]]],

[[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

...,

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]]],

[[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

...,

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]]],

...,

[[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

...,

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]]],

[[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

...,

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]]],

[[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

...,

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]]]], dtype=float32)

X\_test

array([[[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

...,

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]]],

[[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

...,

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]]],

[[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

...,

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]]],

...,

[[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

...,

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]]],

[[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

...,

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]]],

[[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

...,

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]],

[[0.],

[0.],

[0.],

...,

[0.],

[0.],

[0.]]]], dtype=float32)

Applying One Hot Encoding

number\_of\_classes = 20

y\_train = np\_utils.to\_categorical(y\_train,number\_of\_classes)

y\_test = np\_utils.to\_categorical(y\_test,number\_of\_classes)

y\_train[0]

array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0.], dtype=float32)

y\_test

array([[0., 0., 0., ..., 0., 0., 0.],

[0., 0., 1., ..., 0., 0., 0.],

[0., 1., 0., ..., 0., 0., 0.],

...,

[0., 0., 0., ..., 0., 0., 0.],

[0., 0., 0., ..., 0., 0., 0.],

[0., 0., 0., ..., 0., 0., 0.]], dtype=float32)

model creation

model = Sequential()

model.add(Conv2D(64,(3,3),input\_shape=(28,28,1),activation="relu"))

model.add(Conv2D(32,(3,3),activation="relu"))

model.add(MaxPooling2D((2,2)))

model.add(Flatten())

model.add(Dense(number\_of\_classes,activation="softmax"))

model.summary()

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 26, 26, 64) 640

conv2d\_1 (Conv2D) (None, 24, 24, 32) 18464

max\_pooling2d (MaxPooling2D (None, 12, 12, 32) 0

)

flatten (Flatten) (None, 4608) 0

dense (Dense) (None, 20) 92180

=================================================================

Total params: 111,284

Trainable params: 111,284

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Model compilation

model.compile(loss="categorical\_crossentropy",optimizer="Adam",metrics=["accuracy"])

Train the model

model.fit(X\_train,y\_train,epochs=10,validation\_data=(X\_test,y\_test),batch\_size=32)

Epoch 1/10

1875/1875 [==============================] - 206s 109ms/step - loss: 0.2241 - accuracy: 0.9506 - val\_loss: 0.0835 - val\_accuracy: 0.9752

Epoch 2/10

1875/1875 [==============================] - 196s 105ms/step - loss: 0.0746 - accuracy: 0.9775 - val\_loss: 0.0756 - val\_accuracy: 0.9762

Epoch 3/10

1875/1875 [==============================] - 198s 105ms/step - loss: 0.0590 - accuracy: 0.9822 - val\_loss: 0.0821 - val\_accuracy: 0.9768

Epoch 4/10

1875/1875 [==============================] - 197s 105ms/step - loss: 0.0471 - accuracy: 0.9851 - val\_loss: 0.0890 - val\_accuracy: 0.9770

Epoch 5/10

1875/1875 [==============================] - 195s 104ms/step - loss: 0.0401 - accuracy: 0.9876 - val\_loss: 0.0744 - val\_accuracy: 0.9806

Epoch 6/10

1875/1875 [==============================] - 195s 104ms/step - loss: 0.0340 - accuracy: 0.9893 - val\_loss: 0.0927 - val\_accuracy: 0.9774

Epoch 7/10

1875/1875 [==============================] - 194s 104ms/step - loss: 0.0274 - accuracy: 0.9915 - val\_loss: 0.0729 - val\_accuracy: 0.9834

Epoch 8/10

1875/1875 [==============================] - 194s 103ms/step - loss: 0.0257 - accuracy: 0.9924 - val\_loss: 0.0878 - val\_accuracy: 0.9814

Epoch 9/10

1875/1875 [==============================] - 195s 104ms/step - loss: 0.0230 - accuracy: 0.9931 - val\_loss: 0.1060 - val\_accuracy: 0.9806

Epoch 10/10

1875/1875 [==============================] - 194s 103ms/step - loss: 0.0215 - accuracy: 0.9933 - val\_loss: 0.1080 - val\_accuracy: 0.9799

Observing the metrics

metrics=model.evaluate(X\_test,y\_test,verbose=0)

print("Metrics(Test Loss & Test Accuracy):")

print(metrics)

Metrics(Test Loss & Test Accuracy):

[0.10799671709537506, 0.9799000024795532]

Test the model

prediction = model.predict(X\_test[:4])

print(prediction)

1/1 [==============================] - 0s 92ms/step

[[2.3213846e-21 1.2361092e-20 1.3299491e-12 2.4462179e-14 6.0258951e-23

8.8684134e-21 2.7768175e-29 1.0000000e+00 2.0004091e-15 2.2394742e-13

2.0498932e-17 9.2686456e-18 1.1635386e-16 2.2390158e-18 1.2966898e-16

1.3297199e-17 1.2113255e-15 7.5911792e-19 2.8659866e-17 1.4978137e-16]

[6.9119814e-11 2.6640326e-08 1.0000000e+00 1.5507807e-12 2.2414792e-16

1.5020233e-18 2.5328071e-09 3.6340596e-18 2.1907884e-08 1.0689899e-17

2.9735371e-15 7.2056701e-16 1.2506362e-14 3.3254159e-16 4.1007094e-14

3.6598279e-16 6.6564923e-13 6.6131740e-17 1.3557149e-15 1.1202374e-15]

[7.7130238e-16 9.9999547e-01 6.1150490e-10 2.1368451e-14 1.0306390e-08

5.8895429e-13 5.5677939e-14 2.2215945e-14 4.5528609e-06 1.0291512e-17

6.7287052e-15 9.7331838e-14 1.5531489e-14 7.8771337e-15 8.5694588e-14

5.2174055e-14 8.4508943e-16 7.3548063e-13 1.9848642e-14 3.7636659e-14]

[1.0000000e+00 5.4667512e-23 2.4307383e-15 6.6788842e-18 1.8069466e-19

5.1434537e-17 7.9950508e-12 1.4670674e-17 8.9687738e-14 2.3780449e-13

1.6349322e-20 3.9431183e-21 1.5496893e-20 1.2390640e-21 1.8741801e-20

1.5800853e-20 2.6178145e-18 1.4321822e-23 6.0590011e-21 1.1927565e-20]]

print(np.argmax(prediction,axis = 1))

print(y\_test[:4])

[7 2 1 0]

[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]

Observing the metrics

metrics=model.evaluate(X\_test,y\_test,verbose=1)

print("Metrics(Test Loss & Test Accuracy):")

print(metrics)

313/313 [==============================] - 7s 23ms/step - loss: 0.1080 - accuracy: 0.9799

Metrics(Test Loss & Test Accuracy):

[0.10799671709537506, 0.9799000024795532]

Test The Model

prediction = model.predict(X\_test[1:5])

print(prediction)

1/1 [==============================] - 0s 21ms/step

[[6.91198140e-11 2.66403255e-08 1.00000000e+00 1.55078066e-12

2.24147920e-16 1.50202330e-18 2.53280708e-09 3.63405962e-18

2.19078835e-08 1.06898991e-17 2.97353711e-15 7.20567011e-16

1.25063618e-14 3.32541588e-16 4.10070944e-14 3.65982793e-16

6.65649231e-13 6.61317400e-17 1.35571488e-15 1.12023739e-15]

[7.71302378e-16 9.99995470e-01 6.11504902e-10 2.13684508e-14

1.03063904e-08 5.88954287e-13 5.56779388e-14 2.22159446e-14

4.55286090e-06 1.02915123e-17 6.72870522e-15 9.73318377e-14

1.55314892e-14 7.87713366e-15 8.56945875e-14 5.21740549e-14

8.45089430e-16 7.35480633e-13 1.98486416e-14 3.76366588e-14]

[1.00000000e+00 5.46675117e-23 2.43073828e-15 6.67888415e-18

1.80694658e-19 5.14345372e-17 7.99505080e-12 1.46706735e-17

8.96877380e-14 2.37804486e-13 1.63493217e-20 3.94311828e-21

1.54968927e-20 1.23906395e-21 1.87418011e-20 1.58008525e-20

2.61781447e-18 1.43218218e-23 6.05900105e-21 1.19275648e-20]

[3.86578858e-14 8.72129680e-22 2.09616881e-17 7.81928622e-18

1.00000000e+00 2.08864891e-14 1.28101165e-15 7.05227350e-15

5.98234040e-09 1.61962390e-13 3.39168959e-16 1.06734374e-17

1.03352340e-16 7.17151839e-19 4.59796880e-17 1.80138745e-17

8.95966371e-17 1.64461939e-15 1.10847405e-17 1.37731581e-16]]

print(np.argmax(prediction,axis = 1))

print(y\_test[1:5])

[2 1 0 4]

[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]

Save The Model

model.save("MNIST.h5")