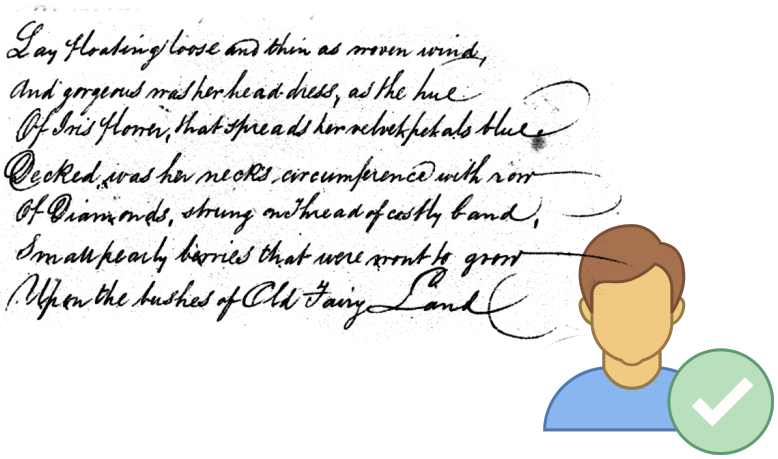


Handwriting Verification

In Biometric systems

Hassan Teymoori, 1947458

Mohammadreza Shabani, 1966731

June 2022

Abstract

Identification of individuals using their handwriting is one of the stimulating problems in the biometric systems. It can be used in various fields like security and criminological analysis, antique documents and literature analysis, forgery and signature analysis, etc. Handwriting has a crucial character in the demonstration of the identity and cultured traits of an individual. It is the critical distinctiveness of an individual. Methodologies that are based on deep learning are verified as flexible techniques for extracting features from huge volumes of data and give solid and accurate predictions of the underlying patterns than traditional methodologies like Local Binary Pattern(LBP) and etc. An automatic handwriting recognition system aids in identifying whether the stated handwriting is accurately matched and allocated to the original writer of that handwriting.

There are two modes of data capturing in any kind of writer identification, they are online and offline. Spatial coordinate values are considered mainly in the former case whereas the temporal details are considered in the latter. Concerning the textual content, offline writer identification can be categorized as text-dependent and text-independent. Text-dependent procedures highly emphasize context and semantics in the handwritten image, so it needs an image having fixed text content and evaluates the resemblance of the input image with already listed prototypes for this purpose. In contrast, text-independent handwriting recognition emphasizes the images that do not hang on the text content which is fixed.

Since the convolutional neural networks and their state of art architectures are very powerful in extracting deep features and their huge success in the fields of computer vision made us implement the proposed model of the writer identification system. It is done in three stages. The first stage is Image pre-processing where various filters are applied to images to remove noise. The second stage is feature extraction using AlexNet-based CNN architecture. The patterns are learned and deep features are extracted to accurately classify different handwriting styles. The final stage is writer classification using all-against-all methods to evaluate the model.

1. Introduction

Automatic handwriting recognition and writer verification system help in determining the valid writer of a particular handwritten text image among the number of writers whose handwriting has been already trained. Many research scholars had shown interest in this domain because of its large number of applications like security and criminological analysis, antique documents , literature analysis, and etc. The traditional approaches extract features from handwriting using hand-designed and manual calculations or from image processing techniques. They used techniques such as scale-invariant feature transforms, local binary patterns (LBP), wavelet transforms, etc, where the final feature vectors are extracted from local patches of handwriting images and constructed models like a bag of words, etc. With the advent of deep learning algorithms like CNNs, ANNs, etc, the feature extraction work has become much easier. Since the convolutional neural networks and their state of art architectures like AlexNet, VGG, GoogleNet are very powerful in extracting deep features and their huge success in the fields of computer vision made several research scholars employ CNN techniques for writer identification.

1.1. AlexNet Architecture:

AlexNet was the first convolutional network that used GPU to boost performance.

1. In general, AlexNet architecture consists of 5 convolutional layers, 3 max-pooling layers, 2 normalization layers, 2 fully connected layers, and 1 softmax layer.

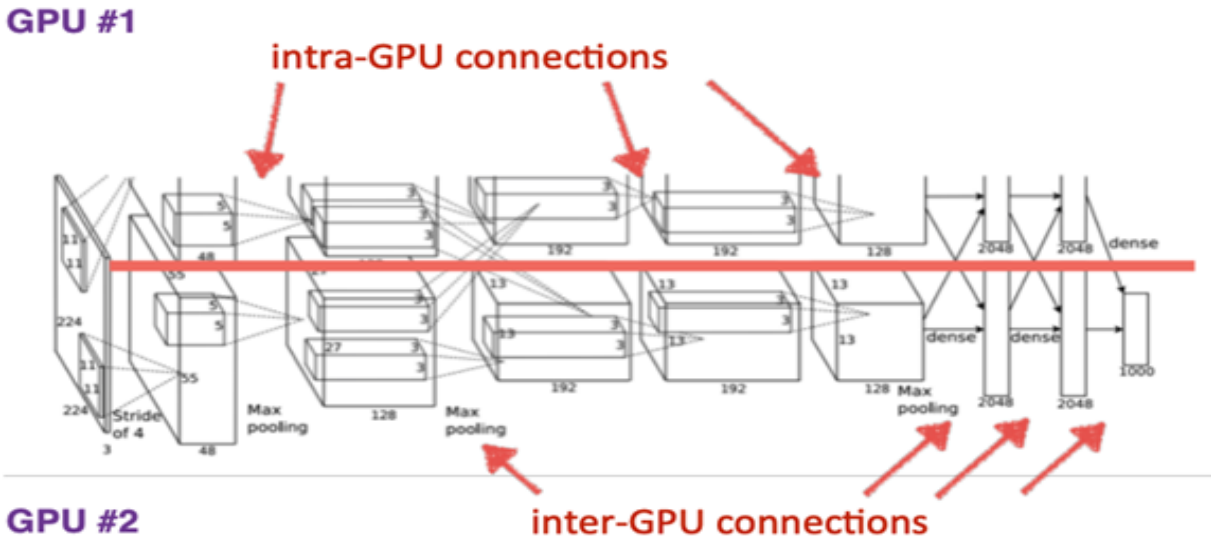
2. Each convolutional layer consists of convolutional filters and a nonlinear activation function ReLU.

3. The pooling layers are used to perform max pooling.

4. Input size is fixed due to the presence of fully connected layers.

5. The input size is an argument that must be provided for the AlexNet architecture. Even though in many projects we have found that they use (224, 224) image shape, we decided to increase the size of the input image as the resolution of the images was high. With the 224x224 image shape, we realized that we have less amount of text inside the split image, which was not informative enough.

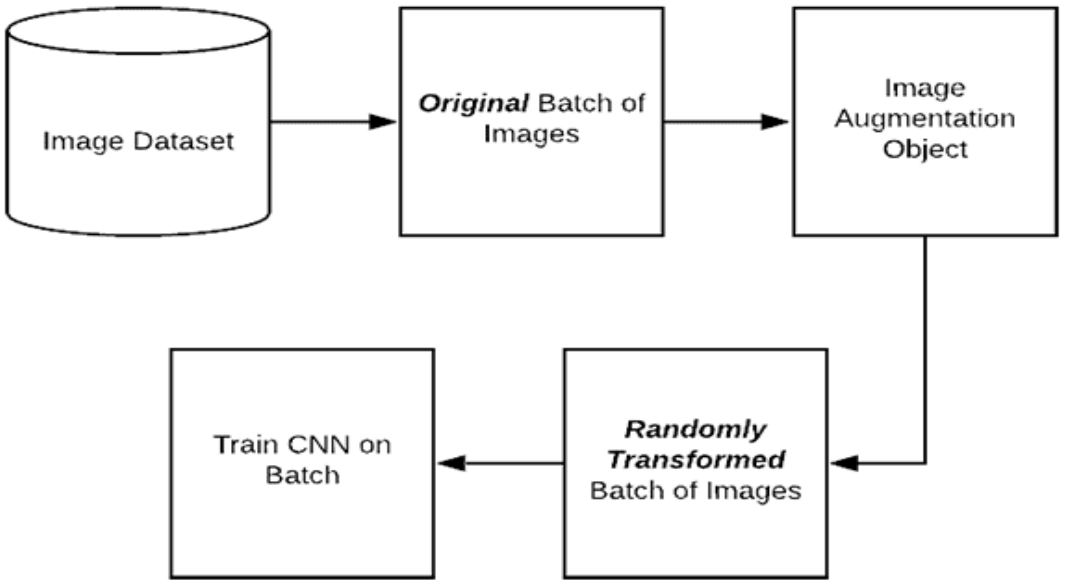
6. AlexNet overall has 60 million parameters.



AlexNet Architecture

1.2. Data Augmentation:

When you show a Neural Network, different variations of the same image, it helps prevent overfitting. It also forces the Neural Network to memorize the key features and helps in generating additional data. Overall, our goal when applying data augmentation is to increase the generalizability of the model.



In-place/on-the-fly data augmentation

2. Data Providing and Pre-processing :

Providing appropriate training datasets and testing datasets is the main objective for any machine learning task. There are several important dataset on the internet for specific kind of tasks.

2.1. Raw DataSet:

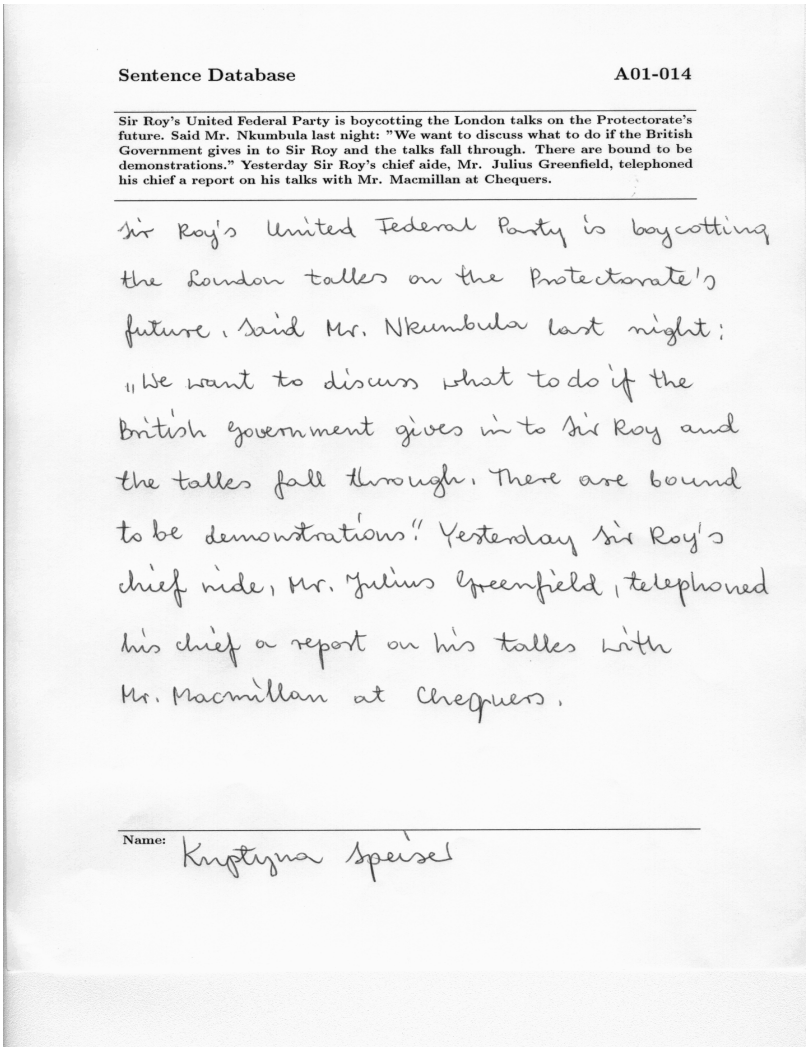
In this project, we used the IAM Handwriting Database contains forms of handwritten English text. Link

The be able to download the dataset into our local machine, we implemented a `dataprovider` module which is able to download the dataset using the `Downloader` class.

dataprovider.py modules

```
12  class Downloader:
13      """ A class used to retrieve the raw data from google drive """
14
15      def __init__(self, path=config.path.get('raw_data')):
16          """
17          initialize the class
18          :param path: the location in which the downloded file will be stored
19          :param file_name: the name of downloded file, e.g: test.zip
20          """
21          if not os.path.exists(path):
22              os.mkdir(path)
23          self.path = path
24          self.file_name = None
25
26      def download(self, google_id, file_name):...
37
38      def extract(self, auto_delete=True):...
54
55      def __cleaning(self):...
```

The database contains forms of unconstrained handwritten text, which were scanned at a resolution of 300dpi and saved as PNG images with 256 gray levels. The figure below provides a sample of a complete form with the specific writer's ID and Name. The IAM Handwriting Database has 657 writers who contributed samples of their handwriting.



Sentence Database A01-014

Sir Roy's United Federal Party is boycotting the London talks on the Protectorate's future. Said Mr. Nkumbula last night: "We want to discuss what to do if the British Government gives in to Sir Roy and the talks fall through. There are bound to be demonstrations." Yesterday Sir Roy's chief aide, Mr. Julius Greenfield, telephoned his chief a report on his talks with Mr. Macmillan at Chequers.

Sir Roy's United Federal Party is boycotting the London talks on the Protectorate's future. Said Mr. Nkumbula last night: "We want to discuss what to do if the British government gives in to Sir Roy and the talks fall through. There are bound to be demonstrations." Yesterday Sir Roy's chief aide, Mr. Julius Greenfield, telephoned his chief a report on his talks with Mr. Macmillan at Chequers.

Name: Krystyna Speisel

Raw DataSet Sample

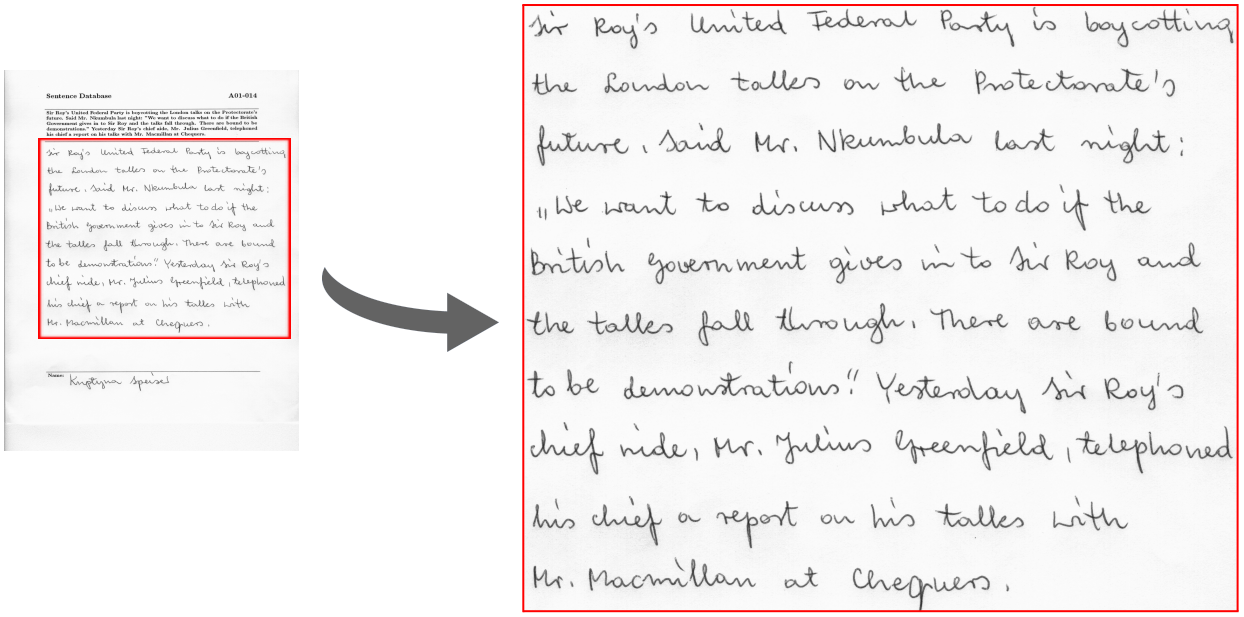
As you can see, the written english text as well as the writer ID appears on top of the form which will be removed during the pre-processing step in order to access the pure handwriting part.

2.2. Paragraphs Extracting:

All forms are provided as PNG files and the corresponding form label files, including segmentation information and a variety of estimated parameters, are included in the image files as meta-information in XML format which is described in XML file and XML file format.

The XML files as well as the other annotation information are downloaded using the same `Downloader` class from `dataprovider` modules.

As you can observe in the above figure and mentioned in the previous section, the raw image consists of some information that we don't need, so by using the corresponding XML files, we extracted only handwritten paragraphs with respect to the given coordinates.



Extracted paragraph image sample from the Raw DataSet

2.2. Image Preprocessing:

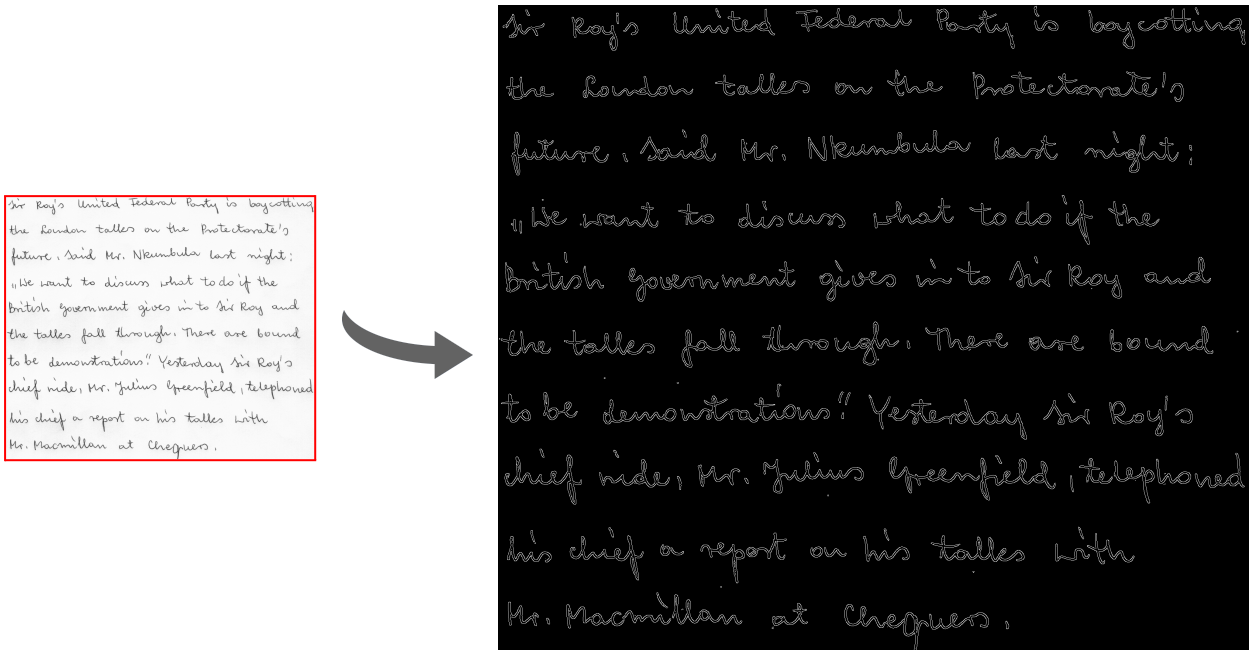
In this stage, the main objective is to remove the noise and variations in background lighting, since the images are differently captured by every writer. This is done using a **Gaussian blur filter** and finally extracting the edges alone from the handwriting image. This is done using **Canny Edge Detector**.

Gaussian Blur filtering is an operation in which a Gaussian filter instead of a box filter is convolved over an image. It is a low-pass filter that eliminates the high-frequency components and reduces the unwanted noise in the image.

The Canny edge detector is an edge detection operation that uses a multi-stage algorithm to detect an extensive range of edges in images.

All the preprocessing processes ,which have been mentioned above, implemented by using the open-cv library. We created an interface over the open-cv related functions and bring the functionality into our own modules so called `Preprocessing.`

```python
class Preprocessing:
    """
    A class to apply some image processing processes to the handwritings
    """
    def __init__(self, path=config.path.get('paragraphs_edged')):…

    def apply_to_directory(self, paragraph_dir):
        for filename in tqdm(os.listdir(paragraph_dir)):
            if not filename.startswith('.'):
                self.load_image(
                    os.path.join(paragraph_dir, filename)
                ).RGB_to_GrayScale().blurring().thresholding().save(filename)
        return

    def blurring(self, kernel=(5,5) , sigma=0):…

    def RGB_to_GrayScale(self):…

    def load_image(self, img_path):…
        You, last month • an interface over some of open-cv image processin…
    def thresholding(self, MinThreshold=30, MaxThreshold=50):…
```



Edged Image Sample

Now, we have a perfect sample that is ready to be trained. But we need to differentiate our testing samples as well as the training samples. For this regard we need to select a number of top writers in which they have at least a good sufficient of samples on their disposal in order to give us the ability to perform a full evaluation and training tasks.

2.3. Annotation and Writer Selection:

In this stage, We filtered and selected just writers who have at least **9** samples from the data frame using the information from the annotation XML file. The idea behind this selection is to have a suitable number of train and test samples. As the result, we have **340** forms with **31** different writers. Each writer has a specific ID, so it would be easy to handle the categorization of the forms.

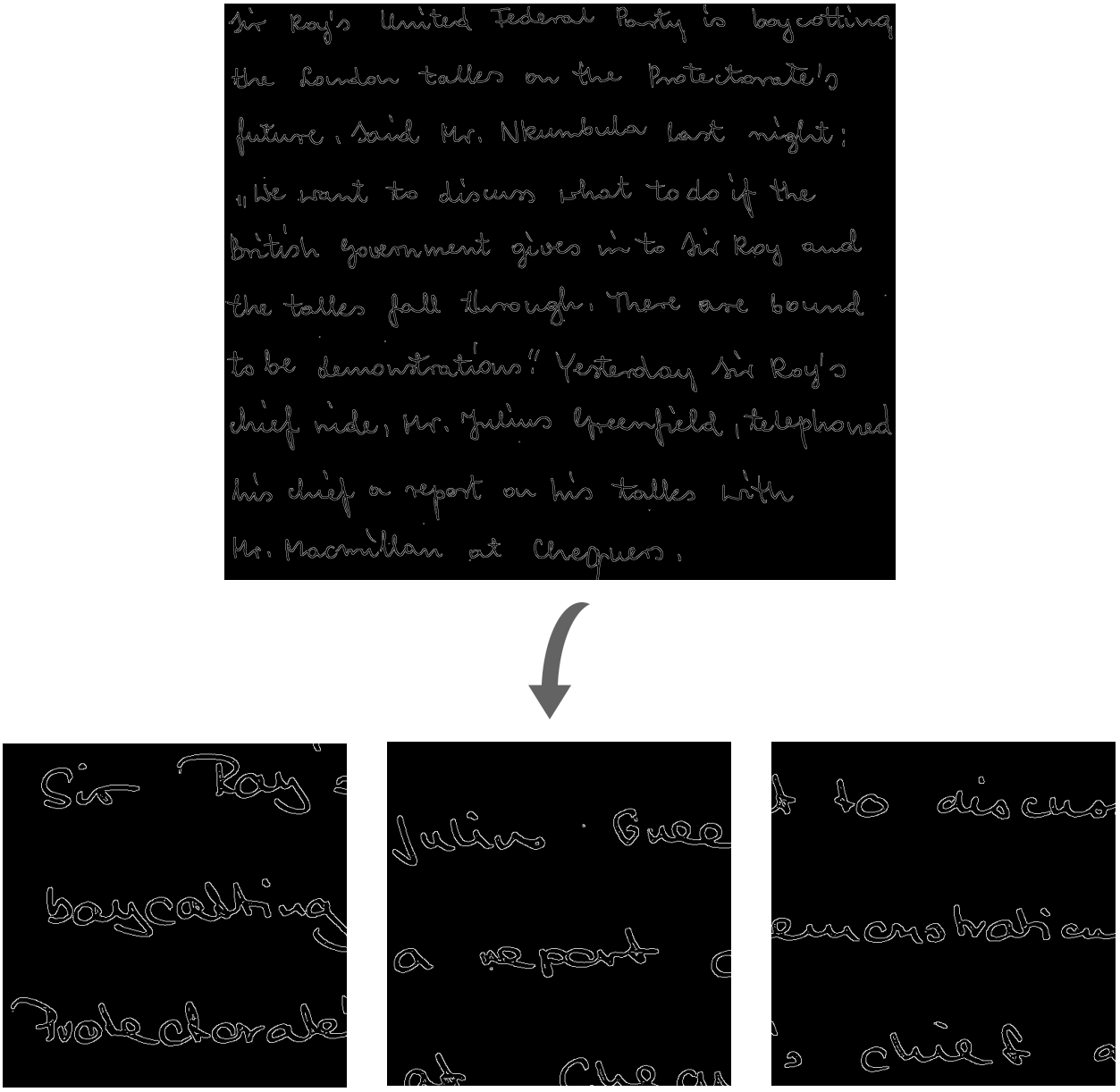
2.4. Split Data:

One of the most important parts of data processing would be this stage. Since it is a biometric task (not a Machine Learning one), we need to define some important specific criteria:

- Each writer must have at least one sample in the test set to be testified.(unlike random splitting in machine learning)
- There should be a sufficient number of training samples due to the nature of the problem.
- There should be a good number of testing samples as well.
- Since the task does not depend on the image itself or text but is to identify the writer, thus we could crop the samples into multiple smaller ones in order to increase the number of the training and testing samples.
- For the training, the overlap between the smaller samples (after cropping) is allowed while for the testing samples overlap is not allowed at all.

We considered **4** forms out of at least **9** per each writer, for the testing set. On the other hand, the remaining which would be at least **5** samples have been considered as the training set. Later on, once we want to prevent having an unbalanced train set, we take only the minimum number of the samples for training which is **5** samples.
For increasing the Dataset samples, we enhanced cropping edged images into several images with sizes of **500 x 500.**

As mentioned above, we considered no overlap between the cropped images but for the train set we considered a little overlap with a specific threshold to have a larger training set. For each sample of the training set, we cropped the edged image into **8** different smaller images. Similarly, for each of the test samples, we cropped the edged image into just **3** different smaller images (to avoid having any overlap).



Samples of Cropped Images

Overall, we finally have **1240** images belonging to **31** different writers for the training set, and **372** images for the testing set.
These numbers are without considering any augmentation process.

The provided datasets, which is the final version of the dataset to be trained or tested for this project, has been uploaded to Google Drive. Uploading in google drive is efficient for accessing data while working in google colab.
Before starting the training phase, let us summarize everything:

- ➢ We have **31** identities, with at least **9** unique forms per each identity.
- ➢ There are **5** unique forms per each identity in the training set
- ➢ There are **4** unique forms per each identity in the test set
- ➢ Each of the training samples cropped into **8** different sub-images
- ➢ Each of the testing samples cropped into **3** different sub-images
- ➢ Overall we have **31 * 5 * 8 = 1240** training samples
- ➢ Overall we have **31 * 4 * 3 = 372** test samples

3. Training Phase:

Now we have everything to start the training phase. Let's define some constants for our deep architecture. Each of the constants has its own application which is summarized in the table below:

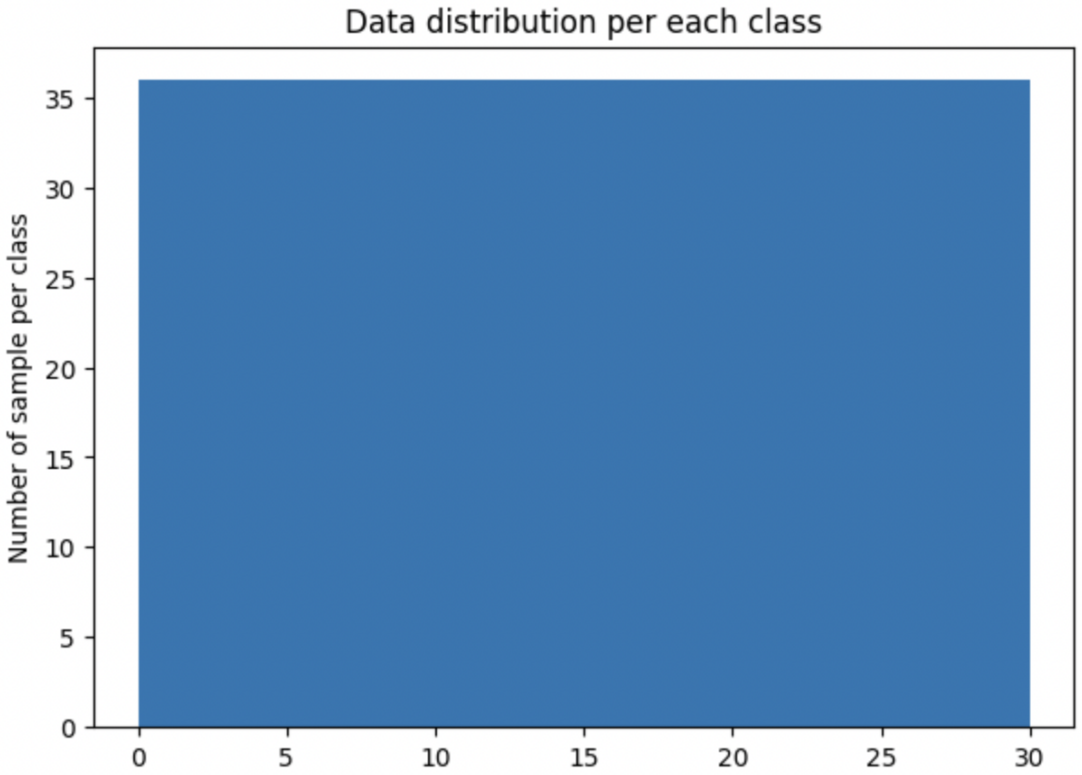
CONSTANT_NAME	APPLICATION
BATCH_SIZE	# samples that will be passed through to the network at one time
IMAGE_SIZE	width of the image
IMAGE_SHAPE	shape of the image (image_height, image_width)
N_CLASSES	the output of the model, the classification layer of the model

BATCH_SIZE = 32
IMAGE_SIZE = 500
IMAGE_SHAPE = (IMAGE_SIZE, IMAGE_SIZE)
N_CLASSES = 31

In the real world, it is not uncommon to come across unbalanced data sets where you might have class A with 90 observations and class B with 10 observations.

One of the rules in machine learning is that it is important to balance the data set. The main reason for this is to give equal priority to each class. To visualize the class distribution we can use the histogram distribution.
Based on the following plot, we can see that our data is perfectly balanced.

3.1. Image Generation:

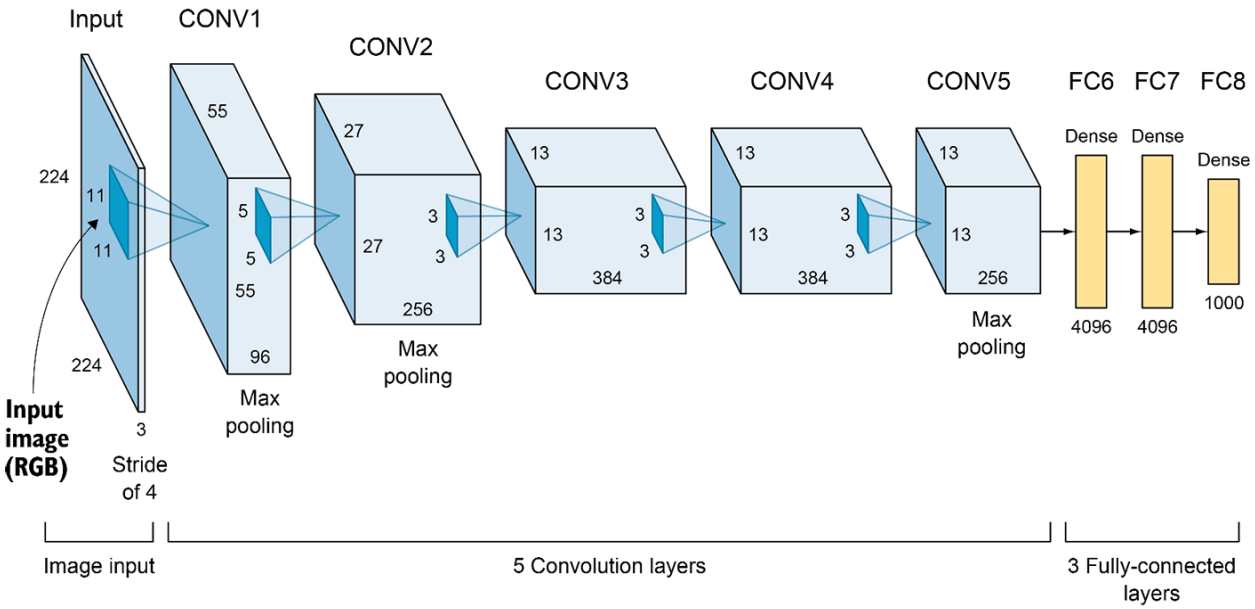


We have trained 3 different models according to three different approaches using **Keras Image Generator**:

1. A data generation without any augmentation
2. Generation with only rotation
3. Generation with more augmentation such as Rotation, shift, brightness, zooming, rescale

3.2. Implementation Part:

Constructing a Convolutional Neural Network Model based on AlexNet architecture is essential for training the handwriting images for classification. To be able to train a neural network we used the TensorFlow and Keras library. The network architecture is shown in the following figure. The network is constructed by referencing AlexNet State of Art CNN architecture. It has 4 convolution layers followed by 4 pooling layers, each after every convolution layer. We decided to remove the last convolution layers in order to decrease the complexity; It was a personal choice; The feature vector is obtained from a flattened layer and sent to two fully connected layers. Other configurations of the network are described below.



AlexNet CNN architecture

Activation Functions used:

- In convolution layer: Relu
- In Output layer: Softmax

Later on, we will remove the classification layer in order to access the feature extraction layer. This could give us the ability to perform a biometric evaluation task. The reason why we train and then remove the classification layer is that we wanted to obtain the best possible weight for our model. Once we have a good model with corresponding weights in our disposal, we would remove the classification layer to access the features. We are sure that our model extracts discriminative features as we have trained it with training a number of training samples and reached a good performance by training the model.

Optimizer used: Stochastic Gradient Descent (SGD)
Loss Function used: categorical_crossentropy
Number of Epochs: 150 (But we used " early stopping ")
Evaluation Metric used: accuracy

To reduce overfitting, Batch Normalization and Dropout of 0.4 are used in CNN and ANN layers respectively.
We also used some callback functions in Keras API such as:

1. **ModelCheckpoint**: callback is used in conjunction with training using the *model.fit()* to save a model or weights (in a checkpoint file) at some interval, so the model or weights can be loaded later to continue the training from the state saved.
2. **EarlyStopping**: Assuming the goal of training is to minimize the loss. With this, the metric to be monitored would be a loss, and the mode would be min. A model.fit() training loop will check at the end of every epoch whether the loss is no longer decreasing, considering the min_delta and patience if applicable. Once it's found to no longer decrease, the training terminates.
3. **ReduceLROnPlateau**: Models often benefit from reducing the learning rate by a factor of 2-10 learning decreases. This callback monitors a quantity and if no improvement is seen for a **patience** number of epochs, the learning rate is reduced.

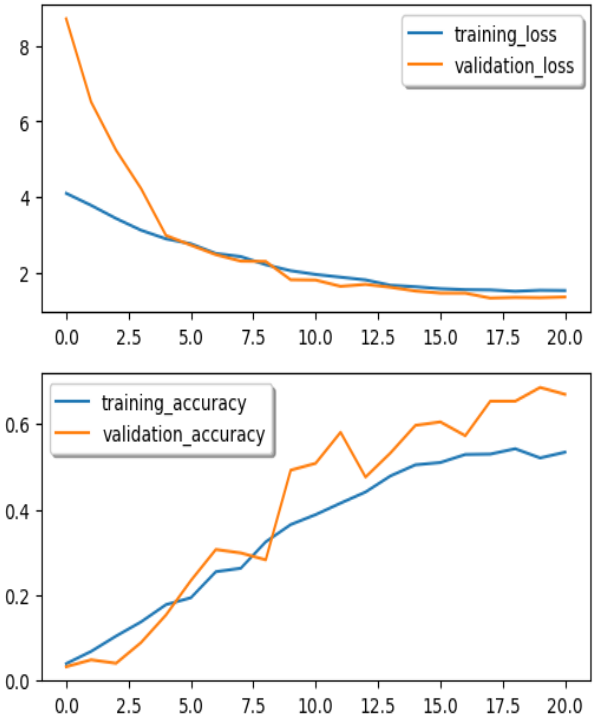
```
# reduces learning rate if no improvement are seen
learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss',
                                            patience=1,
                                            verbose=0,
                                            factor=0.2,
                                            min_lr=0.000001)

# stop training if no improvements are seen
early_stop = EarlyStopping(monitor="val_loss",
                           mode="min",
                           patience=3,
                           restore_best_weights=True)

# saves model weights to file
checkpoint = ModelCheckpoint(os.path.join('/content/gdrive/MyDrive/biometric_project/model', 'cp-{epoch:04d}.h5'),
                           monitor='val_loss',
                           verbose=0,
                           save_best_only=True,
                           mode='min',
                           save_weights_only=True)
```

Now, everything is ready to train the model.
We have trained 3 different models according to three different approaches.
In the following the result of the loss and accuracy of them are visualized.

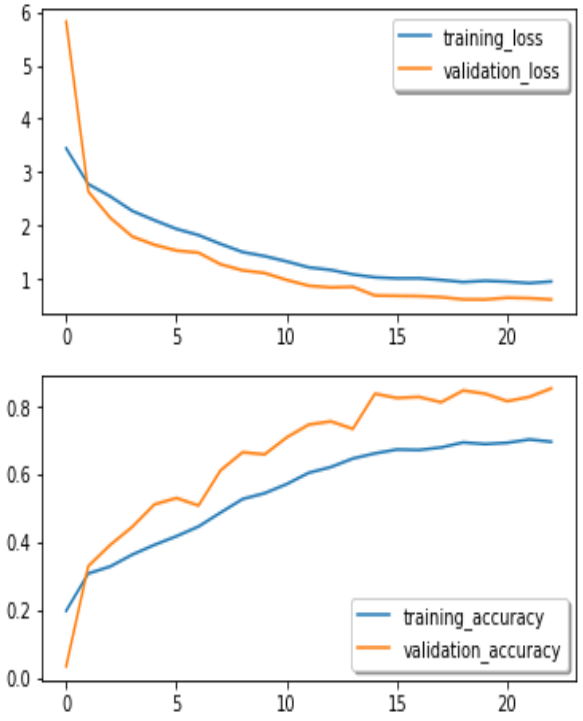
3.3. Loss and Accuracy Visualization:

In the following, you can see the figures corresponding to the training loss and validation loss for two of the models. One thing to notice is that 124 samples from the train set were used as a validation set during the training.

The next figure represents the accuracy of the validation and training set. As you can see, there is no overfitting in our models.



first model with just rotation augmentation



second model with more augmentation

3.4. Test Set Accuracy:

Let's evaluate two of the models and see some examples tested on some samples taken from the *test_set*. One important note to consider here is that the following is just a representation of the model's performance regarding the typical machine learning approaches and metrics.

In the evaluation section we perform a biometric evaluation analysis.

Let's now consider some examples from the test set. We have created a function that receives an image as an input and predicts the identity of the given image by using the model. to be able to see more results we have defined a threshold to make a comparison.

```
[ ] THRESHOLD = 0.04

def apply_threshold(dic, threshold=THRESHOLD):

    for (key, value) in dic:
        if value >= threshold:
            print(f'author id: {key} ----------> score: {str(value*100)}%')

[ ] import numpy as np
    from keras.preprocessing.image import img_to_array, load_img

    classes = val_set.class_indices

    def test_on_a_image(path):
      test_image = img_to_array(load_img(path, target_size = (IMAGE_SIZE, IMAGE_SIZE)))
      result = model.predict(np.expand_dims(test_image, axis = 0))
      dic = {}
      for (key, value) in classes.items():
          dic[key] = float(format(result[0][value],'.5f'))

      return apply_threshold(sorted(dic.items(), key=lambda x: x[1], reverse=True)[:10])
```

This is a function that we implemented to perform a prediction over handwritten images.

3.4.1. First Model With only Rotation:

Let's see some examples of the outcome of the function above from the first model that has been considered only the rotation augmentation.

```
[ ] test_on_a_image('/content/test_set/0/a01-007u-0.png')

    author id: 0 ----------> score: 79.88199999999999%
    author id: 346 ----------> score: 9.014999999999999%
    author id: 348 ----------> score: 4.9%

[ ] test_on_a_image('/content/test_set/344/g06-011m-0.png')

    author id: 153 ----------> score: 63.247%
    author id: 344 ----------> score: 12.092%
    author id: 634 ----------> score: 12.011%
    author id: 346 ----------> score: 4.833%

[ ] test_on_a_image('/content/test_set/155/c03-000f-1.png')

    author id: 155 ----------> score: 60.33%
    author id: 150 ----------> score: 7.619%
    author id: 384 ----------> score: 6.5009999999999994%
    author id: 342 ----------> score: 4.027%

    test_on_a_image('/content/test_set/336/g06-011e-2.png')

    author id: 336 ----------> score: 37.208000000000006%
    author id: 333 ----------> score: 10.807%
    author id: 634 ----------> score: 9.17%
    author id: 346 ----------> score: 8.997%
    author id: 347 ----------> score: 6.76%
    author id: 342 ----------> score: 5.318%
    author id: 153 ----------> score: 5.108%
    author id: 635 ----------> score: 4.531000000000001%
```

As you can see, despite the fact that in some cases the score of similarity is not quite high, it is on top of the list. Let's stress the model by all the test samples to see the overall accuracy over the test set in the case of a solo rotation model.

```
[ ] model.load_weights('/content/gdrive/MyDrive/biometric_project/model_solo_rotation/cp-0016.h5')

loss, acc = model.evaluate(test_set)
print("Restored model, accuracy: {:5.2f}%".format(100 * acc))

12/12 [==============================] - 13s 1s/step - loss: 1.8915 - accuracy: 0.4489
Restored model, accuracy: 44.89%
```

3.4.2. Second Model With More Augmentation:

Let's see some examples of the outcome of the function above from the second model that has considered many augmentation options like rotation, brightness, zooming etc.

```
[ ] test_on_a_image('/content/test_set/0/a01-007u-0.png')

    author id: 0 ----------> score: 99.947%

[ ] test_on_a_image('/content/test_set/344/g06-011m-0.png')

    author id: 346 ----------> score: 78.269%
    author id: 153 ----------> score: 11.905000000000001%
    author id: 634 ----------> score: 7.295999999999999%

[ ] test_on_a_image('/content/test_set/155/c03-000f-1.png')

    author id: 155 ----------> score: 86.467%
    author id: 0 ----------> score: 6.039%

test_on_a_image('/content/test_set/336/g06-011e-2.png')

    author id: 0 ----------> score: 47.239%
    author id: 346 ----------> score: 21.458%
    author id: 336 ----------> score: 20.237%
```

Despite the fact that the scores of similarities are higher than the previous model (solo rotation) , we can see that the model makes more mistakes in some identities.

It might be the case that these mistakes are limited only to these identities that we have chosen randomly to analyze; but in any case they are mistakes!

Let’s bring all the test samples to see whether the model is really performing poorly!

```
[ ] model.load_weights('/content/gdrive/MyDrive/biometric_project/model_full_aug/cp-0018.h5')

[ ] loss, acc = model.evaluate(test_set)
    print("Restored model, accuracy: {:5.2f}%".format(100 * acc))

    12/12 [==============================] - 18s 1s/step - loss: 1.3178 - accuracy: 0.6317
    Restored model, accuracy: 63.17%
```

As you can see from the above evaluation, despite the first interpretation, the model performs better than the solo_rotation model which confirms the fact that image augmentation has enhanced the performance.

3.4.3. Observation:

★ While the first model brings more author’ scores and is less sure about them, the second model is more strict to the final list.

★ While, in the first interpretation, it seems that the first model has fewer augmentation options and the result is more acceptable than the second one, but after analyzing all the identities and the whole test set, It becomes clear that more augmentation gives a better robust performance to the system.

★ Of course, these are just personal observations; we must perform a full biometric evaluation to better analyze the systems. Or in another word, these are all the metrics and performances related to machine learning. We need to analyze the system according to the biometric metrics to ensure that whether this Hypothesis is correct or not.

3.4.3. Feature extraction model

So far we have trained three models and seen the performance of the two of them regarding the machine learning evaluation. Those classification models are able to classify an image among **31** different authors. But In biometrics, we need feature extractor modules in order to create and extract the relevant features during the enrollment and save them in our gallery. Therefore, what we are going to do in the evaluation phase is to remove the last layer of our models which is the classification layer in order to access the feature layer.

4. Evaluation:

What we are going to do before starting a biometric evaluation task, is to remove the last layer of our models which is the classification layer in order to access the feature layer. Then once we have the feature extractor models, we will perform an all-against-all method in order to stress the system evaluation.

```
[ ] model.load_weights('/content/gdrive/MyDrive/biometric_project/model_solo_rotation/cp-0016.h5')

[ ] last_feature_layer = model.get_layer(index=24).output

[ ] feature_extractor = keras.Model(
        inputs = model.input,
        outputs = last_feature_layer
    )

    feature_extractor.compile(optimizer=keras.optimizers.SGD(),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

We have 3 feature extractor after applying the above code to each of the classification models. Now everything is ready to extract all the relevant features from the test samples. We have 31 identities in our test set, each of them has 12 templates. The size of the images are still the same 500 x 500.

```
IMAGE_SIZE = 500
BATCH_SIZE = 32
IMAGE_SHAPE = (IMAGE_SIZE, IMAGE_SIZE)
N_IDENTITY=31
N_TEMPLATES = 12
```

We are going to create a similarity matrix. Our matrix should have 372 * 372 dimension (31 * 12 = 372). In the rows we have **probes**.

- Each row (we have 372 rows) is a **31** recognition operation.
- In the column, we have **372** templates.
- Each row contains **1** genuine claim.
- Each row contains number_of_identities - 1 imposters (**30** operations)
- For each operation we considered a group of templates with the same identity with maximum similarity. Then this similarity was compared with the given threshold. This is known as **multiple-template verification** in the literature.

To be able to create a similarity matrix, the first step is to pass the test set into each of the feature extractor models in order to have a prediction for each test set sample. The model predicts a feature vector for each image. Then we gather the result into a list of the feature vectors in which each element in the final list is a feature vector corresponding to each sample in the dataset. The final list is 372 feature vectors that are stored into a list.

To be able to compare feature vectors with each other we defined two measures. The cosine similarity and correlation. In the end, we realized that the cosine similarity has better results in each of the models.

```
def cosine_similarity_matrix(features):
    matrix = []
    cosine_function = lambda a,b : np.dot(a, b) / (np.linalg.norm(a)*np.linalg.norm(b))
    for feature in features:
        similarities = [
            cosine_function(feature, x) for x in features
        ]

        matrix.append(np.array(similarities))
    return np.array(matrix)


def correlation_similarity_matrix(features):
    return np.corrcoef(features)
```

Each of the functions receives a list of feature vectors and returns a matrix of the matched scores between each of the templates. Thus, as we had 372 feature vector, the matrix of the correlation and similarity is a 372x372 matrix.

```
[ ] cosine_similarity       = cosine_similarity_matrix(feature_test_set)
    correlation_similarity  = correlation_similarity_matrix(feature_test_set)

[ ] print(cosine_similarity.shape)
    print(correlation_similarity.shape)

    (372, 372)
    (372, 372)
```

We finally have a 372 * 372 matrix that gives us the ability to perform an all-against-all biometric evaluation task.

One more interesting thing to do, before starting analyzing the matrix, is to create a confusion matrix from the scores which will be able to show how good a biometric system is according to a given threshold.

To create such a matrix we defined a function that implements the pseudo code that we have seen in the slides. The idea behind the function is exactly the same as the course materials.

```python
def confusion(similarity_matrix, threshold):
    confusion_matrix = np.array([
        [0, 0], # [Genuine Match(GM, GA)     False   Non-Match(FNM, FR)]
        [0, 0]  # [False   Match(FM, FA)     Genuine Non-Match(GNM, GR)]
    ])

    for probe in range(0, similarity_matrix.shape[0]): # for each probe of the matrix ===> probe
        for identity in range(N_IDENTITY): # loop over the columns based on the identities

            # each tamplate of the identity in the matrix has a column index 372 * 372
            '''
                id_0 template_0   0 1 2 3 4 5 6 7 8 9 10 11 __ 12 13 14 15 16 17 18 19 20 21 22 23 __ ... 370 371
                id_0 template_1   0 1 2 3 4 5 6 7 8 9 10 11 __ 12 13 14 15 16 17 18 19 20 21 22 23 __ ... 370 371
                id_0 template_3   0 1 2 3 4 5 6 7 8 9 10 11 __ 12 13 14 15 16 17 18 19 20 21 22 23 __ ... 370 371
                        ...
                id_0 template_11  0 1 2 3 4 5 6 7 8 9 10 11 __ 12 13 14 15 16 17 18 19 20 21 22 23 __ ... 370 371
                id_1 template_0   0 1 2 3 4 5 6 7 8 9 10 11 __ 12 13 14 15 16 17 18 19 20 21 22 23 __ ... 370 371
                    ...
                id_30 template_11 0 1 2 3 4 5 6 7 8 9 10 11 __ 12 13 14 15 16 17 18 19 20 21 22 23 __ ... 370 371
            '''
            identity_templates_indexes = np.arange(identity * N_TEMPLATES, (identity + 1) * N_TEMPLATES)
            identity_templates_indexes_excluded = identity_templates_indexes[np.where(identity_templates_indexes!=probe)]
            identity_templates_scores  = similarity_matrix[probe, identity_templates_indexes_excluded]

            if (identity_templates_scores.max() > threshold): # possible genuine acceptance
                if(identity == probe//N_TEMPLATES): # check to see whether the identity id is the same probe identity
                    confusion_matrix[0, 0] +=1      # Genuine Match or Genuine Acceptance (GM, GA)
                else: confusion_matrix[1, 0] +=1.   # False Match or False Acceptance (FM, FA)
            else: # possible imposters
                if(identity == probe//N_TEMPLATES): # check to see whether the identity id is the same probe identity
                    confusion_matrix[0, 1] += 1     # False Non-Match or False Rejection (FNM, FR)
                else: confusion_matrix[1, 1] += 1   # Genuine Non-Match or Genuine Rejection(GNM, GR)

    return confusion_matrix
```

We can give any threshold to this function and see the number of GA, FNM, etc.

```
[ ]  confusion(correlation_similarity, threshold=0.75)

     array([[  229,   143],
            [  522, 10638]])

[ ]  confusion(cosine_similarity, threshold=0.75)

     array([[  231,   141],
            [  612, 10548]])
```

To be able to perform a full evaluation of the biometric on the similarity matrix that we have extracted in the previous pages, we need to identify all the thresholds and stress the system. For this reason we took the advantage of the pyeer library.

This library is using two different scores as the input. One corresponds to imposter scores, and the other is the genuine scores.

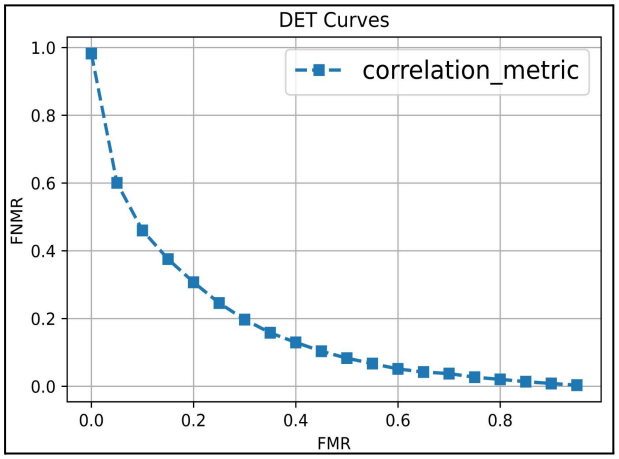
The first thing to do is to separate these two scores according to the matrix. We have implemented a similar confusion matrix but a bit more general with respect to the scores in a way instead of 4 type of values (GA, FA, GR, FR) we extracted 2 types of scores, one for genuine users and another for imposters.

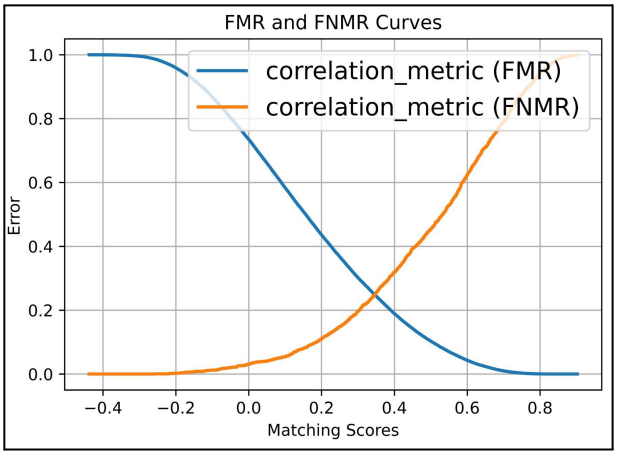
```python
def separated_imposters_genuine_scores(similarity_matrix):
    genuine_scores = []
    imposter_scores = []
    for probe in range(0, similarity_matrix.shape[0]): # for each probe of the matrix ===> probe
        for identity in range(N_IDENTITY): # loop over the columns based on the identities

            identity_templates_indexes = np.arange(identity * N_TEMPLATES, (identity + 1) * N_TEMPLATES)
            identity_templates_indexes_excluded = identity_templates_indexes[np.where(identity_templates_indexes!=probe)]
            identity_templates_scores  = similarity_matrix[probe, identity_templates_indexes_excluded]

            if(identity == probe//N_TEMPLATES):
                genuine_scores.append(identity_templates_scores)
            else:
                imposter_scores.append(identity_templates_scores)

    return np.array(genuine_scores), np.array(imposter_scores)
```

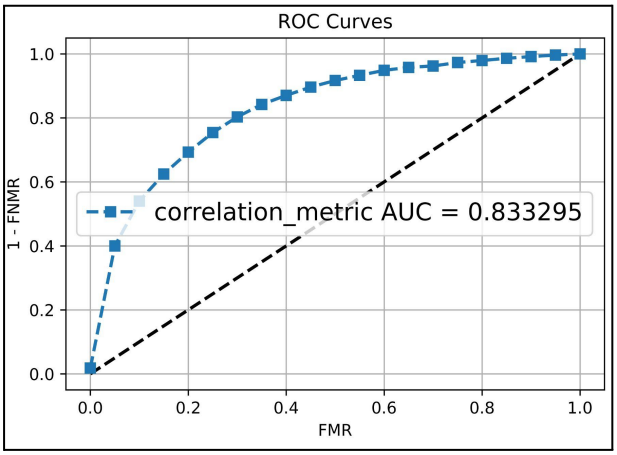
We have everything to start analyzing the evaluation. We have analyzed three models. The results for each of the models are in the following.

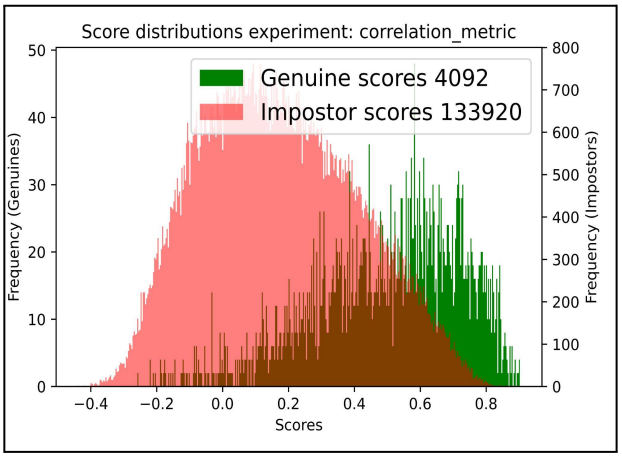
4.1 Model with no augmentation:

This model was the only model in which correlation measure has a better score with respect to cosine similarity. But in any case, the results are not much satisfactory compared to the other models.





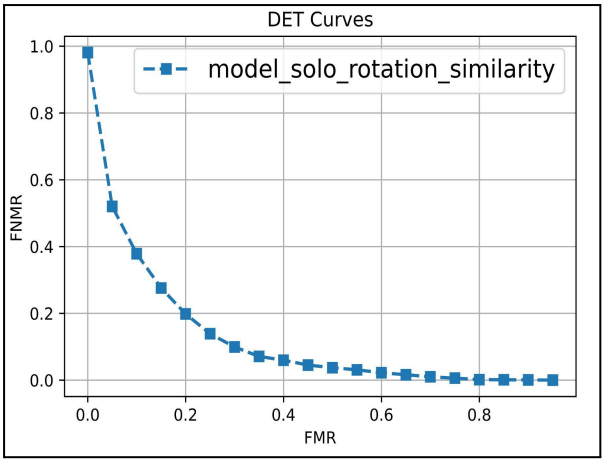


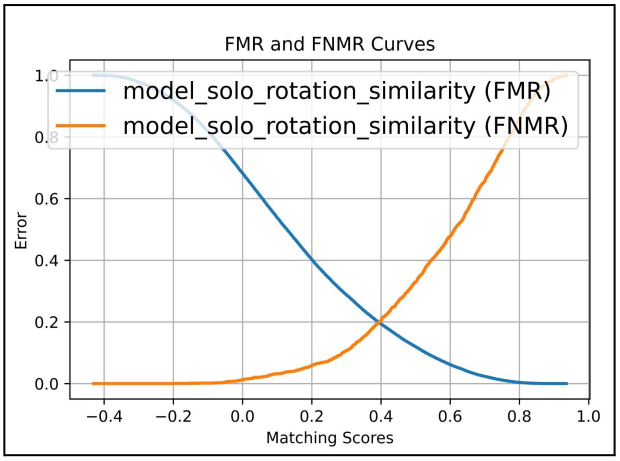


According to the distribution figure, the overlap part is where we have wrong scores.

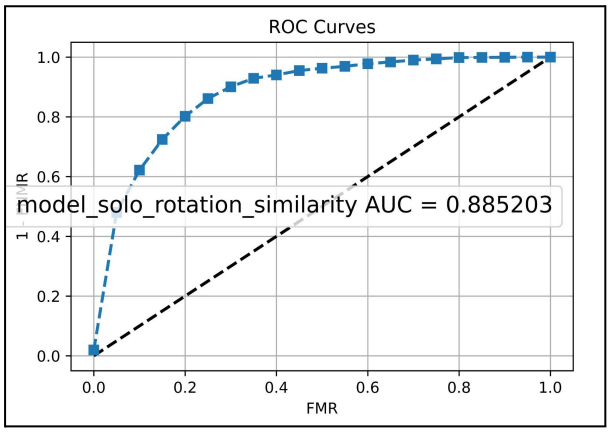
4.2 Model with solo rotation augmentation:

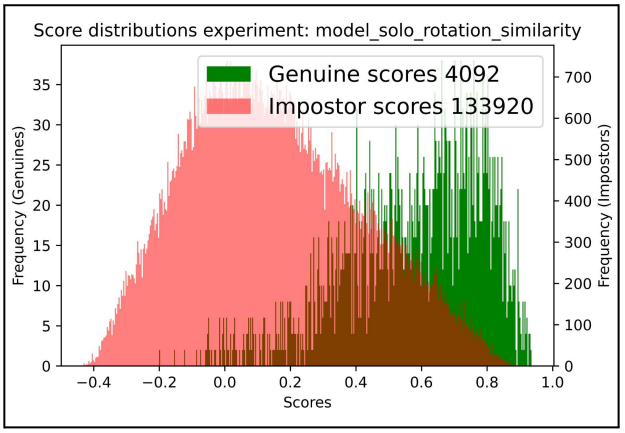
This model had a better result with respect to the previous one both in cosine and correlation measures. Cosine similarity had a better result in this case. According to the outcome of the figures of Merit we realized that adding more augmentation options could be effective to increase the performance.





It is clear that regarding the DET curve, it is bent more to the bottom left which shows a better result. Similarly, the EER in the right picture is equal to 0.2 which is better than the previous model.





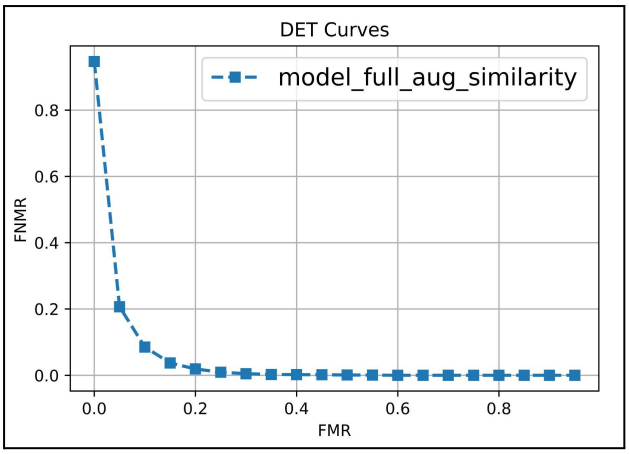
According to the area under ROC curves, we have seen improvement in performance, as it increases from 0.83 in the previous model (no augmentation) up to 0.88.

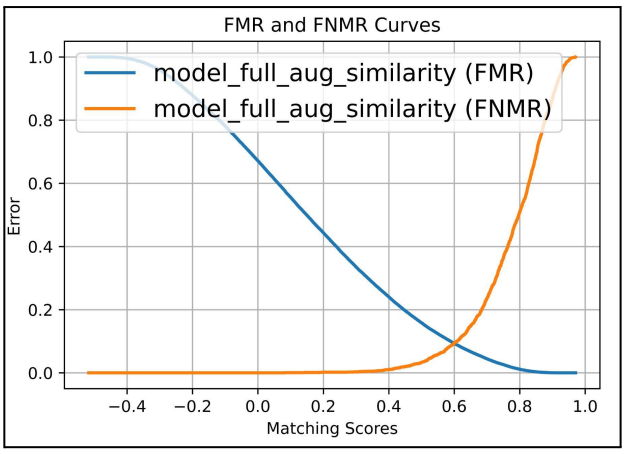
It is clear from the figures above that we have an improvement in performance by applying the augmentation to the input images. We assumed much more augmentation options and retrained the model again.

Then we created a new similarity matrix and performed a new biometric evaluation analysis. The results are quite surprising.

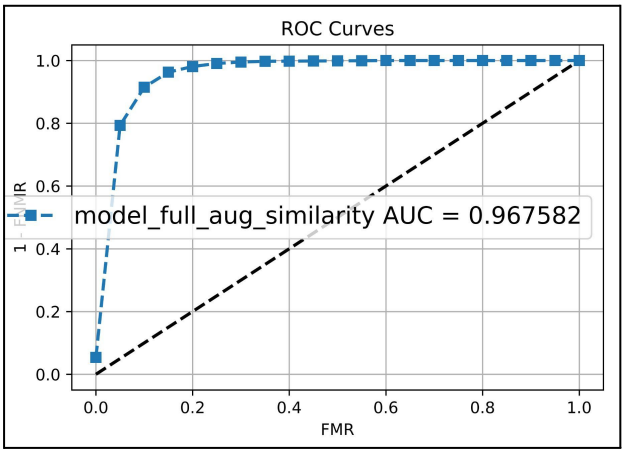
4.3 Model with more augmentation options:

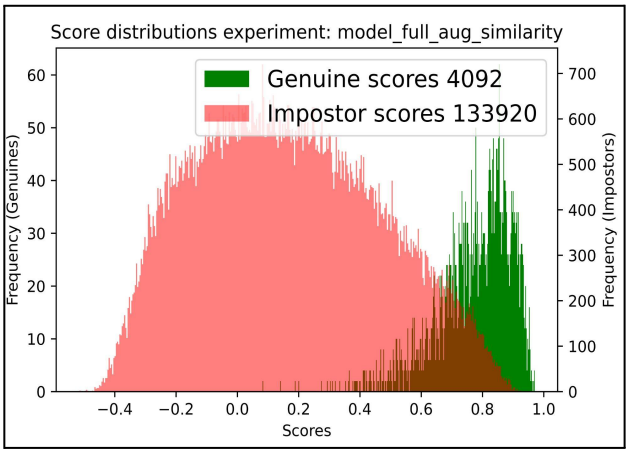
This model was the best model based on the evaluation result. We measure both the cosine and correlation. Cosine similarity had a better result also in this case. According to the outcome of the figures of Merit we realized that by adding more augmentation options we will definitely have much better performances.





We have seen much improvement in the detection error trade-off curve. The EER also dropped drastically to a lower value (almost 0.1).





The ROC curves and area under the curve shows a huge improvement on the performance with respect to the previous models. The AUC is now equal to almost 0.97 which is quite impressive due to the nature of the task.

The overlap part also reduced dramatically compared to the previous ones.

Libraries and References

Handwriting recognition using Deep Learning in Keras

Depite the fact that this paper is for the digit handwriting recognition but it allowed us to build the foundation of our project, it gave us a lot of information about how deal with the task and other useful information.

Handwriting Digit Recognition

Similarly to the previous paper this paper is also for the digit handwriting recognition but it also gave us a lot of useful information about machine learning techniques and deep learning approaches.

Gdown: Used to download the raw files from the google drive.

OpenCV-Python: is used for some image preprocessing.

Tqdm: tqdm is a library in Python which is used for creating Progress Meters or Progress Bars. Here, it is used for measuring the loops taking time.

Pandas: used inorder to work with the annotation information

Pytest-shutil: is used to work with files like move, copy and delete functionalities

Tensorflow: is used to train the model as well as image augmentation

Keras: Keras is an open-source software library that provides a Python interface for artificial neural networks.

Matplotlib: is used to visualize some of the important figures and distribution

Numpy: is used to perform some of the functionalities like correlation

Pyeer: it is a python package for biometric systems performance evaluation.