Modelling

Import the required modules ¶

In [1]:
```
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten , Dropout
from keras.layers import Dense
from keras.preprocessing.image import ImageDataGenerator
```

```
2023-04-10 19:35:30.329588: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic
library libcudart.so.10.1
```

1- Preparing the train and test data

In [2]:
```
train_data = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

test_data = ImageDataGenerator(rescale=1./255)

training_set = train_data.flow_from_directory(r'Adaptive Threshold Mean Rename/Train/',
    target_size=(96, 96),
    batch_size=10,
    color_mode='grayscale',
    class_mode='categorical'
)

testing_set = test_data.flow_from_directory(r'Adaptive Threshold Mean Rename/Test/',
    target_size=(96, 96),
    batch_size=5,
    color_mode='grayscale',
    class_mode='categorical'
)
```

```
Found 32000 images belonging to 32 classes.
Found 22400 images belonging to 32 classes.
```

2- Build the CNN model

In [3]:
```
# Initializing the CNN or Neural Network
classifier = Sequential()

# First convolution layer and pooling
classifier.add(Convolution2D(32, (3, 3), input_shape=(96, 96, 1), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))

# Avoiding 25% of the neurons to avoid overfitting of the data.
classifier.add(Dropout(0.25))

# Second convolution layer and pooling
classifier.add(Convolution2D(64, (3, 3), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))

# Third convolution layer and pooling
classifier.add(Convolution2D(128, (3, 3), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))

# Fourth convolution layer and pooling
classifier.add(Convolution2D(256, (3, 3), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))

# Flattening the layers
classifier.add(Flatten())

# Connecting all layers together.
classifier.add(Dense(units=128, activation='relu'))
classifier.add(Dense(units=32, activation='softmax'))

# Compile the CNN model.
classifier.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

classifier.fit(
        training_set,
        epochs=5,
        validation_data=testing_set
)
```

```
2023-04-10 19:38:17.132204: I tensorflow/compiler/jit/xla_cpu_device.cc:41] Not creating XLA devices, tf_xla_enable_xla_
devices not set
2023-04-10 19:38:17.132780: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic libra
```

```
ry 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared object file: No such file or directory
2023-04-10 19:38:17.132797: W tensorflow/stream_executor/cuda/cuda_driver.cc:326] failed call to cuInit: UNKNOWN ERROR
(303)
2023-04-10 19:38:17.132820: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel driver does not appear to
be running on this host (alisina): /proc/driver/nvidia/version does not exist
2023-04-10 19:38:17.133166: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized wit
h oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:
SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-04-10 19:38:17.133388: I tensorflow/compiler/jit/xla_gpu_device.cc:99] Not creating XLA devices, tf_xla_enable_xla_
devices not set
2023-04-10 19:38:18.490589: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR optimizatio
n passes are enabled (registered 2)
2023-04-10 19:38:18.524669: I tensorflow/core/platform/profile_utils/cpu_utils.cc:112] CPU Frequency: 1696065000 Hz


Epoch 1/5
   1/3200 [..............................] - ETA: 2:20:40 - loss: 3.3965 - accuracy: 0.1000

2023-04-10 19:38:21.150929: W tensorflow/core/framework/cpu_allocator_impl.cc:80] Allocation of 21196800 exceeds 10% of
free system memory.
2023-04-10 19:38:21.214284: W tensorflow/core/framework/cpu_allocator_impl.cc:80] Allocation of 25660800 exceeds 10% of
free system memory.

   2/3200 [..............................] - ETA: 16:07 - loss: 3.4719 - accuracy: 0.0750

2023-04-10 19:38:21.469000: W tensorflow/core/framework/cpu_allocator_impl.cc:80] Allocation of 21196800 exceeds 10% of
free system memory.
2023-04-10 19:38:21.514681: W tensorflow/core/framework/cpu_allocator_impl.cc:80] Allocation of 25660800 exceeds 10% of
free system memory.

   3/3200 [..............................] - ETA: 17:19 - loss: 3.5083 - accuracy: 0.0722

2023-04-10 19:38:21.785967: W tensorflow/core/framework/cpu_allocator_impl.cc:80] Allocation of 21196800 exceeds 10% of
free system memory.

3200/3200 [==============================] - 1131s 353ms/step - loss: 0.8526 - accuracy: 0.7606 - val_loss: 0.0041 - va
l_accuracy: 0.9992
Epoch 2/5
3200/3200 [==============================] - 861s 269ms/step - loss: 0.0162 - accuracy: 0.9954 - val_loss: 1.0348e-04 -
val_accuracy: 1.0000
Epoch 3/5
3200/3200 [==============================] - 853s 267ms/step - loss: 0.0094 - accuracy: 0.9973 - val_loss: 1.6208e-04 -
val_accuracy: 1.0000
Epoch 4/5
3200/3200 [==============================] - 829s 259ms/step - loss: 0.0064 - accuracy: 0.9985 - val_loss: 9.1734e-04 -
val_accuracy: 0.9997
Epoch 5/5
```

```
3200/3200 [==============================] - 5766s 2s/step - loss: 0.0074 - accuracy: 0.9983 - val_loss: 7.3975e-06 - v
al_accuracy: 1.0000
```

Out[3]: `<tensorflow.python.keras.callbacks.History at 0x7f9f04abaf10>`

3- Save the model

In [4]:
```
model_json = classifier.to_json()
with open("model_03.json", "w") as json_file:
    json_file.write(model_json)
classifier.save_weights('model_03.h5')
```

Predicting

Import Required Modules

In [6]:
```
from keras.models import model_from_json
import numpy as np
import cv2 as cv
import operator, random
import glob, sys, os
```

1- Loading the model

In [7]:
```
json_file = open("model_03.json", "r")
model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(model_json)

# load weights into new model
loaded_model.load_weights("model_03.h5")
print("Loaded model from disk")
```

```
Loaded model from disk
```

2- Test the prediction of model

In [8]:

```
font = cv.FONT_HERSHEY_COMPLEX_SMALL

capture = cv.VideoCapture(0)

while True:
    _, frame = capture.read()
    height,width = frame.shape[:2] # frame.shape[0] -> height, frame.shape[1] -> width

    # Simulating mirror image
    frame = cv.flip(frame, 1)

    # Got this from collect-data.py
    # Coordinates of the ROI(Region of interested)
    x1 = int(0.5*frame.shape[1])
    y1 = 10
    x2 = frame.shape[1]-10
    y2 = int(0.5*frame.shape[1])

    # Drawing the ROI for bounding box
    cv.rectangle(frame, (x1-1, y1-1), (x2+1, y2+1), (255,0,0) ,1)

    # Extracting the ROI
    roi = frame[y1:y2, x1:x2]

    # Resizing the ROI so it can be fed to the model for prediction
    roi = cv.resize(roi, (96, 96))
    roi = cv.cvtColor(roi, cv.COLOR_BGR2GRAY)
    blur_img = cv.medianBlur(roi, 5)
    test_image = cv.adaptiveThreshold(blur_img, 255, cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY, 11, 2)
    cv.imshow("Result", test_image)

    # Batch of 1
    result = loaded_model.predict(test_image.reshape(1, 96, 96, 1))
    print(result)

    prediction = {
        'Alef' : result[0][0],
        'Alef A' : result[0][1],
        'Ayen' : result[0][2],
        'Beeh' : result[0][3],
        'Daal' : result[0][4],
        'Feeh' : result[0][5],
        'Gaph' : result[0][6],
        'Ghain' : result[0][7],
        'Heeh' : result[0][8],
        'Heeh 2' : result[0][9],
```

```
    'Hamza' : result[0][10],
    'Jeem' : result[0][11],
    'Kaph' : result[0][12],
    'Kheh' : result[0][13],
    'Laam' : result[0][14],
    'Meem' : result[0][15],
    'Nonn' : result[0][16],
    'Peeh' : result[0][17],
    'Qaph' : result[0][18],
    'Reeh' : result[0][19],
    'Seeh' : result[0][20],
    'Saad' : result[0][21],
    'Shen' : result[0][22],
    'Seen' : result[0][23],
    'Teeh' : result[0][24],
    'Toyy' : result[0][25],
    'Woww' : result[0][26],
    'Yaah' : result[0][27],
    'Zaal' : result[0][28],
    'Zaad' : result[0][29],
    'Zeeh' : result[0][30],
    'Zoyy' : result[0][31],
}

sa =max(result[0]*100)
print("\n",sa,result[0])


# Sorting based on top prediction
prediction = sorted(prediction.items(), key=operator.itemgetter(1), reverse=True)

# Displaying the predictions
cv.rectangle(frame, (x1, y1+260) , (x2, y2) , (0,0,0),-6)

if sa>99:
    # Displaying the predictions
    cv.putText(frame,"  " + prediction[0][0],(x1, y1+290), font, 1,(255,255,255),1,cv.LINE_AA)
    pred = prediction[0][0]

cv.putText(frame, prediction[0][0], (10, 120), cv.FONT_HERSHEY_PLAIN, 1, (145,195,215), 1)
cv.imshow("Result", frame)

interrupt = cv.waitKey(1)
if interrupt & 0xFF == ord('q'): # Esc key
    break
```

```
capture.release()
cv.destroyAllWindows()
```

```
 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.
  0. 0. 0. 0. 0. 0. 0. 0.]]

 100.0 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 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.
  0. 0. 0. 0. 0. 0. 0. 0.]]

 100.0 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 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.
  0. 0. 0. 0. 0. 0. 0. 0.]]

 100.0 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 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.
  0. 0. 0. 0. 0. 0. 0. 0.]]

 100.0 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```