**TESTING OF DIFFERENT ALGORITHMS ON DIFFERENT DATASET AND THEIR ACCURACY:**

**Code for the testing of CNN algorithm on the NIST Special-19 Handwritten Character Dataset:**

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
import os  
from sklearn.preprocessing import MinMaxScaler  
import tensorflow as tf  
from keras.models import Sequential  
from keras.layers import Dense  
from keras.layers import Dropout  
from keras.layers import Flatten  
from keras.layers.convolutional import Conv2D  
from keras.layers.convolutional import MaxPooling2D  
from keras import backend as K  
from keras.utils import np\_utils  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import confusion\_matrix  
  
#ignore warning messages  
import warnings  
warnings.filterwarnings('ignore')  
  
sns.set()  
dataset = pd.read\_csv("A\_Z Handwritten Data.csv").astype('float32')  
dataset.rename(columns={'0':'label'}, inplace=True)  
  
# Splite data the X - Our data , and y - the prdict label  
X = dataset.drop('label',axis = 1)  
y = dataset['label']  
#print("shape:",X.shape)  
#print("culoms count:",len(X.iloc[1]))  
#print("784 = 28X28")  
  
X.head()  
from sklearn.utils import shuffle  
  
X\_shuffle = shuffle(X)  
  
#plt.figure(figsize = (12,10))  
row, colums = 4, 4  
for i in range(16):  
 plt.subplot(colums, row, i+1)  
 plt.imshow(X\_shuffle.iloc[i].values.reshape(28,28),interpolation='nearest', cmap='Greys')  
#plt.show()  
  
#print("Amount of each labels")  
  
# Change label to alphabets  
alphabets\_mapper = {0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',6:'G',7:'H',8:'I',9:'J',10:'K',11:'L',12:'M',13:'N',14:'O',15:'P',16:'Q',17:'R',18:'S',19:'T',20:'U',21:'V',22:'W',23:'X',24:'Y',25:'Z'}  
dataset\_alphabets = dataset.copy()  
dataset['label'] = dataset['label'].map(alphabets\_mapper)  
  
label\_size = dataset.groupby('label').size()  
label\_size.plot.barh(figsize=(10,10))  
#plt.show()  
  
#print("We have very low observations for I and F ")  
#print("I count:", label\_size['I'])  
#print("F count:", label\_size['F'])  
  
# splite the data  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y)  
  
# scale data  
standard\_scaler = MinMaxScaler()  
standard\_scaler.fit(X\_train)  
  
X\_train = standard\_scaler.transform(X\_train)  
X\_test = standard\_scaler.transform(X\_test)  
  
#print("Data after scaler")  
X\_shuffle = shuffle(X\_train)  
  
plt.figure(figsize = (12,10))  
row, colums = 4, 4  
for i in range(16):  
 plt.subplot(colums, row, i+1)  
 plt.imshow(X\_shuffle[i].reshape(28,28),interpolation='nearest', cmap='Greys')  
#plt.show()  
  
X\_train = X\_train.reshape(X\_train.shape[0], 28, 28, 1).astype('float32')  
X\_test = X\_test.reshape(X\_test.shape[0], 28, 28, 1).astype('float32')  
  
y\_train = np\_utils.to\_categorical(y\_train)  
y\_test = np\_utils.to\_categorical(y\_test)  
  
  
cls = Sequential()  
cls.add(Conv2D(32, (5, 5), input\_shape=(28, 28, 1), activation='relu'))  
cls.add(MaxPooling2D(pool\_size=(2, 2)))  
cls.add(Dropout(0.3))  
cls.add(Flatten())

cls.add(Dense(128, activation='relu'))  
cls.add(Dense(len(y.unique()), activation='softmax'))  
  
cls.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])  
history = cls.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=5, batch\_size=100, verbose=2)  
  
scores = cls.evaluate(X\_test,y\_test, verbose=0)  
print("CNN Score:",scores[1])  
  
plt.plot(history.history['loss'])  
plt.plot(history.history['val\_loss'])  
plt.title('Model loss')  
plt.ylabel('Loss')  
plt.xlabel('Epoch')  
plt.legend(['Train', 'Test'], loc='upper left')  
plt.show()

**Output for CNN implementation on the NIST special-19 Dataset**

"C:\Users\MEET PARMAR\PycharmProjects\Internship\_project\venv\Scripts\python.exe" "C:/Users/MEET PARMAR/PycharmProjects/Internship\_project/CNN\_for\_handwritten.py"

2022-12-26 17:14:09.296324: I tensorflow/core/platform/cpu\_feature\_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

Epoch 1/5

2794/2794 - 123s - loss: 0.2003 - accuracy: 0.9437 - val\_loss: 0.0925 - val\_accuracy: 0.9753 - 123s/epoch - 44ms/step

Epoch 2/5

2794/2794 - 125s - loss: 0.0806 - accuracy: 0.9770 - val\_loss: 0.0705 - val\_accuracy: 0.9811 - 125s/epoch - 45ms/step

Epoch 3/5

2794/2794 - 125s - loss: 0.0602 - accuracy: 0.9827 - val\_loss: 0.0577 - val\_accuracy: 0.9853 - 125s/epoch - 45ms/step

Epoch 4/5

2794/2794 - 126s - loss: 0.0480 - accuracy: 0.9859 - val\_loss: 0.0521 - val\_accuracy: 0.9870 - 126s/epoch - 45ms/step

Epoch 5/5

2794/2794 - 97s - loss: 0.0399 - accuracy: 0.9879 - val\_loss: 0.0510 - val\_accuracy: 0.9877 - 97s/epoch - 35ms/step

**CNN Score: 0.9877246022224426**

**Average Accuracy – 98.77%**

**Code for testing accuracy of CNN on MNIST digits dataset:**

from tensorflow.keras.datasets import mnist  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Conv2D  
from tensorflow.keras.layers import MaxPool2D  
from tensorflow.keras.layers import Flatten  
from tensorflow.keras.layers import Dropout  
from tensorflow.keras.layers import Dense  
#loading data  
(X\_train,y\_train) , (X\_test,y\_test)=mnist.load\_data()  
#reshaping data  
X\_train = X\_train.reshape((X\_train.shape[0], X\_train.shape[1], X\_train.shape[2], 1))  
X\_test = X\_test.reshape((X\_test.shape[0],X\_test.shape[1],X\_test.shape[2],1))  
#checking the shape after reshaping  
print(X\_train.shape)  
print(X\_test.shape)  
#normalizing the pixel values  
X\_train=X\_train/255  
X\_test=X\_test/255  
#defining model  
model=Sequential()  
#adding convolution layer  
model.add(Conv2D(32,(3,3),activation='relu',input\_shape=(28,28,1)))  
#adding pooling layer  
model.add(MaxPool2D(2,2))  
#adding fully connected layer  
model.add(Flatten())  
model.add(Dense(100,activation='relu'))  
#adding output layer  
model.add(Dense(10,activation='softmax'))  
#compiling the model  
model.compile(loss='sparse\_categorical\_crossentropy',optimizer='adam',metrics=['accuracy'])  
#fitting the model  
model.fit(X\_train,y\_train,epochs=10)  
model.evaluate(X\_test,y\_test)

**Output for CNN implementation on MNIST digit dataset:**

"C:\Users\MEET PARMAR\PycharmProjects\Internship\_project\venv\Scripts\python.exe" "C:/Users/MEET PARMAR/PycharmProjects/Internship\_project/CNN\_implmentation.py"

(60000, 28, 28, 1)

(10000, 28, 28, 1)

2022-12-27 08:41:38.605477: I tensorflow/core/platform/cpu\_feature\_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

Epoch 1/10

1875/1875 [==============================] - 33s 17ms/step - loss: 0.1568 - accuracy: 0.9533

Epoch 2/10

1875/1875 [==============================] - 32s 17ms/step - loss: 0.0537 - accuracy: 0.9835

Epoch 3/10

1875/1875 [==============================] - 34s 18ms/step - loss: 0.0346 - accuracy: 0.9890

Epoch 4/10

1875/1875 [==============================] - 35s 19ms/step - loss: 0.0230 - accuracy: 0.9924

Epoch 5/10

1875/1875 [==============================] - 33s 17ms/step - loss: 0.0158 - accuracy: 0.9952

Epoch 6/10

1875/1875 [==============================] - 31s 17ms/step - loss: 0.0108 - accuracy: 0.9967

Epoch 7/10

1875/1875 [==============================] - 31s 17ms/step - loss: 0.0087 - accuracy: 0.9974

Epoch 8/10

1875/1875 [==============================] - 33s 17ms/step - loss: 0.0062 - accuracy: 0.9980

Epoch 9/10

1875/1875 [==============================] - 30s 16ms/step - loss: 0.0058 - accuracy: 0.9981

Epoch 10/10

1875/1875 [==============================] - 31s 17ms/step - loss: 0.0044 - accuracy: 0.9984

313/313 [==============================] - 2s 7ms/step - loss: 0.0510 - accuracy: 0.9884

Process finished with exit code 0

**Average accuracy – 98.84%**

**Code for testing accuracy of RNN\_LSTM on MNIST digits dataset:**  
  
import numpy as np  
import tensorflow as tf  
from tensorflow import keras  
from tensorflow.keras import layers  
model = keras.Sequential()  
# Add an Embedding layer expecting input vocab of size 1000, and  
# output embedding dimension of size 64.  
model.add(layers.Embedding(input\_dim=1000, output\_dim=64))  
  
# Add a LSTM layer with 128 internal units.  
model.add(layers.LSTM(128))  
  
# Add a Dense layer with 10 units.  
model.add(layers.Dense(10))  
model.summary()  
model = keras.Sequential()  
model.add(layers.Embedding(input\_dim=1000, output\_dim=64))  
  
# The output of GRU will be a 3D tensor of shape (batch\_size, timesteps, 256)  
model.add(layers.GRU(256, return\_sequences=True))  
  
# The output of SimpleRNN will be a 2D tensor of shape (batch\_size, 128)  
model.add(layers.SimpleRNN(128))  
  
model.add(layers.Dense(10))  
  
model.summary()  
  
  
encoder\_vocab = 1000  
decoder\_vocab = 2000  
  
encoder\_input = layers.Input(shape=(None,))  
encoder\_embedded = layers.Embedding(input\_dim=encoder\_vocab, output\_dim=64)(  
 encoder\_input  
)  
  
# Return states in addition to output  
output, state\_h, state\_c = layers.LSTM(64, return\_state=True, name="encoder")(  
 encoder\_embedded  
)  
encoder\_state = [state\_h, state\_c]  
  
decoder\_input = layers.Input(shape=(None,))  
decoder\_embedded = layers.Embedding(input\_dim=decoder\_vocab, output\_dim=64)(  
 decoder\_input  
)  
  
# Pass the 2 states to a new LSTM layer, as initial state  
decoder\_output = layers.LSTM(64, name="decoder")(  
 decoder\_embedded, initial\_state=encoder\_state  
)  
output = layers.Dense(10)(decoder\_output)  
  
model = keras.Model([encoder\_input, decoder\_input], output)  
model.summary()  
  
  
paragraph1 = np.random.random((20, 10, 50)).astype(np.float32)  
paragraph2 = np.random.random((20, 10, 50)).astype(np.float32)  
paragraph3 = np.random.random((20, 10, 50)).astype(np.float32)  
  
lstm\_layer = layers.LSTM(64, stateful=True)  
output = lstm\_layer(paragraph1)  
output = lstm\_layer(paragraph2)  
output = lstm\_layer(paragraph3)  
  
lstm\_layer.reset\_states()  
  
  
  
paragraph1 = np.random.random((20, 10, 50)).astype(np.float32)  
paragraph2 = np.random.random((20, 10, 50)).astype(np.float32)  
paragraph3 = np.random.random((20, 10, 50)).astype(np.float32)  
  
lstm\_layer = layers.LSTM(64, stateful=True)  
output = lstm\_layer(paragraph1)  
output = lstm\_layer(paragraph2)  
  
existing\_state = lstm\_layer.states  
  
new\_lstm\_layer = layers.LSTM(64)  
new\_output = new\_lstm\_layer(paragraph3, initial\_state=existing\_state)  
  
  
  
  
model = keras.Sequential()  
  
model.add(  
 layers.Bidirectional(layers.LSTM(64, return\_sequences=True), input\_shape=(5, 10))  
)  
model.add(layers.Bidirectional(layers.LSTM(32)))  
model.add(layers.Dense(10))  
  
model.summary()  
  
  
batch\_size = 64  
input\_dim = 28  
  
units = 64  
output\_size = 10

def build\_model(allow\_cudnn\_kernel=True):  
 if allow\_cudnn\_kernel:  
   
 lstm\_layer = keras.layers.LSTM(units, input\_shape=(None, input\_dim))  
 else:  
 lstm\_layer = keras.layers.RNN(  
 keras.layers.LSTMCell(units), input\_shape=(None, input\_dim)  
 )  
 model = keras.models.Sequential(  
 [  
 lstm\_layer,  
 keras.layers.BatchNormalization(),  
 keras.layers.Dense(output\_size),  
 ]  
 )  
 return model  
  
  
mnist = keras.datasets.mnist  
  
(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()  
x\_train, x\_test = x\_train / 255.0, x\_test / 255.0  
sample, sample\_label = x\_train[0], y\_train[0]  
  
  
model = build\_model(allow\_cudnn\_kernel=True)  
  
model.compile(  
 loss=keras.losses.SparseCategoricalCrossentropy(from\_logits=True),  
 optimizer="sgd",  
 metrics=["accuracy"],  
)  
  
  
model.fit(  
 x\_train, y\_train, validation\_data=(x\_test, y\_test), batch\_size=batch\_size, epochs=1  
)  
  
  
  
noncudnn\_model = build\_model(allow\_cudnn\_kernel=False)  
noncudnn\_model.set\_weights(model.get\_weights())  
noncudnn\_model.compile(  
 loss=keras.losses.SparseCategoricalCrossentropy(from\_logits=True),  
 optimizer="sgd",  
 metrics=["accuracy"],  
)  
noncudnn\_model.fit(  
 x\_train, y\_train, validation\_data=(x\_test, y\_test), batch\_size=batch\_size, epochs=1  
)  
  
  
import matplotlib.pyplot as plt  
  
with tf.device("CPU:0"):  
 cpu\_model = build\_model(allow\_cudnn\_kernel=True)  
 cpu\_model.set\_weights(model.get\_weights())  
 result = tf.argmax(cpu\_model.predict\_on\_batch(tf.expand\_dims(sample, 0)), axis=1)  
 print(  
 "Predicted result is: %s, target result is: %s" % (result.numpy(), sample\_label)  
 )  
 plt.imshow(sample, cmap=plt.get\_cmap("gray"))  
  
  
  
class NestedCell(keras.layers.Layer):  
 def \_\_init\_\_(self, unit\_1, unit\_2, unit\_3, \*\*kwargs):  
 self.unit\_1 = unit\_1  
 self.unit\_2 = unit\_2  
 self.unit\_3 = unit\_3  
 self.state\_size = [tf.TensorShape([unit\_1]), tf.TensorShape([unit\_2, unit\_3])]  
 self.output\_size = [tf.TensorShape([unit\_1]), tf.TensorShape([unit\_2, unit\_3])]  
 super(NestedCell, self).\_\_init\_\_(\*\*kwargs)  
  
 def build(self, input\_shapes):  
   
 i1 = input\_shapes[0][1]  
 i2 = input\_shapes[1][1]  
 i3 = input\_shapes[1][2]  
  
 self.kernel\_1 = self.add\_weight(  
 shape=(i1, self.unit\_1), initializer="uniform", name="kernel\_1"  
 )  
 self.kernel\_2\_3 = self.add\_weight(  
 shape=(i2, i3, self.unit\_2, self.unit\_3),  
 initializer="uniform",  
 name="kernel\_2\_3",  
 )  
  
 def call(self, inputs, states):  
   
 input\_1, input\_2 = tf.nest.flatten(inputs)  
 s1, s2 = states  
  
 output\_1 = tf.matmul(input\_1, self.kernel\_1)  
 output\_2\_3 = tf.einsum("bij,ijkl->bkl", input\_2, self.kernel\_2\_3)  
 state\_1 = s1 + output\_1  
 state\_2\_3 = s2 + output\_2\_3  
  
 output = (output\_1, output\_2\_3)  
 new\_states = (state\_1, state\_2\_3)  
  
 return output, new\_states  
  
 def get\_config(self):  
 return {"unit\_1": self.unit\_1, "unit\_2": unit\_2, "unit\_3": self.unit\_3}  
  
  
  
unit\_1 = 10  
unit\_2 = 20  
unit\_3 = 30  
  
i1 = 32  
i2 = 64  
i3 = 32  
batch\_size = 64  
num\_batches = 10  
timestep = 50  
  
cell = NestedCell(unit\_1, unit\_2, unit\_3)  
rnn = keras.layers.RNN(cell)  
  
input\_1 = keras.Input((None, i1))  
input\_2 = keras.Input((None, i2, i3))  
  
outputs = rnn((input\_1, input\_2))  
  
model = keras.models.Model([input\_1, input\_2], outputs)  
  
model.compile(optimizer="adam", loss="mse", metrics=["accuracy"])  
  
  
  
input\_1\_data = np.random.random((batch\_size \* num\_batches, timestep, i1))  
input\_2\_data = np.random.random((batch\_size \* num\_batches, timestep, i2, i3))  
target\_1\_data = np.random.random((batch\_size \* num\_batches, unit\_1))  
target\_2\_data = np.random.random((batch\_size \* num\_batches, unit\_2, unit\_3))  
input\_data = [input\_1\_data, input\_2\_data]  
target\_data = [target\_1\_data, target\_2\_data]  
  
model.fit(input\_data, target\_data, batch\_size=batch\_size)

**Output for RNN\_LSTM implementation on MNIST digit dataset:**

\MEET PARMAR\PycharmProjects\Internship\_project\venv\Scripts\python.exe" "C:/Users/MEET PARMAR/PycharmProjects/Internship\_project/venv/RNN\_LSTM.py"

2022-12-27 08:48:47.918388: I tensorflow/core/platform/cpu\_feature\_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

Model: "sequential"

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

Layer (type) Output Shape Param #

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

embedding (Embedding) (None, None, 64) 64000

lstm (LSTM) (None, 128) 98816

dense (Dense) (None, 10) 1290

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

Total params: 164,106

Trainable params: 164,106

Non-trainable params: 0

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

Model: "sequential\_1"

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

Layer (type) Output Shape Param #

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

embedding\_1 (Embedding) (None, None, 64) 64000

gru (GRU) (None, None, 256) 247296

simple\_rnn (SimpleRNN) (None, 128) 49280

dense\_1 (Dense) (None, 10) 1290

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

Total params: 361,866

Trainable params: 361,866

Non-trainable params: 0

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

Model: "model"

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

Layer (type) Output Shape Param # Connected to

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

input\_1 (InputLayer) [(None, None)] 0 []

input\_2 (InputLayer) [(None, None)] 0 []

embedding\_2 (Embedding) (None, None, 64) 64000 ['input\_1[0][0]']

embedding\_3 (Embedding) (None, None, 64) 128000 ['input\_2[0][0]']

encoder (LSTM) [(None, 64), 33024 ['embedding\_2[0][0]']

(None, 64),

(None, 64)]

decoder (LSTM) (None, 64) 33024 ['embedding\_3[0][0]',

'encoder[0][1]',

'encoder[0][2]']

dense\_2 (Dense) (None, 10) 650 ['decoder[0][0]']

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

Total params: 258,698

Trainable params: 258,698

Non-trainable params: 0

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

Model: "sequential\_2"

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

Layer (type) Output Shape Param #

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

bidirectional (Bidirectiona (None, 5, 128) 38400

l)

bidirectional\_1 (Bidirectio (None, 64) 41216

nal)

dense\_3 (Dense) (None, 10) 650

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

Total params: 80,266

Trainable params: 80,266

Non-trainable params: 0

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

938/938 [==============================] - 29s 28ms/step - loss: 0.9331 - accuracy: 0.7027 - val\_loss: 0.5613 - val\_accuracy: 0.8140

938/938 [==============================] - 28s 28ms/step - loss: 0.3844 - accuracy: 0.8828 - val\_loss: 0.2898 - val\_accuracy: 0.9106

Predicted result is: [3], target result is: 5

10/10 [==============================] - 5s 435ms/step - loss: 0.7742 - rnn\_1\_loss: 0.2918 - rnn\_1\_1\_loss: 0.4823 - rnn\_1\_accuracy: 0.1297 - rnn\_1\_1\_accuracy: 0.0335

Process finished with exit code 0

**Average Accuracy – 91.06%**

**Code for testing accuracy of ANN on MNIST digits dataset:**

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
# Ignore the warnings  
import warnings  
warnings.filterwarnings("ignore")  
#loading MNIST dataset  
from tensorflow.keras.datasets import mnist  
(X\_train,y\_train) , (X\_test,y\_test)=mnist.load\_data()  
#visualizing the image in train data  
plt.imshow(X\_train[0])  
#visualizing the first 20 images in the dataset  
for i in range(20):  
 #subplot  
 plt.subplot(5, 5, i+1)  
 # plotting pixel data  
 plt.imshow(X\_train[i], cmap=plt.get\_cmap('gray'))  
# show the figure  
plt.show()  
X\_train\_flat=X\_train.reshape(len(X\_train),28\*28)  
X\_test\_flat=X\_test.reshape(len(X\_test),28\*28)  
#checking the shape after flattening  
print(X\_train\_flat.shape)  
print(X\_test\_flat.shape)  
X\_train\_flat=X\_train\_flat/255  
X\_test\_flat=X\_test\_flat/255  
  
  
##importing necessary libraries  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense  
#Step 1 : Defining the model  
model=Sequential()  
model.add(Dense(10,input\_shape=(784,),activation='softmax'))  
#Step 2: Compiling the model  
model.compile(loss='sparse\_categorical\_crossentropy',optimizer='adam',metrics=['accuracy'])  
#Step 3: Fitting the model  
model.fit(X\_train\_flat,y\_train,epochs=10)  
model.evaluate(X\_test\_flat,y\_test)  
y\_predict = model.predict(X\_test\_flat)  
y\_predict[3] #printing the 3rd index

**Output of ANN implementation on MNIST digits dataset:**

"C:\Users\MEET PARMAR\PycharmProjects\Internship\_project\venv\Scripts\python.exe" "C:/Users/MEET PARMAR/PycharmProjects/Internship\_project/venv/Internship\_ANN\_implementation.py"

(60000, 784)

(10000, 784)

2022-12-27 08:59:46.945564: I tensorflow/core/platform/cpu\_feature\_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

Epoch 1/10

1875/1875 [==============================] - 5s 2ms/step - loss: 0.4721 - accuracy: 0.8763

Epoch 2/10

1875/1875 [==============================] - 5s 3ms/step - loss: 0.3038 - accuracy: 0.9155

Epoch 3/10

1875/1875 [==============================] - 5s 3ms/step - loss: 0.2831 - accuracy: 0.9211

Epoch 4/10

1875/1875 [==============================] - 5s 3ms/step - loss: 0.2729 - accuracy: 0.9237

Epoch 5/10

1875/1875 [==============================] - 5s 3ms/step - loss: 0.2661 - accuracy: 0.9257

Epoch 6/10

1875/1875 [==============================] - 5s 3ms/step - loss: 0.2617 - accuracy: 0.9274

Epoch 7/10

1875/1875 [==============================] - 5s 2ms/step - loss: 0.2583 - accuracy: 0.9291

Epoch 8/10

1875/1875 [==============================] - 5s 3ms/step - loss: 0.2550 - accuracy: 0.9293

Epoch 9/10

1875/1875 [==============================] - 5s 3ms/step - loss: 0.2529 - accuracy: 0.9295

Epoch 10/10

1875/1875 [==============================] - 5s 3ms/step - loss: 0.2512 - accuracy: 0.9307

313/313 [==============================] - 1s 2ms/step - loss: 0.2637 - accuracy: 0.9276

313/313 [==============================] - 1s 2ms/step

Process finished with exit code 0

**Average Accuracy – 92.76%**