Name: Mohamed Mahmoud Ahmed Amer. **ID:** 17107584.

Implementation of CNN

Dataset: Sign Language MNIST (American Sign Language).

American Sign Language (ASL) is a complete, natural language with linguistic properties like spoken languages and a grammar distinct from English. Hand and face gestures are used to convey ASL. It is the predominant language of many deaf and hard-of-hearing people in North America, as well as many hearing people. The dataset format is designed to closely resemble the classic MNIST format. Every training and test case represents a label (0-25) as a one-to-one map for each alphabetic letter A-Z (due to gesture movements, there are no cases for 9=J or 25=Z). The training data (27,455 cases) and test data (7172 cases) are around half the size of the regular MNIST, but they are otherwise identical, with a header row of label, pixel1, pixel2...pixel784 representing a single 28x28 pixel image with grayscale values ranging from 0-255. Multiple users repeating the gesture against various backgrounds were reflected in the original hand gesture image data. The Sign Language MNIST data was derived by greatly expanding the limited number (1704) of color images included in the study that were not cropped around the hand region of interest.

Dataset: https://www.kaggle.com/datamunge/sign-language-mnist.

Code:

The first part of the code simply includes all the imports that we will use in the code.

Imports

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import keras
from keras.models import Sequential
from keras.layers import Dense, Conv2D , MaxPool2D , Flatten , Dropout , BatchNormalization
from keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
from keras.callbacks import ReduceLROnPlateau
```

Then we load the dataset and show an example of what the data looks like

Loading The Dataset

```
train_df = pd.read_csv("/content/sign_mnist_train.csv")
test_df = pd.read_csv("/content/sign_mnist_test.csv")
test = pd.read_csv("/content/sign_mnist_test.csv")
y = test['label']
train_df.head()
```

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pixel13	pixel14	pixel15	pixel16	p
0	3	107	118	127	134	139	143	146	150	153	156	158	160	163	165	159	166	
1	6	155	157	156	156	156	157	156	158	158	157	158	156	154	154	153	152	
2	2	187	188	188	187	187	186	187	188	187	186	185	185	185	184	184	184	
3	2	211	211	212	212	211	210	211	210	210	211	209	207	208	207	206	203	
4	13	164	167	170	172	176	179	180	184	185	186	188	189	189	190	191	189	

5 rows × 785 columns

And as we mentioned before the training data (27,455 cases) and test data (7172 cases) with a header row of label, pixel1, pixel2...pixel784 representing a single 28x28 pixel image with grayscale values ranging from 0-255. Then we label the data. Every training and test case represents a label (0-25) as a one-to-one map for each alphabetic letter A-Z.

Labeling Data

```
[ ]  y_train = train_df['label']
     y_test = test_df['label']
     del train_df['label']
     del test_df['label']
```

```
[ ]  from sklearn.preprocessing import LabelBinarizer
     label_binarizer = LabelBinarizer()
     y_train = label_binarizer.fit_transform(y_train)
     y_test = label_binarizer.fit_transform(y_test)
```

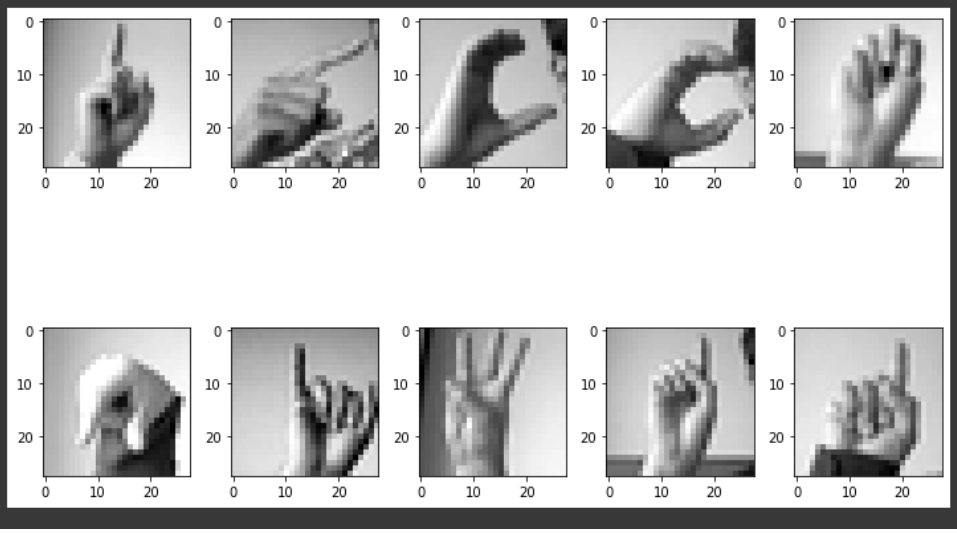
```
[ ]  x_train = train_df.values
     x_test = test_df.values
```

The next step is data normalization which helps the CNN to converge faster by preforming grayscale normalization to reduce the illumination differences and it will also converge faster since we normalize it to [0...1] data instead of [0...255].

Data Normalization

```
[ ]  x_train = x_train / 255
     x_test = x_test / 255
     x_train = x_train.reshape(-1,28,28,1)
     x_test = x_test.reshape(-1,28,28,1)
```

```
[ ]  #Showing the first 10 images
     f, ax = plt.subplots(2,5)
     f.set_size_inches(10, 10)
     k = 0
     for i in range(2):
         for j in range(5):
             ax[i,j].imshow(x_train[k].reshape(28, 28) , cmap = "gray")
             k += 1
         plt.tight_layout()
```



Next to avoid overfitting we will do some data augmentation and some of the approaches that will help with this are flipping the image randomly (horizontally or vertically), random crops, random translations, rotations, and color jitters. By applying these approaches, we will easily increase the size of the dataset which will improve the results.

Data Augmentaion (To Prevent Overfitting)

```
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images


datagen.fit(x_train)
```

Next, we will create our CNN model with our different convolution, pooling, dropout layers. To get the best results from our datasets

Creating the Model With CNN

```
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy', patience = 2, verbose=1,factor=0.5, min_lr=0.00001)
model = Sequential()
model.add(Conv2D(75 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu' , input_shape = (28,28,1)))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(50 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(25 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Flatten())
model.add(Dense(units = 512 , activation = 'relu'))
model.add(Dropout(0.3))
model.add(Dense(units = 24 , activation = 'softmax'))
model.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics = ['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 75)        750
_________________________________________________________________
batch_normalization (BatchNo (None, 28, 28, 75)        300
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 75)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 50)        33800
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 50)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 14, 14, 50)        200
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 50)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 25)          11275
_________________________________________________________________
batch_normalization_2 (Batch (None, 7, 7, 25)          100
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 25)          0
_________________________________________________________________
flatten (Flatten)            (None, 400)               0
_________________________________________________________________
dense (Dense)                (None, 512)               205312
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 24)                12312
=================================================================
Total params: 264,049
Trainable params: 263,749
Non-trainable params: 300
_________________________________________________________________
```

Next, we train our model using the model. Fit () function

Training the Model

```
[ ]  history = model.fit(datagen.flow(x_train,y_train, batch_size = 128) ,epochs = 20 ,
                         validation_data = (x_test, y_test) , callbacks = [learning_rate_reduction])
```

```
Epoch 1/20
215/215 [==============================] - 41s 38ms/step - loss: 1.8520 - accuracy: 0.4557 - val_loss: 3.5759 - val_accuracy: 0.0976
Epoch 2/20
215/215 [==============================] - 8s 37ms/step - loss: 0.2526 - accuracy: 0.9156 - val_loss: 2.2791 - val_accuracy: 0.3622
Epoch 3/20
215/215 [==============================] - 8s 37ms/step - loss: 0.1146 - accuracy: 0.9628 - val_loss: 0.1992 - val_accuracy: 0.9282
Epoch 4/20
215/215 [==============================] - 8s 36ms/step - loss: 0.0684 - accuracy: 0.9779 - val_loss: 0.0524 - val_accuracy: 0.9886
Epoch 5/20
215/215 [==============================] - 8s 37ms/step - loss: 0.0449 - accuracy: 0.9858 - val_loss: 0.0983 - val_accuracy: 0.9689
Epoch 6/20
215/215 [==============================] - 8s 37ms/step - loss: 0.0381 - accuracy: 0.9890 - val_loss: 0.1398 - val_accuracy: 0.9520

Epoch 00006: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
Epoch 7/20
215/215 [==============================] - 8s 37ms/step - loss: 0.0252 - accuracy: 0.9918 - val_loss: 0.0154 - val_accuracy: 0.9954
Epoch 8/20
215/215 [==============================] - 8s 37ms/step - loss: 0.0157 - accuracy: 0.9954 - val_loss: 0.1036 - val_accuracy: 0.9625
Epoch 9/20
215/215 [==============================] - 8s 37ms/step - loss: 0.0154 - accuracy: 0.9954 - val_loss: 0.0062 - val_accuracy: 0.9978
Epoch 10/20
215/215 [==============================] - 8s 37ms/step - loss: 0.0091 - accuracy: 0.9976 - val_loss: 0.0102 - val_accuracy: 0.9962
Epoch 11/20
215/215 [==============================] - 8s 37ms/step - loss: 0.0106 - accuracy: 0.9970 - val_loss: 0.0467 - val_accuracy: 0.9808

Epoch 00011: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
Epoch 12/20
215/215 [==============================] - 8s 37ms/step - loss: 0.0088 - accuracy: 0.9975 - val_loss: 0.0020 - val_accuracy: 1.0000
Epoch 13/20
215/215 [==============================] - 8s 37ms/step - loss: 0.0064 - accuracy: 0.9983 - val_loss: 0.0027 - val_accuracy: 1.0000
Epoch 14/20
215/215 [==============================] - 8s 37ms/step - loss: 0.0069 - accuracy: 0.9985 - val_loss: 0.0024 - val_accuracy: 0.9992

Epoch 00014: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
Epoch 15/20
215/215 [==============================] - 8s 36ms/step - loss: 0.0058 - accuracy: 0.9982 - val_loss: 9.3287e-04 - val_accuracy: 1.0000
Epoch 16/20
215/215 [==============================] - 8s 36ms/step - loss: 0.0044 - accuracy: 0.9990 - val_loss: 0.0012 - val_accuracy: 0.9999

Epoch 00016: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
Epoch 17/20
215/215 [==============================] - 8s 37ms/step - loss: 0.0040 - accuracy: 0.9990 - val_loss: 6.6606e-04 - val_accuracy: 0.9999
Epoch 18/20
215/215 [==============================] - 8s 36ms/step - loss: 0.0041 - accuracy: 0.9992 - val_loss: 4.8597e-04 - val_accuracy: 0.9999

Epoch 00018: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
Epoch 19/20
215/215 [==============================] - 8s 37ms/step - loss: 0.0030 - accuracy: 0.9995 - val_loss: 4.3281e-04 - val_accuracy: 1.0000
Epoch 20/20
215/215 [==============================] - 8s 37ms/step - loss: 0.0048 - accuracy: 0.9983 - val_loss: 5.7071e-04 - val_accuracy: 1.0000

Epoch 00020: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.
```

Next, we print the accuracy of our model.

```
print("Accuracy of the model is - " , model.evaluate(x_test,y_test)[1]*100 , "%")

225/225 [==============================] - 1s 3ms/step - loss: 5.7071e-04 - accuracy: 1.0000
Accuracy of the model is -  100.0 %
```

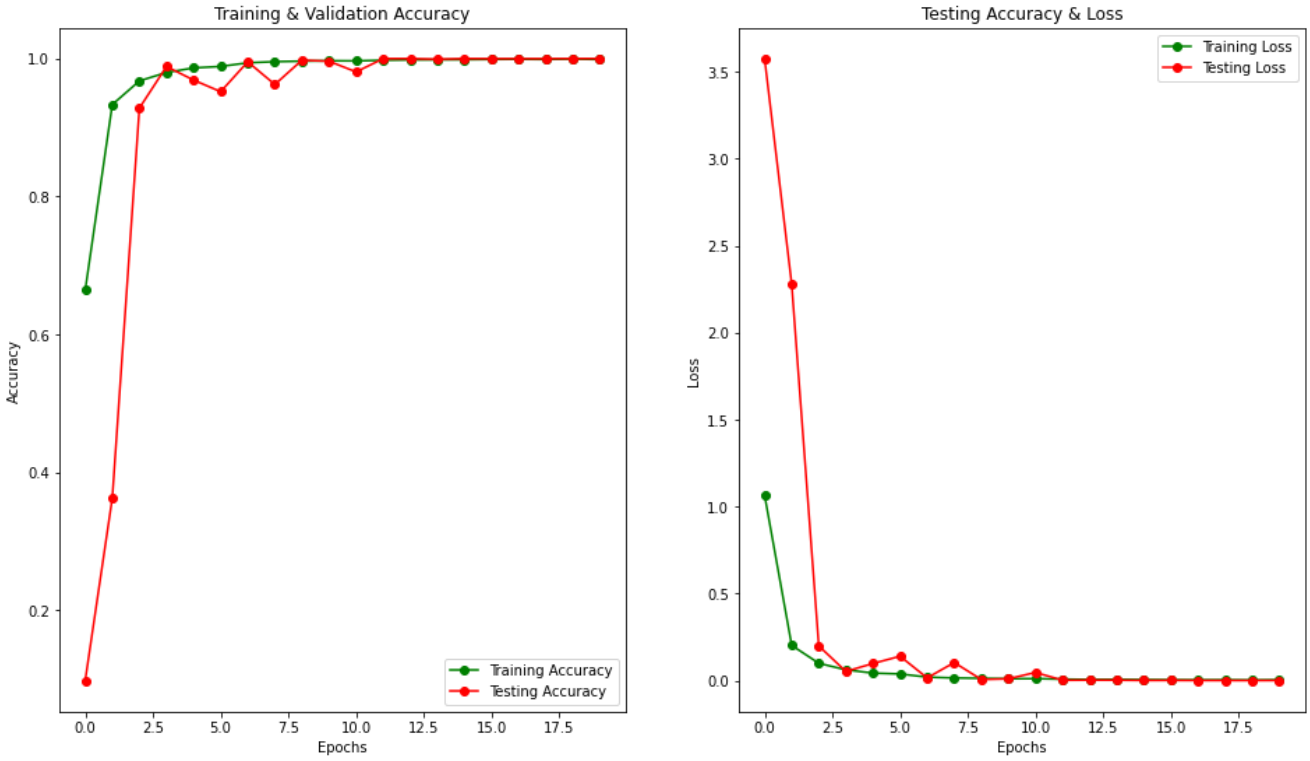
Lastly, we visualize the results to see the model accuracy.

Analysis & Visualization

```
epochs = [i for i in range(20)]
fig , ax = plt.subplots(1,2)
train_acc = history.history['accuracy']
train_loss = history.history['loss']
val_acc = history.history['val_accuracy']
val_loss = history.history['val_loss']
fig.set_size_inches(16,9)

ax[0].plot(epochs , train_acc , 'go-' , label = 'Training Accuracy')
ax[0].plot(epochs , val_acc , 'ro-' , label = 'Testing Accuracy')
ax[0].set_title('Training & Validation Accuracy')
ax[0].legend()
ax[0].set_xlabel("Epochs")
ax[0].set_ylabel("Accuracy")

ax[1].plot(epochs , train_loss , 'g-o' , label = 'Training Loss')
ax[1].plot(epochs , val_loss , 'r-o' , label = 'Testing Loss')
ax[1].set_title('Testing Accuracy & Loss')
ax[1].legend()
ax[1].set_xlabel("Epochs")
ax[1].set_ylabel("Loss")
plt.show()
```



And with the confusion matrix we can see that all the predictions were correct, and the model accuracy is 100%.

Confusion Matrix & Heat Map

```
[ ]  cm = confusion_matrix(y,predictions)
     cm = pd.DataFrame(cm , index = [i for i in range(25) if i != 9] , columns = [i for i in range(25) if i != 9])
     plt.figure(figsize = (15,15))
     sns.heatmap(cm,cmap= "Blues", linecolor = 'black' , linewidth = 1 , annot = True, fmt='')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcbc03cec90>
```