# This code was compiled by individual diligence because we did not find scientific papers that process data gloves using the convolutional neural network algorithm #

# The libraries #

import os

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.metrics import classification\_report

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import f1\_score

from sklearn.metrics import confusion\_matrix

from keras.utils.np\_utils import to\_categorical

from sklearn.utils import class\_weight

import warnings

warnings.filterwarnings('ignore')

# Enter the data #

mit\_test\_data = pd.read\_csv("/content/drive/MyDrive/Colab Notebooks/MY DATA NOMBER.CSV", header=None)

mit\_train\_data = pd.read\_csv("/content/drive/MyDrive/الداتا بدل حروف أرقام.csv", header=None)

# Data display #

print("MIT test dataset")

print(mit\_test\_data.info())

print("MIT train dataset")

print(mit\_train\_data.info())

# Taking random samples from the test data and displaying the shape of the signal as well as classifying them #

# take a random distribution

sample = mit\_test\_data.sample(20)

# remove the target column

sampleX = sample.iloc[:,sample.columns != 16]

import matplotlib.pyplot as plt

plt.style.use('classic')

# plt samples

for index, row in sampleX.iterrows():

    plt.plot(np.array(range(0, 16)) ,row)

plt.xlabel("time")

plt.ylabel("magnitude")

plt.title("Sign language sign format \nrandom sample")

plt.show()

plt.style.use("ggplot")

plt.title("Number of record in each category")

plt.hist(sample.iloc[:,sample.columns == 16].transpose())

plt.show()

# Show the output of the data #

print("Train data")

print("Type\tCount")

print((mit\_train\_data[16]).value\_counts())

print("-------------------------")

print("Test data")

print("Type\tCount")

print((mit\_test\_data[16]).value\_counts())

# Classifying the data set again to display the detailed information for each letter and its number, as well as the shape of its sign later #

# randomly sampling from each class

classes=mit\_train\_data.groupby(16,group\_keys=False).apply(lambda mit\_train\_data : mit\_train\_data.sample(1))

# peek on classes

print (classes)

# The sign of each letter #

# plotting classes SLR

plt.figure(figsize=(22,16))

# D

plt.subplot(4, 7, 1)

plt.plot(classes.iloc[0,:16])

plt.title('D')

# X

plt.subplot(4,7, 2)

plt.plot(classes.iloc[1,:16])

plt.title('X')

# S

plt.subplot(4, 7, 3)

plt.plot(classes.iloc[2,:16])

plt.title('S')

#B

plt.subplot(4, 7, 4)

plt.plot(classes.iloc[3,:16])

plt.title('B')

# H

plt.subplot(4, 7, 5)

plt.plot(classes.iloc[4,:16])

plt.title('H')

# C

plt.subplot(4, 7, 6)

plt.plot(classes.iloc[5,:16])

plt.title('C')

# K

plt.subplot(4, 7, 7)

plt.plot(classes.iloc[6,:16])

plt.title('K')

# M

plt.subplot(4, 7, 8)

plt.plot(classes.iloc[7,:16])

plt.title('M')

# A

plt.subplot(4, 7, 9)

plt.plot(classes.iloc[8,:16])

plt.title('A')

# Z

plt.subplot(4, 7, 10)

plt.plot(classes.iloc[9,:16])

plt.title('Z')

# R

plt.subplot(4, 7, 11)

plt.plot(classes.iloc[10,:16])

plt.title('R')

# F

plt.subplot(4, 7, 12)

plt.plot(classes.iloc[11,:16])

plt.title('F')

# T

plt.subplot(4, 7, 13)

plt.plot(classes.iloc[12,:16])

plt.title('T')

# O

plt.subplot(4, 7, 14)

plt.plot(classes.iloc[13,:16])

plt.title('O')

# N

plt.subplot(4, 7, 15)

plt.plot(classes.iloc[14,:16])

plt.title('N')

# Y

plt.subplot(4, 7, 16)

plt.plot(classes.iloc[15,:16])

plt.title('Y')

# V

plt.subplot(4, 7, 17)

plt.plot(classes.iloc[16,:16])

plt.title('V')

# Q

plt.subplot(4, 7, 18)

plt.plot(classes.iloc[17,:16])

plt.title('Q')

# G

plt.subplot(4, 7, 19)

plt.plot(classes.iloc[18,:16])

plt.title('G')

# I

plt.subplot(4, 7, 20)

plt.plot(classes.iloc[19,:16])

plt.title('I')

# P

plt.subplot(4, 7, 21)

plt.plot(classes.iloc[20,:16])

plt.title('P')

# J

plt.subplot(4, 7, 22)

plt.plot(classes.iloc[21,:16])

plt.title('J')

# W

plt.subplot(4, 7, 23)

plt.plot(classes.iloc[22,:16])

plt.title('W')

# E

plt.subplot(4, 7, 24)

plt.plot(classes.iloc[23,:16])

plt.title('E')

# L

plt.subplot(4, 7, 25)

plt.plot(classes.iloc[24,:16])

plt.title('L')

# U

plt.subplot(4, 7, 26)

plt.plot(classes.iloc[25,:16])

plt.title('U')

# show plot

plt.show()

# Determine the input and output of the data and the data test for later training.

Where the input is specified from column No. 0 to column No. 15, and the output is column No. 16. #

from keras.utils import to\_categorical

print("--- X ---")

X = mit\_train\_data.loc[:, mit\_train\_data.columns != 16]

print(X.head())

print(X.info())

print("--- Y ---")

y = mit\_train\_data.loc[:, mit\_train\_data.columns == 16]

y = to\_categorical(y)

print("--- testX ---")

testX = mit\_test\_data.loc[:, mit\_test\_data.columns != 16]

print(testX.head())

print(testX.info())

print("--- testy ---")

testy = mit\_test\_data.loc[:, mit\_test\_data.columns == 16]

testy = to\_categorical(testy)

# specify the format of the input signal - (to show that we are processing a one-dimensional signal) D1, not two-dimensional D2 as in the pictures #

X=mit\_train\_data.iloc[:,:16].values

testX=mit\_test\_data.iloc[:,:16].values

#for i in range(len(X\_train)):

#    X\_train[i,:16]= add\_gaussian\_noise(X\_train[i,:16])

X1 = X.reshape(len(X), X.shape[1],1)

testX1 = testX.reshape(len(testX),testX.shape[1],1)

# Some libraries that we will need in network configuration #

from keras.models import Sequential

from keras.layers import Dense,MaxPool1D , Dropout, Flatten, MaxPooling1D,Conv1D, ,AveragePooling1D,GlobalAveragePooling1D

from keras.layers import Dropout

from keras.layers import Dense, Convolution1D, MaxPool1D, Flatten, Dropout

from keras.layers import Input

from keras.models import Model

from keras.layers import BatchNormalization

import keras

from keras.callbacks import EarlyStopping, ModelCheckpoint

You may need to install some libraries, as this is done using the following command:

For example, but not limited to, I will install the **normalization** library #

!pip install normalization

# The neural network has been configured, and as I mentioned earlier because there is no reference, we built it ourselves, and we also added a layer to reduce the overfitting #

model = Sequential()

model.add(Conv1D(filters=512, kernel\_size=3, activation='LeakyReLU', padding='same', input\_shape=(X1.shape[1],1)))

model.add(BatchNormalization())

model.add(MaxPooling1D(pool\_size=(2), strides=(2)))

model.add(Dropout(0.6))

model.add(Conv1D(filters=512, kernel\_size=2, activation='LeakyReLU', padding='same'))

model.add(BatchNormalization())

model.add(MaxPooling1D(pool\_size=(2), strides=(2)))

model.add(Dropout(0.5))

model.add(Conv1D(filters=512, kernel\_size=3, activation='LeakyReLU', padding='same'))

model.add(BatchNormalization())

model.add(MaxPooling1D(pool\_size=(2), strides=(2)))

model.add(Dropout(0.4))

#model.add(Conv1D(filters=512, kernel\_size=2, activation='relu', padding='same'))

#model.add(BatchNormalization())

#model.add(MaxPooling1D(pool\_size=(2), strides=(2)))

#model.add(Dropout(0.25))

#model.add(GlobalAveragePooling1D())

model.add(Flatten())

model.add(Dense(512, activation='LeakyReLU'))

model.add(Dropout(0.6))

model.add(Dense(256, activation='LeakyReLU'))

model.add(Dropout(0.5))

model.add(Dense(128, activation='LeakyReLU'))

model.add(Dropout(0.4))

#model.add(Dense(256, activation='relu'))

#model.add(Dropout(0.6))

#model.add(Dense(128, activation='relu'))

#model.add(Dropout(0.5))

#model.add(Dense(64, activation='relu'))

#model.add(Dropout(0.5))

model.add(Dense(27, activation='softmax'))

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

random\_state =0

# View a summary of the network that we've built #

model.summary()

# This code (No. 1) is for showing the accuracy, as well as the accuracy chart and the error chart, but I preferred to detail these codes again (serving the same purpose) for more clarity #

def evaluate\_model(history,testX1,testy,model):

    scores = model.evaluate((testX1),testy, verbose=0)

    print("Accuracy: %.2f%%" % (scores[1]\*100))

    print(history)

    fig1, ax\_acc = plt.subplots()

    plt.plot(history.history['accuracy'])

    plt.plot(history.history['val\_accuracy'])

    plt.xlabel('Epoch')

    plt.ylabel('Accuracy')

    plt.title('Model - Accuracy')

    plt.legend(['Training', 'Validation'], loc='lower right')

    plt.show()

    fig2, ax\_loss = plt.subplots()

    plt.xlabel('Epoch')

    plt.ylabel('Loss')

    plt.title('Model- Loss')

    plt.legend(['Training', 'Validation'], loc='upper right')

    plt.plot(history.history['loss'])

    plt.plot(history.history['val\_loss'])

    plt.show()

    target\_names=['0','1','2','3','4']

    y\_true=[]

    for element in testy:

        y\_true.append(np.argmax(element))

    prediction\_proba=model.predict(testX1)

    prediction=np.argmax(prediction\_proba,axis=1)

    cnf\_matrix = confusion\_matrix(y\_true, prediction)

# This code is for initiating training as well as testing and validating training #

history = model.fit(X1, y, batch\_size=512, epochs=100, validation\_data=(testX1,testy))

#This code is for displaying the Validation Accuracy #

#print('Validation\_Accuracy =')

Acc=model.evaluate(testX1,testy,verbose=0)

print("Validation\_Accuracy = %.2f%%" % (Acc[1]\*100))

# This code is for displaying the Training Accuracy #

#print('Training\_Accuracy =')

Acc1=model.evaluate(X1,y,verbose=0)

print("Training\_Accuracy = %.2f%%" % (Acc1[1]\*100))

# This code is a follower of code number 1 that I talked about. It tracks the Validation Accuracy and gives the schematics directly #

evaluate\_model(history,testX1,testy,model)

y\_pred=model.predict(testX1)

# This code is a follower of code number 1 that I talked about. It tracks the Training Accuracy and gives the same schematics. #

evaluate\_model(history,X1,y,model)

y\_pred=model.predict(X1)

#This code gives the accuracy chart #

import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Val'], loc='lower right')

plt.show()

# This code gives the error chart #

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Val'], loc='upper right')

plt.show()

# This code gives the truth outline #

import itertools

def plot\_confusion\_matrix(cm, classes,

                          normalize=False,

                          title='Confusion matrix',

                          cmap=plt.cm.Blues):

    """

    This function prints and plots the confusion matrix.

    Normalization can be applied by setting `normalize=True`.

    """

    if normalize:

        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

        print("Normalized confusion matrix")

    else:

        print('Confusion matrix, without normalization')

    plt.imshow(cm, interpolation='nearest', cmap=cmap)

    plt.title(title)

    plt.colorbar()

    tick\_marks = np.arange(len(classes))

    plt.xticks(tick\_marks, classes, rotation=45)

    plt.yticks(tick\_marks, classes)

    fmt = '.2f' if normalize else 'd'

    thresh = cm.max() / 2.

    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

        plt.text(j, i, format(cm[i, j], fmt),

                 horizontalalignment="center",

                 color="white" if cm[i, j] > thresh else "black")

    plt.tight\_layout()

    plt.ylabel('True label')

    plt.xlabel('Predicted label')

# Compute confusion matrix

cnf\_matrix = confusion\_matrix(testy.argmax(axis=1), testX1.argmax(axis=1))

np.set\_printoptions(precision=2)

# Plot non-normalized confusion matrix

plt.figure(figsize=(10, 10))

plot\_confusion\_matrix(cnf\_matrix, classes=['N', 'S', 'V', 'F', 'Q'],normalize=True,

                      title='Confusion matrix, with normalization')

plt.show()

#This code, after adding the Drop Out layers, reducing the over-fitting, and increasing the accuracy, was completed, thanks to God, thanks to Him, and then thanks to Dr. Musab Al-Khair's exerted efforts.