Machine Learning Lab Assignment - 5a

Name: Sri Sai Vijaya Aditya Nittala

Roll No.: 177163

Section: A

Handwritten character recognition using Convolutional Neural Networks

```
## MOUNTING GOOGLE DRIVE
from google.colab import drive
drive.mount('/content/drive')
```

```
     Mounted at /content/drive
```

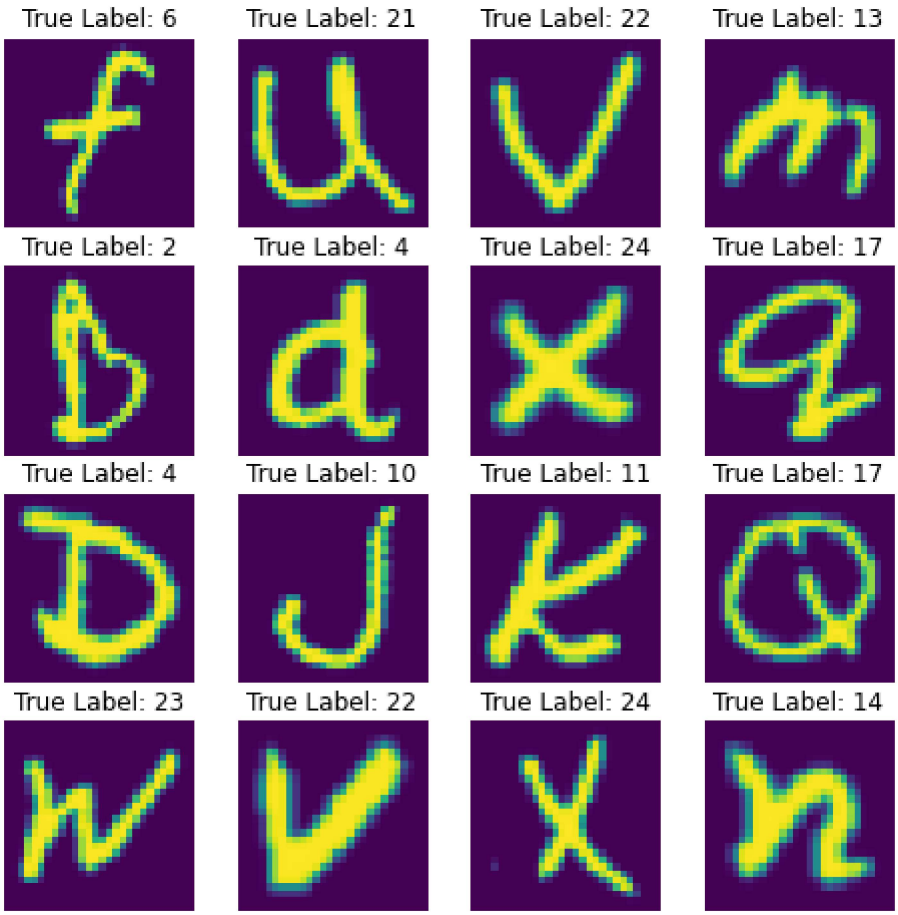
```
## IMPORTING RELEVANT LIBRARIES
import pandas as pd
import numpy as np
from sklearn import preprocessing
import json
import matplotlib.pyplot as plt
from sklearn import model_selection
```

```
## IMPORTING RELEVANT PACKAGES FOR THE DATASET AS WELL AS THE LAYERS FOR THE CNN MODEL
from keras.datasets import mnist
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.utils import np_utils
```

```
## LOADING TRAINING DATA
data = pd.read_csv('/content/drive/MyDrive/emnist.csv')
y = np.array(data['label'])
X = np.array(data.drop(['label'], axis = 1))

## SPLITTING MAIN DATASET INTO TRAINING AND TESTING DATASETS
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size = 0.2, ra

## PRINTING IMAGES FROM THE EMNIST DATASET
print("Sample of images from the EMNIST Dataset : ")
plt.figure(figsize=(8, 8))
for i in range(16):
    plt.subplot(4, 4, i+1)
    plt.axis('off')
```

```
    r = np.random.randint(X_train.shape[0])   ## PICK A RANDON IMAGE TO SHOW
    plt.title('True Label: '+ str(y_train[r])) ## PRINT LABEL
    plt.imshow(X_train[r].reshape(28, 28).T)     ## PRINT IMAGE
plt.show()
```

```
Sample of images from the EMNIST Dataset :
```




```
## NORMALIZING THE PIXEL VALUES FOR TRAINING DATA
X_train = X_train / 255

## CONVERTING TARGET VALUE TO ONE-HOT ENCODED FORM
X_train = np.array(X_train)
b = np.zeros((y_train.size, y_train.max()+1))
b[np.arange(y_train.size), y_train] = 1
y_train = np.array(b)

X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32')
print("y_train shape (after one-hot encoding) : {}".format(y_train.shape))
```

```
y_train shape (after one-hot encoding) : (5598, 27)
```

```
## NORMALIZING THE PIXEL VALUES FOR TESTING DATA
X_test = X_test / 255

## CONVERTING TARGET VALUE TO ONE-HOT ENCODED FORM
```

```
X_test = np.array(X_test)
b1 = np.zeros((y_test.size, 27))
b1[np.arange(y_test.size), y_test] = 1
y_test = np.array(b1)

X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')
print("y_test shape (after one-hot encoding) : {}".format(y_test.shape))
```

```
y_test shape (after one-hot encoding) : (1400, 27)
```

```
num_classes = y_train.shape[1]
print("Number of classes : {}".format(num_classes))
```

```
Number of classes : 27
```

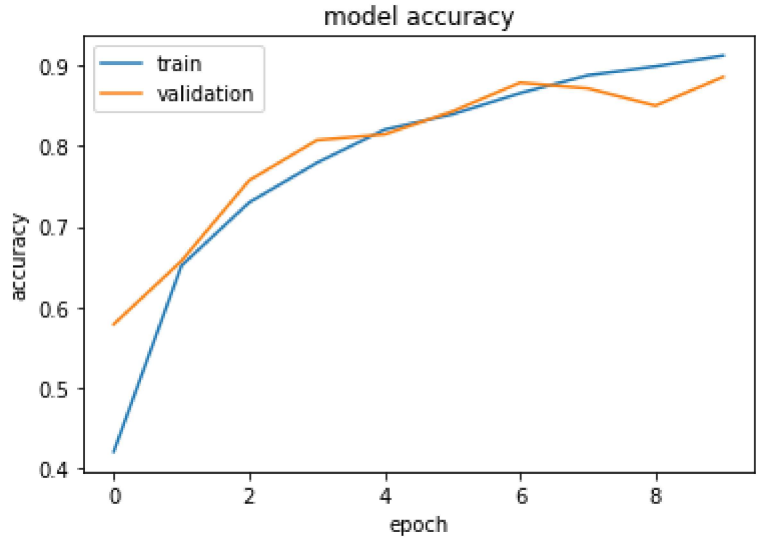
```
## CREATE MODEL AND COMPILE
def baseline_model():
    ## CREATE MODEL
    model = Sequential()
    model.add(Conv2D(32, (5, 5), input_shape=(28, 28, 1), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    ## COMPILE MODEL
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model


## SPLITTING INTO VALIDATION SET
X_test, X_val, y_test, y_val = model_selection.train_test_split(X_test, y_test, test_size=0.1


## BUILDING THE MODEL
model = baseline_model()
## TRAINING THE MODEL
history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=10, batch_size=2
```

```
Epoch 1/10
28/28 [==============================] - 3s 98ms/step - loss: 2.1150 - accuracy: 0.4203
Epoch 2/10
28/28 [==============================] - 3s 93ms/step - loss: 1.1615 - accuracy: 0.6513
Epoch 3/10
28/28 [==============================] - 3s 92ms/step - loss: 0.9076 - accuracy: 0.7297
Epoch 4/10
28/28 [==============================] - 3s 91ms/step - loss: 0.7369 - accuracy: 0.7790
Epoch 5/10
28/28 [==============================] - 3s 93ms/step - loss: 0.6150 - accuracy: 0.8203
Epoch 6/10
28/28 [==============================] - 3s 91ms/step - loss: 0.5238 - accuracy: 0.8392
Epoch 7/10
```

```
28/28 [==============================] - 3s 90ms/step - loss: 0.4482 - accuracy: 0.8655
Epoch 8/10
28/28 [==============================] - 3s 91ms/step - loss: 0.3778 - accuracy: 0.8876
Epoch 9/10
28/28 [==============================] - 3s 91ms/step - loss: 0.3341 - accuracy: 0.8987
Epoch 10/10
28/28 [==============================] - 3s 92ms/step - loss: 0.2960 - accuracy: 0.9121
```

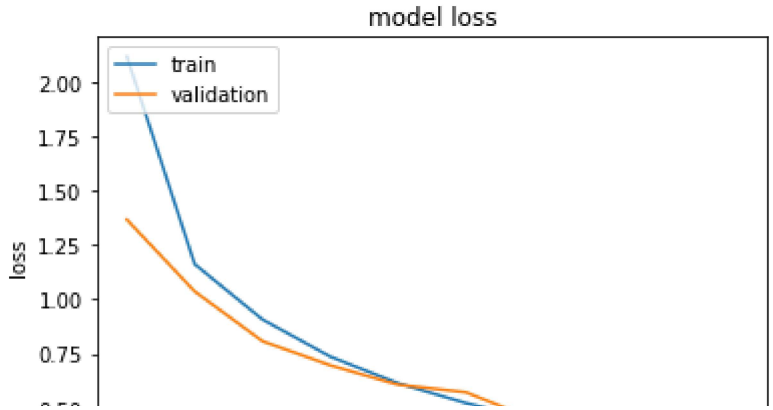
```
print("Training and Cross-validation accuracy with respect to epochs : ")
## ACCURACY VS ITERATIONS GRAPH
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

```
Training and Cross-validation accuracy with respect to epochs :
```




```
print("Model loss with respect to epochs : ")
## LOSS VS ITERATIONS GRAPH
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

Model loss with respect to epochs :




```
## EVALUATING THE MODEL USING TEST DATA
print("Evaluate on test data")
results = model.evaluate(X_test, y_test, batch_size=200)
print("\nTest loss : {}\tTest Accuracy : {}".format(results[0], results[1]))
## PRINTING WEIGHTS
print("\nWeights : {}".format(model.get_weights()))
```

```
Evaluate on test data
7/7 [==============================] - 0s 20ms/step - loss: 0.6018 - accuracy: 0.81

Test loss : 0.6018382906913757  Test Accuracy : 0.8158730268478394

Weights : [array([[[[ 0.07559877, -0.04684798,  0.07995688, -0.06146776,
           0.0455247 , -0.10545624,  0.09506586, -0.03397514,
          -0.16819324,  0.1412148 , -0.0628755 , -0.06410927,
          -0.18496703,  0.05587763, -0.09944803, -0.02207323,
          -0.00628331, -0.16165195, -0.0258592 , -0.0560321 ,
          -0.11447994,  0.09533546, -0.07121418, -0.12385387,
          -0.13440451, -0.09396073, -0.0555824 ,  0.04289359,
          -0.06891114, -0.00474438, -0.19472171, -0.05708386]],

         [[ 0.00832295,  0.0705397 ,  0.05486947,  0.15743311,
           0.11634281, -0.08149445,  0.01779323,  0.03750779,
          -0.20303097,  0.11686318,  0.09643816, -0.02818695,
          -0.14495453,  0.05041317,  0.16592014, -0.1058208 ,
           0.05478789, -0.07739001,  0.11592869, -0.05600768,
          -0.17641965,  0.05165926, -0.0536009 , -0.05931795,
          -0.04454161, -0.07304637, -0.00296392, -0.07516164,
           0.00121478,  0.02622432, -0.1646248 , -0.15751138]],

         [[ 0.11280087, -0.07659526,  0.03588122,  0.08594591,
          -0.02755181,  0.09242579, -0.00354247,  0.0776219 ,
          -0.19833672,  0.12535606,  0.11461823, -0.01851487,
          -0.1653173 , -0.09240803,  0.00969311, -0.14872923,
           0.07010653,  0.13163842,  0.02264245,  0.00096579,
          -0.21174835, -0.04004114, -0.09401277, -0.04960692,
           0.09989455,  0.06613312,  0.06549864,  0.12012096,
           0.15599738,  0.10503724,  0.0683293 , -0.06156811]],

         [[-0.07218449,  0.04188294,  0.0087297 , -0.03849425,
           0.03146308,  0.04155924, -0.03357976,  0.03507708,
          -0.11468395,  0.08409971, -0.03331769,  0.07177191,
```

```
   -0.18843424, -0.02602954, -0.1096208 , -0.23433733,
    0.02519816,  0.02184828, -0.06670864,  0.06821129,
   -0.09128402, -0.10911772, -0.05816105,  0.03153408,
    0.11722262,  0.06546008,  0.09753442,  0.13247652,
    0.20644124,  0.04107277,  0.01588576, -0.08088251]],

  [[-0.04902107, -0.02867775, -0.05932061,  0.005052  ,
    -0.07844762,  0.1115532 , -0.09221043,  0.06526336,
    -0.02569502,  0.14263824, -0.02475065,  0.04760588,
    -0.18480095,  0.09459718, -0.09419844, -0.17238142,
     0.05197101,  0.12991612, -0.20994754,  0.09842738,
    -0.10904741, -0.14598969, -0.17121355,  0.15581207,
     0.05001749,  0.06815002,  0.01114487,  0.12716112,
     0.15318893, -0.01537151,  0.10222033, -0.08397155]]],


 [[[-0.11796667,  0.05501288,  0.04316818,  0.07738925,
    -0.00525875,  0.07275213,  0.06041266,  0.10391583,
    -0.00107907,  0.09410807, -0.04571626,  0.06783501,
    -0.12825339,  0.0617684 , -0.09510325,  0.08048938,
    -0.06210828, -0.1844124 ,  0.02295813,  0.11932912,
     0.1120613 ,  0.12072224,  0.00185034,  0.00552322,
```