HANDWRITING RECOGNITION PROJECT REPORT

Video Link

- https://youtu.be/-SBbm2xIFCM

Problem

In real life, usually understanding of someone's handwriting might be a challenging problem and converting documents on papers to digital environment is necessary but it is not efficient without using a third-party software.

Aim of the Project

Our aim on this project is to reduce unnecessary time consumption and labor force. Also, our main purpose is to prevent undesirable human mistakes that can occur on written communication. And on doing so minimizing the error that might happen on conversion of documents to digital environment.

Solution to the Problem

Our solution to this problem is constructing a model based on artificial intelligence that consist of convolutional neural network and computer vision. The model must be trained with substantial and appropriate data. We decided that EMNIST dataset is more suitable for our model with help of TensorFlow and Keras libraries.

The EMNIST Dataset

The EMNIST dataset is a set of handwritten character digits and converted to a 28x28 pixel image format. The dataset contains 131,600 labelled data which 112,800 of them are divided for train purposes and 18,800 of them for test purposes. Beside of the capital letters the dataset also contains lowercase letters and numbers. Since English alphabet has 26 letters and there are 10 digits, we should have 62 classes, but we have 47 classes in our model. It is because of some letters like 'o', 'O' and 's', 'S' is same in the digital environment. In *Figure-1*, dataset we used, shown with graphically and it also shows amount of data used for each of the letters and digits. Additionally, we can observe some of the lowercase letters accepted as capital letters.

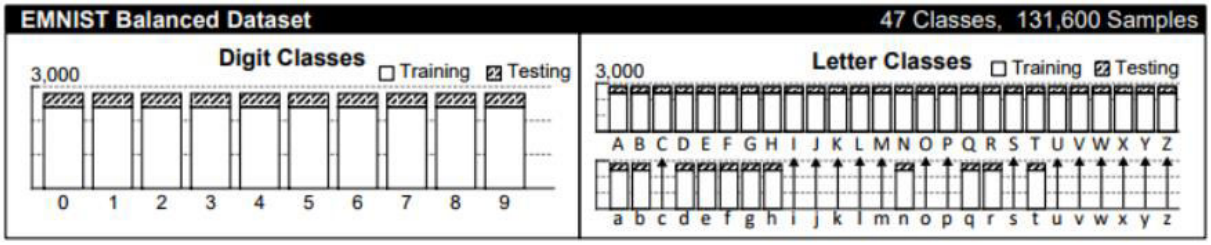


Figure-1.1

In *Figure-1.2*, we can observe some data that selected randomly from the dataset being unprocessed in our model. We can easily see that the data needs adjustment before being used in our model.

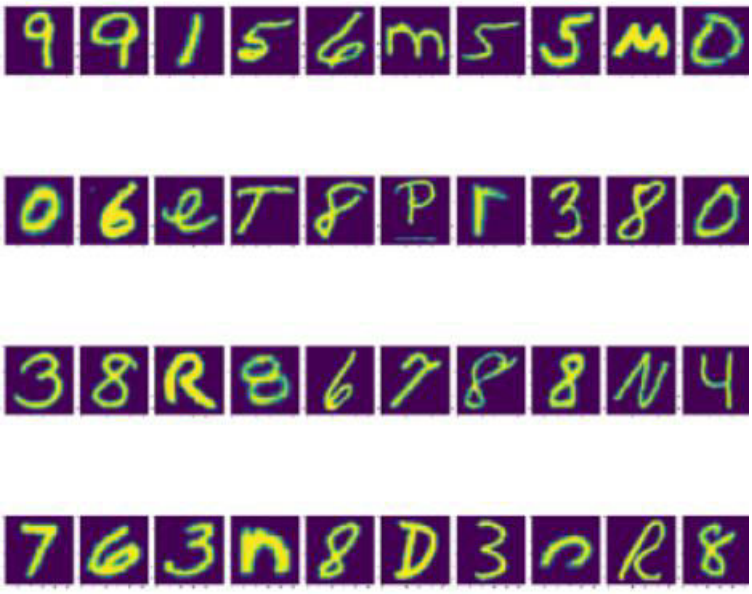


Figure-1.2

The Model

Besides the convolutional neural network, the data adjustments have also very crucial role on the development of the model. Original form of the dataset is formed by one dimensional array, it means one data has 785 unsigned integers that first column stands for the label of the data and others hold the pixel data which are between 0 and 255.

Firstly, we have to take first column as dependent variable than the rest of the columns are considered as independent variables. After that we have to rearrange our data into two-dimensional array that is 28x28 unsigned integers. Since convolutional neural network accepts only numbers between 0 and 1, so we need to normalize our independent variables by dividing with 255 shown as *Figure-2.1*.

```
# Normalise
train_x = train_x.astype('float32')
train_x /= 255
test_x = test_x.astype('float32')
test_x /= 255
```

Figure-2.1

Additionally, we have to flip and rotate data as 90 degree because the data has been set in irregular way as shown in Figure-2.2.

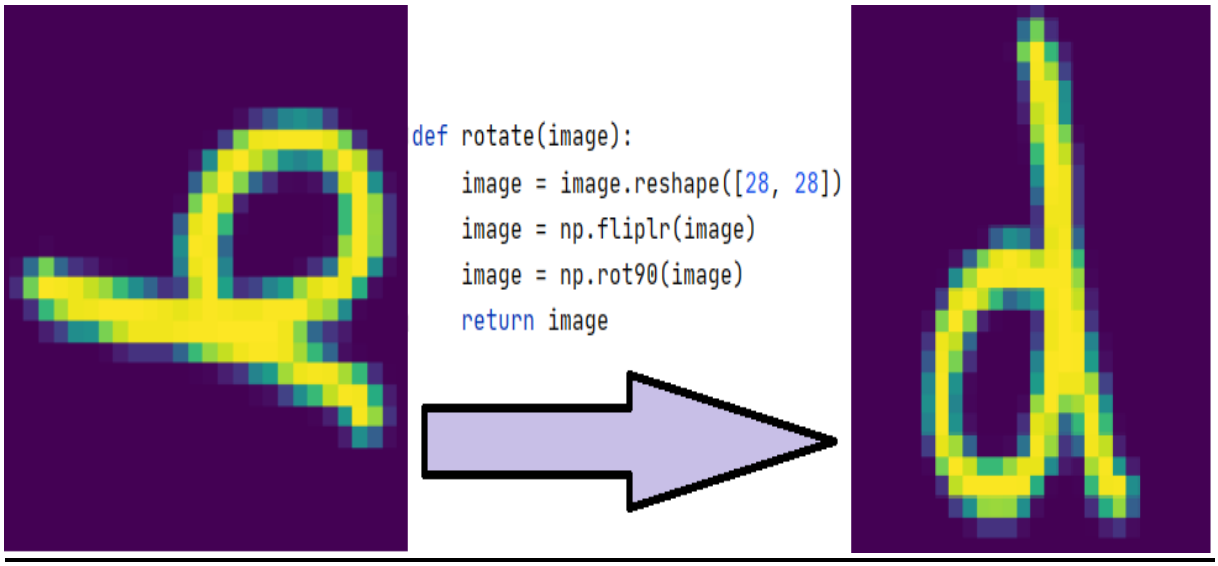


Figure-2.2

After splitting the data for train and test purposes, we reshape the images for convolutional neural network (CNN). Finally, our dataset is ready for CNN.

Building Model

First, we create a Sequential model and after that we start our model as adding convolutional two-dimensional layer for our input. This layer consist of 128 filters and their kernel size is 5x5. And also, we use rectifier (ReLU) activation function since it is our first layer, we have to indicate input shape as 28x28x1. After the convolutional layer to reduce the size, we add max-pooling two-dimensional layer and its pool size is 2x2.

After we done with one convolutional and one pooling layer, we add two more layers as same as first two layers but this time in convolutional layer we use 64 filters and their kernel size is 3x3. We don't have to specify the input shape since TensorFlow automatically handle this for us in running time. For the purpose of converting 64 two-dimensional array that are 7x7, to the 3136 one dimensional array, we use flatten layer. In the end, we transfer the data to the fully connected layer.

After flatten the data, we add dense layer that reduces our data to 128 units and activation function is "relu" same as before layers. To prevent from overfitting, we add dropout layer.

Finally, we add the output layer that consist of 47 classes and the activation function as "softmax". All layer processes shown in *Figure-3.1 (summary of the model)*.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 128)       3328
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 64)        73792
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               401536
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 47)                6063
=================================================================
Total params: 484,719
Trainable params: 484,719
Non-trainable params: 0
_________________________________________________________________
```

Figure-3.1

The character recognition convolutional neural network schema is shown in the *Figure-3.2.*

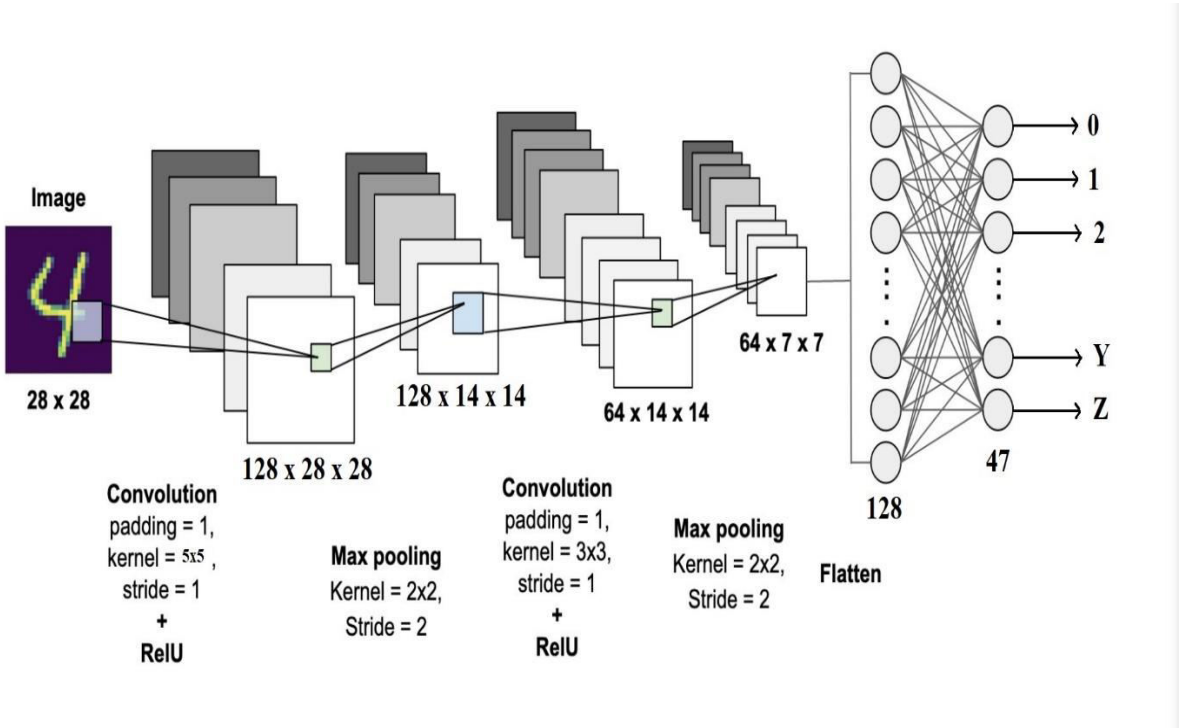


Figure-3.2

In the compilation of the model, we use "categorical_crossentropy" as loss function. For optimizer we use "adam" function and for metrics we use "accuracy" parameter.

Finally, for training our model we use fit function. In this function, we are not taking the data at once but to make it more efficient, we split the data to parts that have 512 data. We are doing all training part in 10 epochs. The function is shown as in the *Figure-3.3*.

```
history = model.fit(train_x, train_y, epochs=10, batch_size=512, verbose=1, validation_data=(val_x, val_y))
```

Figure-3.3

Final values of loss and accuracy are shown in *Figure-3.4*. Loss and accuracy values by epochs graphs are shown in the *Figure-3.5*.

```
Test loss: 0.36574761957584895
Test accuracy: 0.8728655779626155
```

Figure-3.4

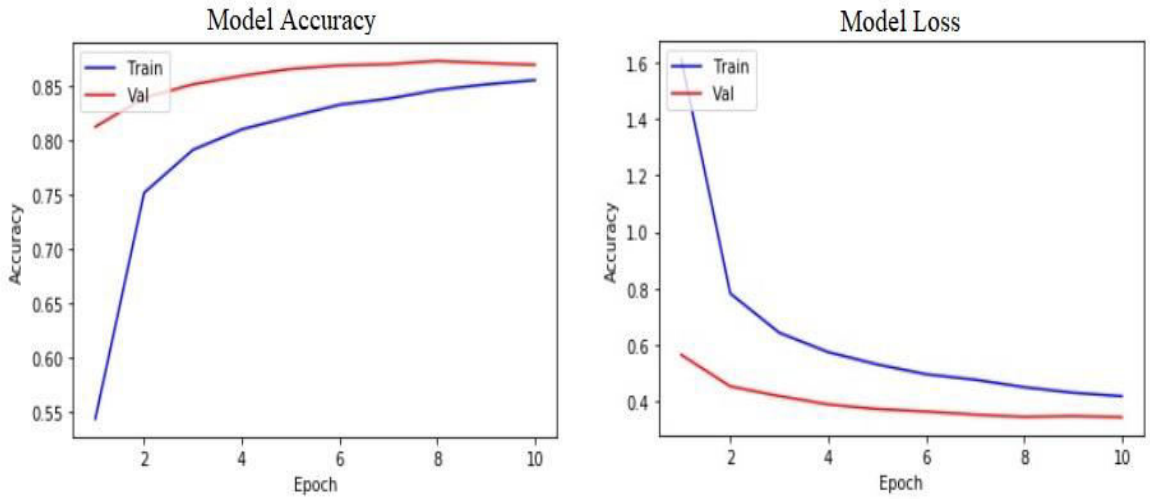


Figure-3.5

User Interface

For simple usage we construct a user interface. The interface has three main purpose. First one is to make user to be able to upload their prepared pictures and observe the output more comfortably. Second one is to process the uploaded pictures within the model and print the resulting picture on the screen. Third one is saving the resulting image as a jpeg format to the determined location. Splash screen of the user interface is shown in the *Figure-4.1*.

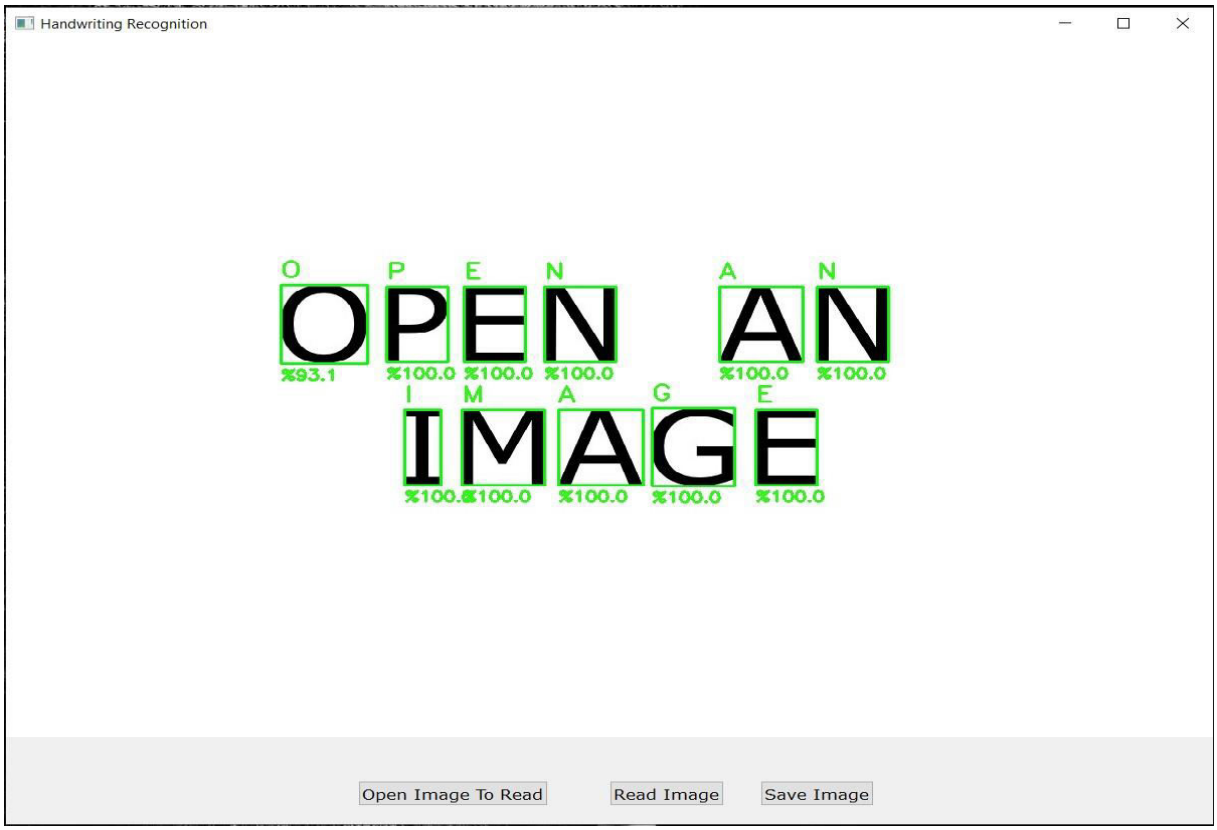


Figure-4.1

Under favor of storing our trained model as h5 file, while initializing the user interface, the model is loading into the memory by the test module. Since the model loaded into the memory, we don't have to train the model for all images. Maximum gap between two letters should be calculated dynamically since everybody has different handwriting styles thus gap between words is different too. To prevent some errors, we initialize default value of max gap variable as 50.

When a user clicks the read button, in the test module "readImage" function will be called by the user interface. The function takes address of image and read the image from that address. It takes a copy of the image and convert it to the black and white. After changing the colors, we define a threshold value as 85. Threshold basically considers the pixel values lower than 85 as white and higher than 85 as black.

To detect the written letters and digits we find the contours and store them in a list. In order to read the contours line by line, we sorted them left to right and top to bottom. Sorting order is shown in *Figure-4.2*.

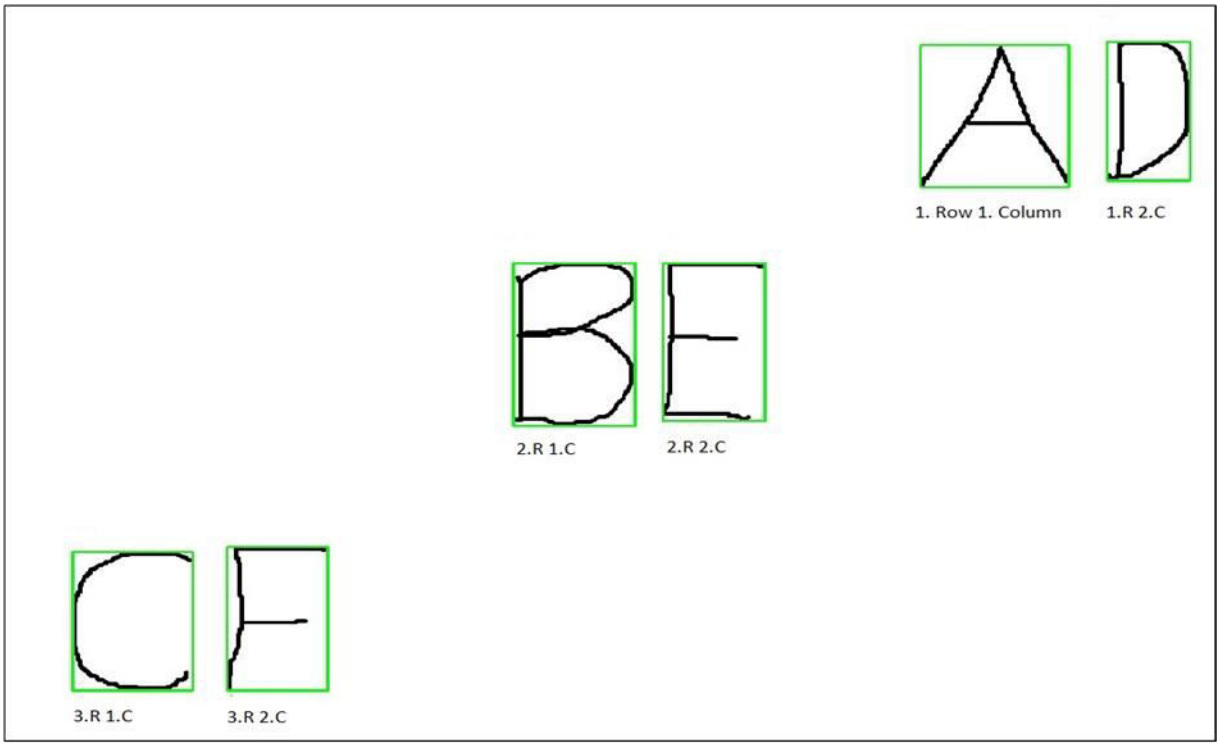


Figure-4.2

After sorting the contours, we traverse the sorted contours one by one and we decide if it is end of a word or end of a line. When a contour is at end of a word, we put a one character of white space after the contour and also if it is at end of a line, we continue with the new line. But the real issue is that to decide whether a contour is the last character of a word or not. To handle this problem, we dynamically calculate gap between contours and find the average gap between letters. The average gap and sorted contour coordinates are shown in *Figure-4.3*.

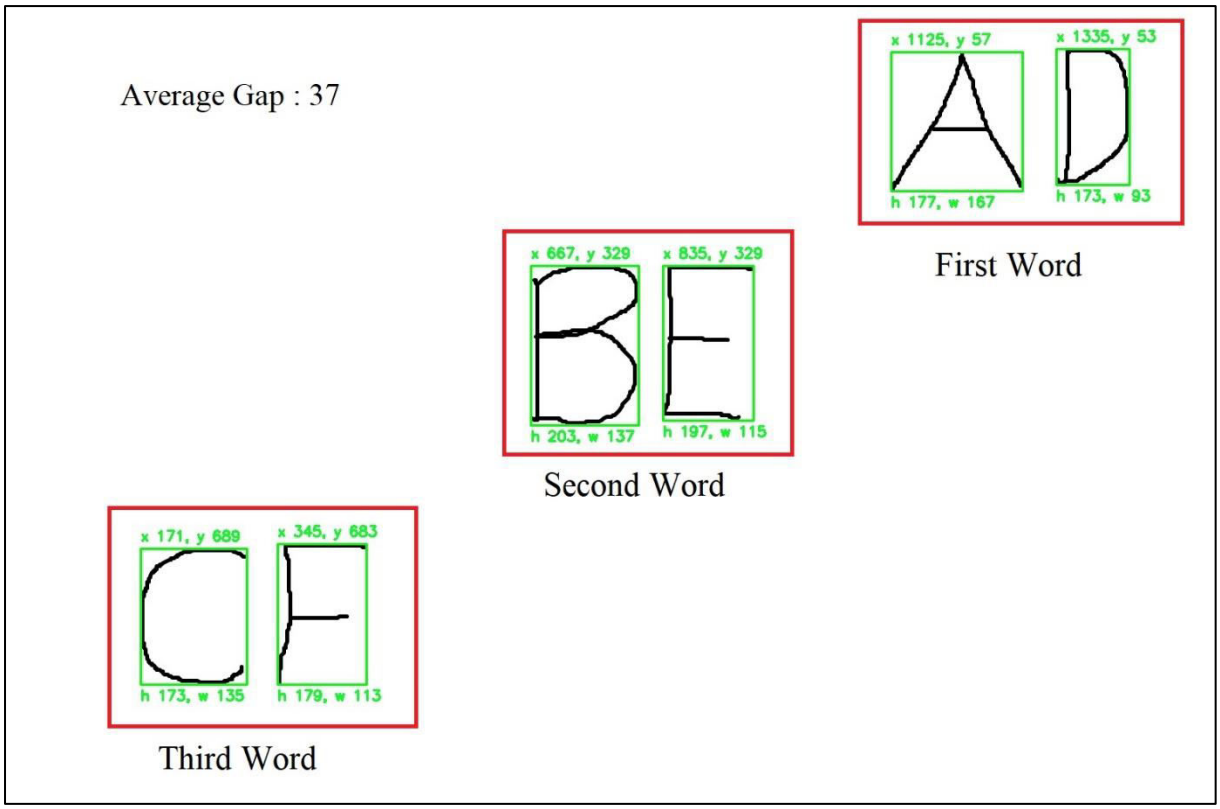


Figure-4.3

After calculating a contour's coordinates and width & height values, we create a copy of contour as 20x20 sub image and then we pad the sub image with 4x4 black pixels. After normalizing the sub image, we have a 28x28 sub image ready to send to the model.

The model takes the sub image as 28x28 two-dimensional array and process it in all layers and produce probability values for all classes. When the model done processing the sub image, we take the highest probability value from all classes. Final image shown in *Figure-4.4*.

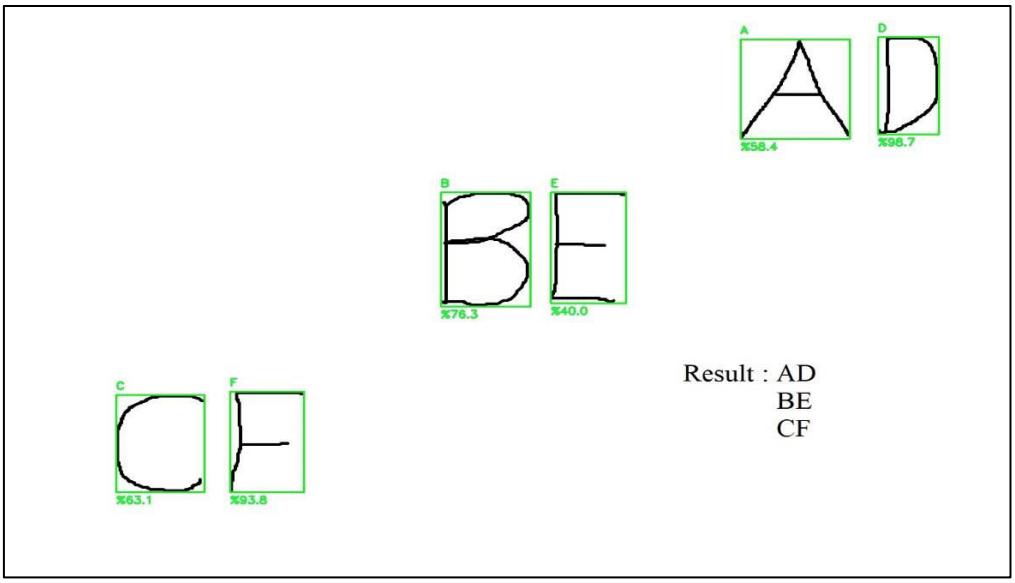


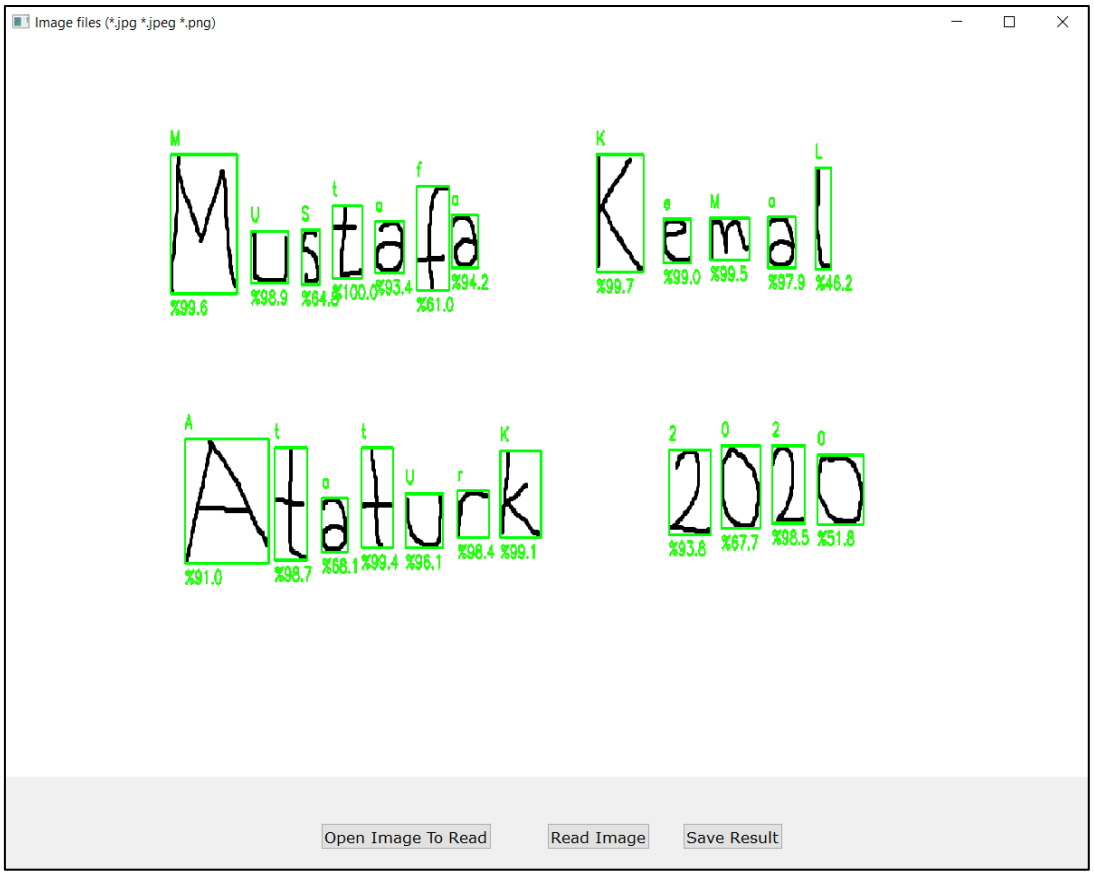
Figure-4.4

In summary, we have created our model with help of CNN using TensorFlow and Keras libraries. And in image processing we detected letters and digits one by one with help of OpenCV. After image processing the resulting image shown to the user. Also, we gave user the option to save the result as a folder that contains text document and final image.

One Example from The Project

- Open Image To Read:

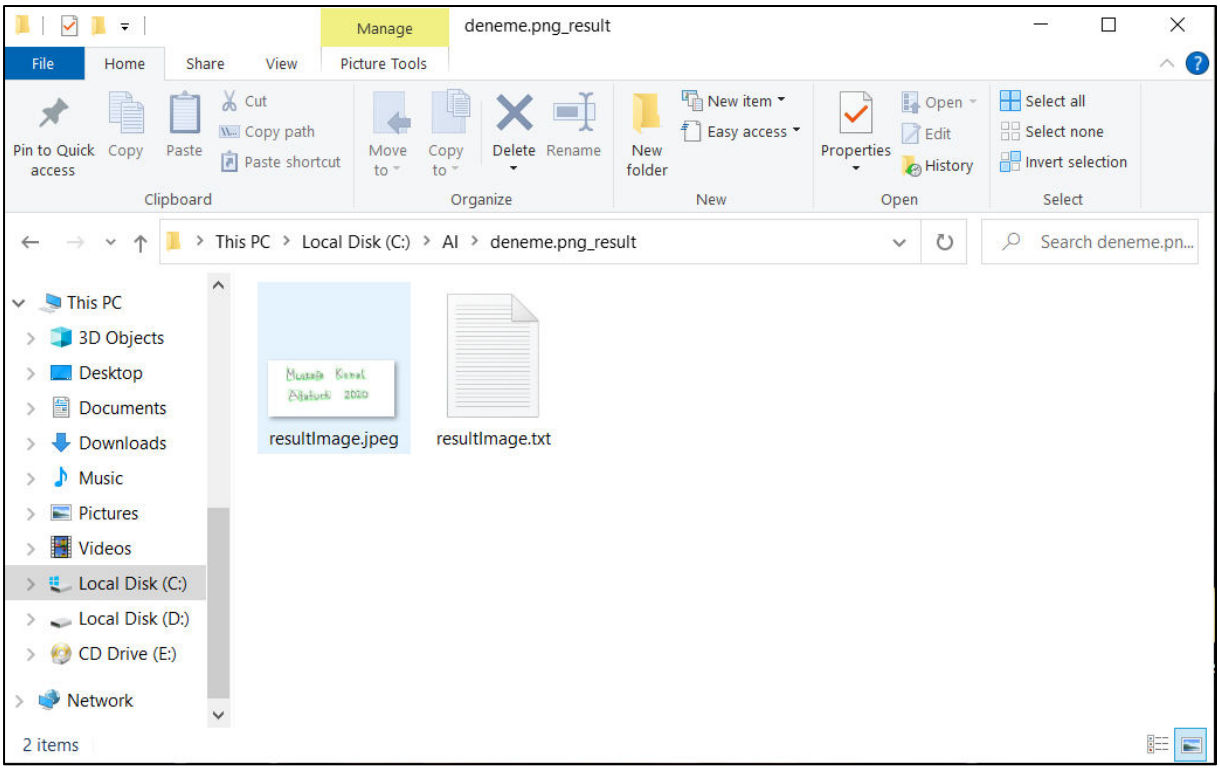
- Read Image:



- Terminal Result:

```
Max Gap : 36 : Counter 22
Mustafa Kemal
Ataturk 2020
```

- Save Result:



References & Sources

- https://www.tensorflow.org/api_docs/python/tf/all_symbols
- https://keras.io/api/
- https://docs.opencv.org/master/
- https://www.bilkav.com/makine-ogrenmesi-egitimi/
- https://www.kaggle.com/crawford/emnist
- https://en.wikipedia.org/wiki/Activation_function