Digital Image Processing SWE1010 – J Component Signature Forgery Detection

Rajveer Kalsi (20MIS1161), Kshitij Kiran Thakare (20MIS1136), Soni Singh (20MIS1137), Shashank Vinayak Burhade (20MIS1050)
Vellore Institute of Technology - Chennai, rajveer.kalsi2020,kshitijkiran.thakare2020, soni.singh2020, shashankvinayak.b2020@vitstudent.ac.in

***Abstract* - Every person has his/her own unique signature that is used mainly for the purposes of personal identification and verification of important documents or legal transactions. There are two kinds of signature verification: static and dynamic. Static(off-line) verification is the process of verifying an electronic or document signature after it has been made, while dynamic(on-line) verification takes place as a person creates his/her signature on a digital tablet or a similar device. Offline signature verification is not efficient and slow for a large number of documents. To overcome the drawbacks of offline signature verification, we have seen a growth in online biometric personal verification such as fingerprints, eye scan etc. In this paper we created CNN model using python for offline signature and after training and validating, the accuracy of testing was 99.70%.**

1. Introduction

Signature is the most commonly used tool for identification of individuals as personal verification in financial institutions. Even in the digital age, people still use their signatures as a primary form of authentication for a range of transactions. Their signatures authorize checks, new account paperwork, loan documents, and more, and to minimize the risk of fraud, financial institutions need the right solutions to detect forgeries quickly and accurately. Therefore, the need for authentication increases rapidly for various security purposes.

In our project, we are planning to implement offline verification of signatures by using different geometric measures. If the absolute difference between the geometric parameters of original signature and the verification signature is greater than a predefined threshold then the signature would be identified as forgery. We aim to develop an efficient system for signature verification and forgery detection based on image processing techniques using python libraries like TensorFlow. A solution is presented where the model is trained with a dataset of signatures, and predictions are made as to whether a provided signature is genuine or forged.

In recent days different methods are developed for made image reliable and secure that is analogous to watermarking like message authentication code, image checksum, image hash, and image shielding. Passive image forensics is a challenging task in image processing techniques. It is not a particular method for all cases but different methods each can detect the special forgery. The stream of passive tempering detection is to deal with analyzing raw image based on different statistics and semantics of an image content to localize tampering of image.

There are several types of image forgery that include image retouching, image splicing, copy and move attack. Image retouching is considered a minimum harmful type of digital image forgery. An original image does not significantly change, but they reduced some features of the original image. This technique is used to edit the image for a popular magazine. This type of image forgery is located in all magazine covers and also it used to improve the specific features of an image.

On the other hand, Image splicing or photo montage refers to make a forgery image and it is more aggressive than image retouching. Image splicing is an easy process and it pastes the regions from isolated sources. This method is referred to as paste-up formed by sticking together the image by using digital tools like Photoshop. This technique is a group of two or more images that are combined to generate fake images. However, copy and move attack is also one of the popular and difficult images tampering technique. It required the cover part of a similar image to add or remove the information. Copy and move attack, the aim is to hide some information in the original image. The detection of the forged image from the original one is very hard. The naked eye is not able to identify the tampered region from a forged image. The image tampering is a general manipulation of digital images. Traditional block-based forgery detection methodologies are categorized as the input images into overlapping and regular image blocks and also tampered regions are identified by matching blocks of pixel or transform coefficient.

Normally, the image forgery detection is performed by using the following techniques: JPEG quantization tables, Chromatic Aberration, Lighting, Camera Response Function (CRF), Bi coherence and higher-order statistics, and Robust matching. The digital cameras encode the images based on JPEG compression, which configures the devices at various compression level. Then, the sign of image tampering is evaluated by analyzing the inconsistency of lateral chromatic aberration . In which, the average angular between the local and global parameters is computed for every pixel in the image. If the average value exceeds the threshold, it is stated that the deviation is unpredictable in the image due to the forgery of the image. Then, for each object in the image, the inconsistencies and the illuminating light source is detected to identify the forgery. Typically, different measurements such as infinite, local and multiple are considered for determining the error rate. Then, the CRF is mainly used to expose the image splicing instituted on the geometry invariant of the image. In which, the suspected boundary is identified within each region of the image, and it is validated for identifying the inconsistencies. The bi-coherence features are widely used for detecting the splicing on images that estimate the mean of magnitude and phase entropy for augmenting the images. Moreover, it extracts the features for the authentic counterpart and incorporates it to capture the characteristics of various object interfaces. Finally, the exact replicas are identified by matching the features concerning the block size, which is done by the use of robust matching. But, it requires the human intervention for interpreting the output of replicas detection. Generally, the region duplication is performed on the image based on the geometrical and illumination adjustments. It is a very simple operation in which a continuous portion of pixels is copied and pasted on some other location in the image. This paper is fully focused on the detailed investigation of the image forgery detection mechanisms.

Objective: To build a system to identify if the signature is forged or real. Signature recognition and verification is a system which is capable of addressing two things namely identification of the signature owner and decision whether the signature is real or forged. This project aims at solving the major problems of forged signatures.

2. Digital Image Forgery Detection Methods

Typically, the methodologies used for forgery detection are classified into two types such as the splicing and copy-move techniques are the categories of the passive technique. The description of these techniques are investigated in the following sub-sections.

2.1. Image splicing

Image splicing is a kind of forgery detection method, in which a single image is created based on the combination of two or more images. It is also termed as image composition, in which various image manipulation operations are performed. Typically, many inconsistencies may be created in the image features due to the splicing operation. In this technique, the composition between the two images is estimated and incorporated for creating a fake image. Based on the image block content, the difference between the illumination and reference illuminate color is estimated. In this digital image forgery, it is very difficult to extract the exact shape of the image. Typically, the image splicing method is categorized into two types such as boundary-based and region based. Here, the features are extracted from the chromatic channel for capturing the tampering artifacts. The utilization of forgery detection approach for detecting the splicing in the digital images. Here, the small inconsistencies in the motion blur are detected by analyzing the special characteristics of image gradients. The stages involved in this detection are image subdivision, motion blur estimation, smoothing, blur computation, interpolation and segmentation. We a machine learning algorithm for detecting the image splices. To increase the effect of photorealism, an image splicing operation is performed with the operations of color and brightness adjustment.

2.2. Copy-Move Method

Among the other forgery methods, the copy-move method an extensively used type of image tampering, where the specific portion is copied and pasted on some other region. The main motive of this method is to hide a significant element or highlight a precise object. We implemented a proficient method for detecting the copy-move forgery. The authors stated that the block matching procedure is used to detect this type of forgery by separating the image into overlapping chunks. Also, it identifies the duplicated connected image blocks by finding the distance between the neighbor blocks. For taking the forgery decision, only the duplicate blocks detection is not enough, because the natural images have many similar blocks. Moreover, the Fourier Mellin Transform (FMT) is used to perform the operations like scaling, translation, and rotation for image forgery detection. This detection methodology can detect blur degradation, noise, and some other arbitrary changes in the duplicate image regions like noise addition and gamma correction gamma is a non-linear adjustment to individual pixel values. The steps involved in this method are image tiling with overlapping, representation blur moment invariants, transformation, similarity analysis, and map creation for duplication region detection. Moreover, the dimensionality of blocks was reduced by using the principle component transformation. We employed a Dyadic undecorated Wavelet Transformation (Dew) technique for detecting blind copy-move image forgery detection. This transformation technique aimed to extract the low frequency and high-frequency components by estimating the similarity between the blocks. Moreover, the Euclidean distance is computed between every pair of blocks in the image. Then, the match is identified by computing the threshold value between the sorted lists. In the wavelet transformation, the down sampling process is not involved, and the coefficients are not shrunk between the scales. We aimed to detect the copy-move forgery by the use of expanding block algorithms. Also, it intended to identify the duplicated regions in the image by estimating the size and shape. It is stated that the copy-move forgery is performed for hiding the region of the image by wrapper it with a duplicate image. Still, recognizing the forged region is extremely intricate due to the precise copy of another region. This detection mechanism contains the stages of feature extraction, comparison, and similarity estimation for taking copy decisions. As shown in figure below the procedure of copy move technique first step the input image preprocessed, second step block division, third step feature extracted, last step the blocks which carry same feature triggered and mapped as a forgery.

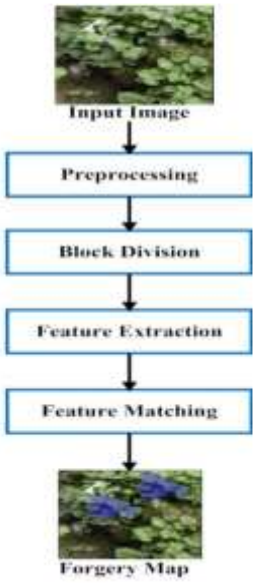
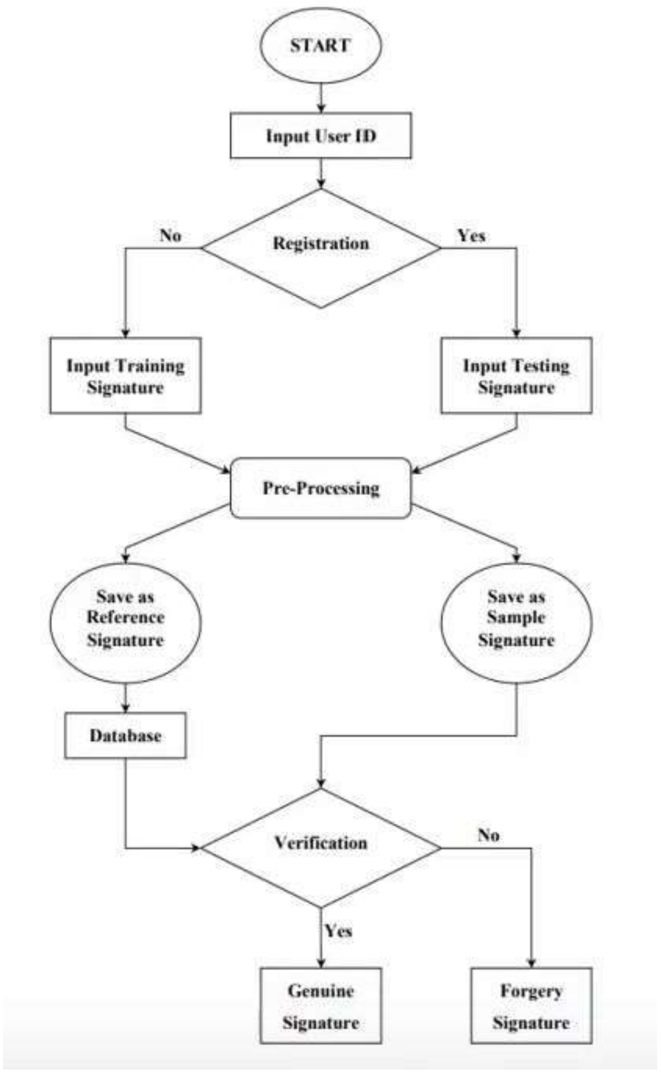


Fig. Procedure of Copy Move technique

3. Implementation



Methodology:

- This signature verification system is started with the input of the User ID.
- The user is required to key in his/her ID to the system. Here, the system will decide whether the user ID is registered or not. If the user ID is not registered, the user has to drop down 5 sets of signatures for training purposes. The signature will then proceed to pre processing and feature extraction stage.
- After this stage, the input training signature will be saved as reference signature in the database.
- If the user ID is registered, the user will then be asked to sign for testing. Then, the input testing signature will proceed to the Pre-processing stage.
- After this stage, the input testing signature will be saved as sample signature and proceed to the verification stage.

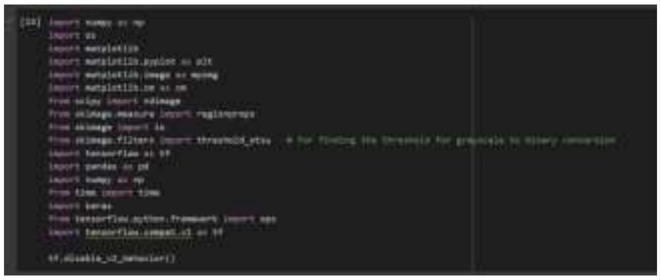
● In the Verification stage, the sample signature will be compared with the reference signature which is stored in the database.

● If the difference between two signatures does not exceed the Threshold value, the sample signature will be accepted as genuine signature and vice versa.

We executed our code using Jupyter notebook. The following are the steps and packages used:

3.1 Import the packages and libraries

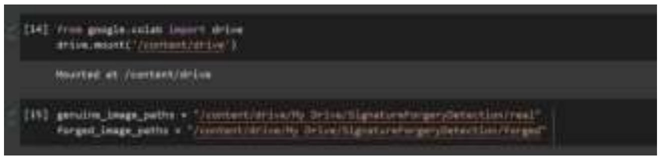
First, we import all the required packages and libraries that we used in the project.



```
[13] import numpy as np
     import os
     import matplotlib
     import matplotlib.pyplot as plt
     import matplotlib.image as mpimg
     import matplotlib.cm as cm
     from scipy import ndimage
     from skimage.measure import regionprops
     from skimage import io
     from skimage.filters import threshold_otsu   # For finding the threshold for grayscale to binary conversion
     import tensorflow as tf
     import pandas as pd
     import numpy as np
     from time import time
     import keras
     from tensorflow.python.framework import ops
     import tensorflow.compat.v1 as tf

     tf.disable_v2_behavior()
```

3.2 Define the path of the signatures

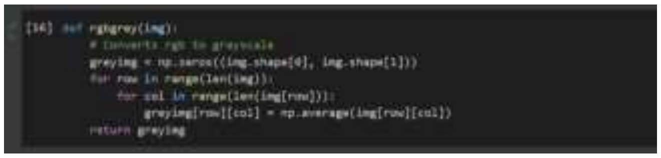
Two paths are created, one for the genuine signatures and the other for the forged signatures which are taken from database of the signatures.



```
[14] from google.colab import drive
     drive.mount('/content/drive')

     Mounted at /content/drive

[15] genuine_image_paths = "/content/drive/My Drive/SignatureForgeryDetection/real"
     forged_image_paths = "/content/drive/My Drive/SignatureForgeryDetection/forged"
```

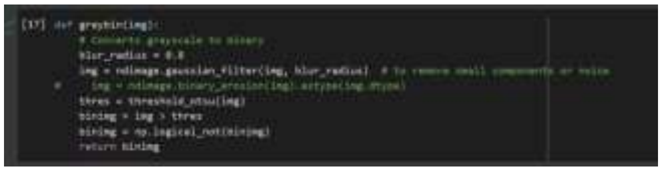
3.3 Pre-processing the inputted signature image

First, we convert the given input image into a grayscale format so that we now have the images only in black, white, and shades of grey colours which will provide us the raw image. This can be helpful to increase the efficiency of the as feature extraction will now be able to give more accurate values.

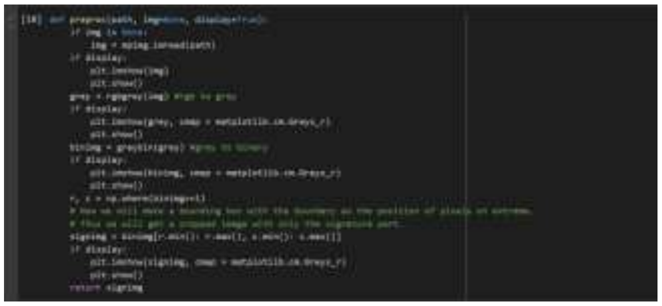


```
[16] def rgbgrey(img):
         # Converts rgb to grayscale
         greyimg = np.zeros((img.shape[0], img.shape[1]))
         for row in range(len(img)):
             for col in range(len(img[row])):
                 greyimg[row][col] = np.average(img[row][col])
         return greyimg
```

Second, we then remove any noise disturbances using the Gaussian filter in the image and then convert it into the binary format as it will be very easy for us to consider and compare them later on.



```
[17] def greybin(img):
         # Converts grayscale to binary
         blur_radius = 0.8
         img = ndimage.gaussian_filter(img, blur_radius)  # to remove small components or noise
     #     img = ndimage.binary_erosion(img).astype(img.dtype)
         thres = threshold_otsu(img)
         binimg = img > thres
         binimg = np.logical_not(binimg)
         return binimg
```

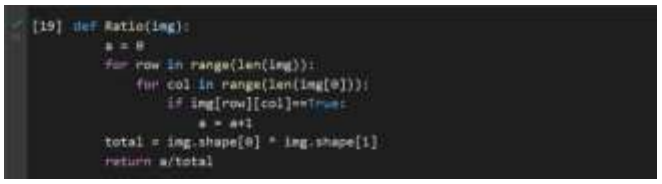


```
[18] def preproc(path, img=None, display=True):
         if img is None:
             img = mpimg.imread(path)
         if display:
             plt.imshow(img)
             plt.show()
         grey = rgbgrey(img) #rgb to grey
         if display:
             plt.imshow(grey, cmap = matplotlib.cm.Greys_r)
             plt.show()
         binimg = greybin(grey) #grey to binary
         if display:
             plt.imshow(binimg, cmap = matplotlib.cm.Greys_r)
             plt.show()
         r, c = np.where(binimg==1)
         # Now we will make a bounding box with the boundary as the position of pixels on extreme.
         # Thus we will get a cropped image with only the signature part.
         signimg = binimg[r.min(): r.max(), c.min(): c.max()]
         if display:
             plt.imshow(signimg, cmap = matplotlib.cm.Greys_r)
             plt.show()
         return signimg
```

3.4 Feature extraction

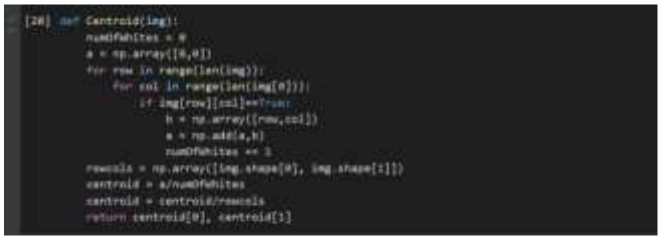
We used the following parameters to differentiate between genuine and real signatures.

Ratio: This calculates the relationship between the height and width of the image. It basically describes the shape of the image.



```
[19] def Ratio(img):
         a = 0
         for row in range(len(img)):
             for col in range(len(img[0])):
                 if img[row][col]==True:
                     a = a+1
         total = img.shape[0] * img.shape[1]
         return a/total
```

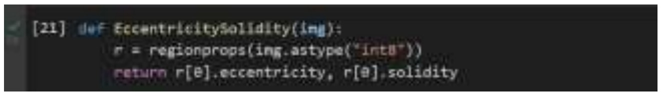
Centroid: The centroid of the image is often considered the intersection point of all the hyperplanes of symmetry within the image, by doing this we get the centre point of the image.



```
[20] def Centroid(img):
         numOfWhites = 0
         a = np.array([0,0])
         for row in range(len(img)):
             for col in range(len(img[0])):
                 if img[row][col]==True:
                     b = np.array([row,col])
                     a = np.add(a,b)
                     numOfWhites += 1
         rowcols = np.array([img.shape[0], img.shape[1]])
         centroid = a/numOfWhites
         centroid = centroid/rowcols
         return centroid[0], centroid[1]
```

Eccentricity: The eccentricity of an ellipse is the ratio of the distance between its foci to the length of its main axis. The value is always in the range of 0 to 1.

Solidity: The area of an image object is divided by the area of its bounding rectangle to determine its extent. The area of an image object divided by the area of its convex hull determines its solidity. A fraction of the size of your actual picture.

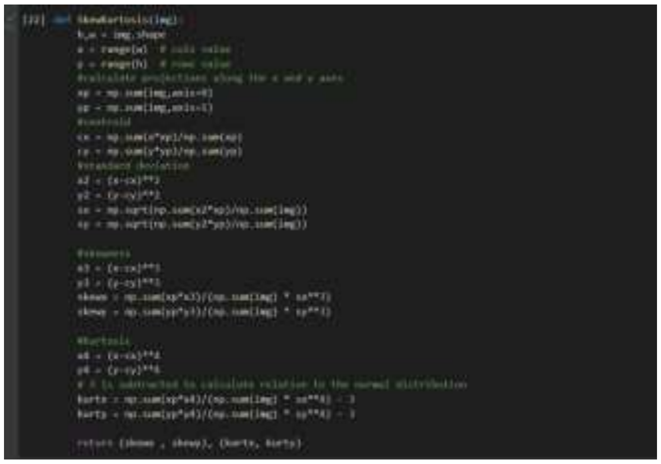


```
[21] def EccentricitySolidity(img):
         r = regionprops(img.astype("int8"))
         return r[0].eccentricity, r[0].solidity
```

Kurtosis: Kurtosis is a statistical indicator of how often the tails of distribution vary from the tails of the regular distribution. It assesses the sharpness of a frequency distribution curve.

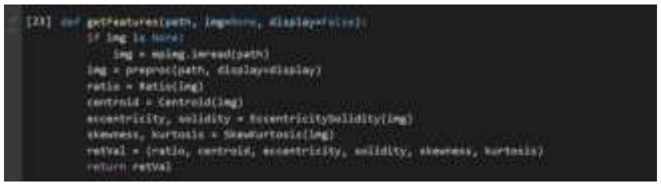
Skew: When transferring data to a digital format, skew detection is one of the first operations performed on scanned documents. Its aim is to align an image before it is processed because text segmentation and recognition methods depend on correctly aligned next lines.

We calculate the values using the respective formulas to obtain the required features.



```
[22] def SkewKurtosis(img):
        h,w = img.shape
        x = range(w)  # cols value
        y = range(h)  # rows value
        #calculate projections along the x and y axes
        xp = np.sum(img,axis=0)
        yp = np.sum(img,axis=1)
        #centroid
        cx = np.sum(x*xp)/np.sum(xp)
        cy = np.sum(y*yp)/np.sum(yp)
        #standard deviation
        x2 = (x-cx)**2
        y2 = (y-cy)**2
        sx = np.sqrt(np.sum(x2*xp)/np.sum(img))
        sy = np.sqrt(np.sum(y2*yp)/np.sum(img))

        #skewness
        x3 = (x-cx)**3
        y3 = (y-cy)**3
        skewx = np.sum(xp*x3)/(np.sum(img) * sx**3)
        skewy = np.sum(yp*y3)/(np.sum(img) * sy**3)

        #Kurtosis
        x4 = (x-cx)**4
        y4 = (y-cy)**4
        # 3 is subtracted to calculate relation to the normal distribution
        kurtx = np.sum(xp*x4)/(np.sum(img) * sx**4) - 3
        kurty = np.sum(yp*y4)/(np.sum(img) * sy**4) - 3

        return (skewx , skewy), (kurtx, kurty)
```

We call a function which calls all the above functions through which we can get the features namely ratio, centroid, eccentricity, solidity, skewness and kurtosis.



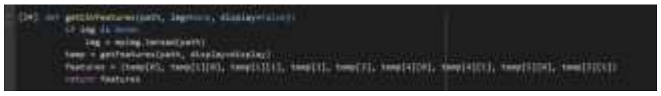
```
[23] def getFeatures(path, img=None, display=False):
        if img is None:
            img = mpimg.imread(path)
        img = preproc(path, display=display)
        ratio = Ratio(img)
        centroid = Centroid(img)
        eccentricity, solidity = EccentricitySolidity(img)
        skewness, kurtosis = SkewKurtosis(img)
        retVal = (ratio, centroid, eccentricity, solidity, skewness, kurtosis)
        return retVal
```

We save all the features in a comma separated value file and retrieve all the features from the same csv file.

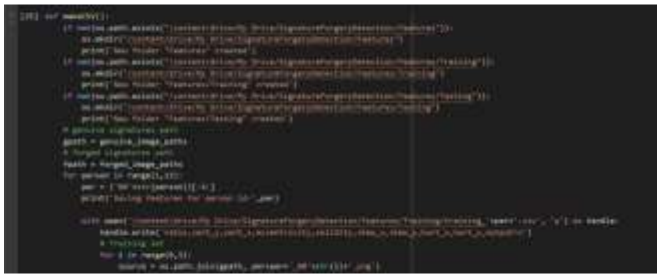
```
[23] def getFeatures(path, img=None, display=False):
        if img is None:
            img = mpimg.imread(path)
        img = preproc(path, display=display)
        ratio = Ratio(img)
        centroid = Centroid(img)
        eccentricity, solidity = EccentricitySolidity(img)
        skewness, kurtosis = SkewKurtosis(img)
        retVal = (ratio, centroid, eccentricity, solidity, skewness, kurtosis)
        return retVal
```

3.5 Saving the features

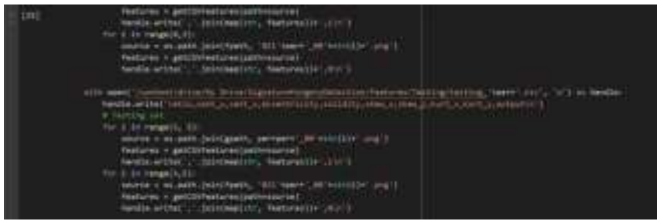
We save the feature we implemented in the process by making a CSV file in a directed path and move on to the next point.



```
[24] def getCSVFeatures(path, img=None, display=False):
        if img is None:
            img = mpimg.imread(path)
        temp = getFeatures(path, display=display)
        features = (temp[0], temp[1][0], temp[1][1], temp[2], temp[3], temp[4][0], temp[4][1], temp[5][0], temp[5][1])
        return features
```



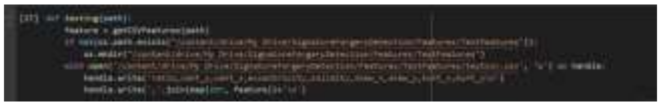
```
[25] def makeCSV():
        if not(os.path.exists('/content/drive/My Drive/SignatureForgeryDetection/Features')):
            os.mkdir('/content/drive/My Drive/SignatureForgeryDetection/Features')
            print('New folder "Features" created')
        if not(os.path.exists('/content/drive/My Drive/SignatureForgeryDetection/Features/Training')):
            os.mkdir('/content/drive/My Drive/SignatureForgeryDetection/Features/Training')
            print('New folder "Features/Training" created')
        if not(os.path.exists('/content/drive/My Drive/SignatureForgeryDetection/Features/Testing')):
            os.mkdir('/content/drive/My Drive/SignatureForgeryDetection/Features/Testing')
            print('New folder "Features/Testing" created')
        # genuine signatures path
        gpath = genuine_image_paths
        # forged signatures path
        fpath = forged_image_paths
        for person in range(1,13):
            per = ('00'+str(person))[-3:]
            print('Saving features for person id-',per)

            with open('/content/drive/My Drive/SignatureForgeryDetection/Features/Training/training_'+per+'.csv', 'w') as handle:
                handle.write('ratio,cent_y,cent_x,eccentricity,solidity,skew_x,skew_y,kurt_x,kurt_y,output\n')
                # Training set
                for i in range(0,3):
                    source = os.path.join(gpath, per+per+'_00'+str(i)+'.png')
```



```
[25]                features = getCSVFeatures(path=source)
                    handle.write(','.join(map(str, features))+',1\n')
                for i in range(0,3):
                    source = os.path.join(fpath, '021'+per+'_00'+str(i)+'.png')
                    features = getCSVFeatures(path=source)
                    handle.write(','.join(map(str, features))+',0\n')

            with open('/content/drive/My Drive/SignatureForgeryDetection/Features/Testing/testing_'+per+'.csv', 'w') as handle:
                handle.write('ratio,cent_y,cent_x,eccentricity,solidity,skew_x,skew_y,kurt_x,kurt_y,output\n')
                # Testing set
                for i in range(3, 5):
                    source = os.path.join(gpath, per+per+'_00'+str(i)+'.png')
                    features = getCSVFeatures(path=source)
                    handle.write(','.join(map(str, features))+',1\n')
                for i in range(3,5):
                    source = os.path.join(fpath, '021'+per+'_00'+str(i)+'.png')
                    features = getCSVFeatures(path=source)
                    handle.write(','.join(map(str, features))+',0\n')
```

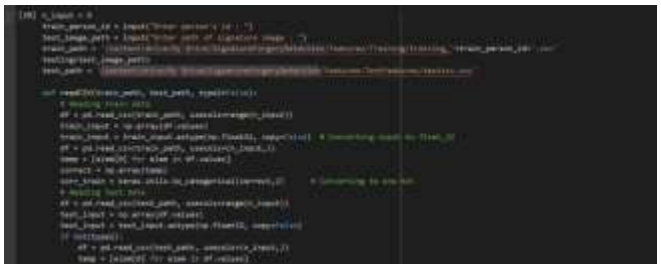
3.6 TensorFlow Model

When the input id is already present in the database, we created a function to input the testing signatures.



```
[27] def testing(path):
        feature = getCSVFeatures(path)
        if not(os.path.exists('/content/drive/My Drive/SignatureForgeryDetection/Features/TestFeatures')):
            os.mkdir('/content/drive/My Drive/SignatureForgeryDetection/Features/TestFeatures')
        with open('/content/drive/My Drive/SignatureForgeryDetection/Features/TestFeatures/testcsv.csv', 'w') as handle:
            handle.write('ratio,cent_y,cent_x,eccentricity,solidity,skew_x,skew_y,kurt_x,kurt_y\n')
            handle.write(','.join(map(str, feature))+'\n')
```

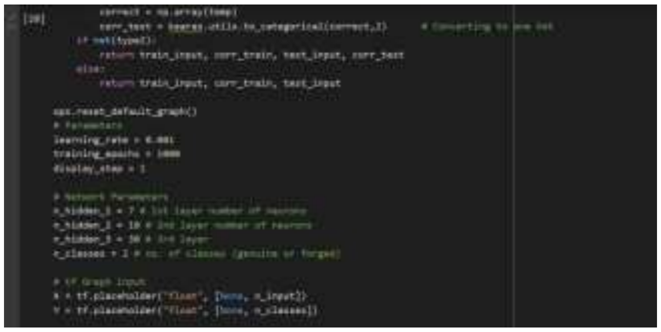
We need to enter the person's id and the path of the signature image to find out whether the signature is genuine or forged.



```
[28] n_input = 9
     train_person_id = input("Enter person's id : ")
     test_image_path = input("Enter path of signature image : ")
     train_path = '/content/drive/My Drive/SignatureForgeryDetection/Features/Training/training_'+train_person_id+'.csv'
     testing(test_image_path)
     test_path = '/content/drive/My Drive/SignatureForgeryDetection/Features/TestFeatures/testcsv.csv'

     def readCSV(train_path, test_path, type2=False):
         # Reading train data
         df = pd.read_csv(train_path, usecols=range(n_input))
         train_input = np.array(df.values)
         train_input = train_input.astype(np.float32, copy=False)  # Converting input to float_32
         df = pd.read_csv(train_path, usecols=(n_input,))
         temp = [elem[0] for elem in df.values]
         correct = np.array(temp)
         corr_train = keras.utils.to_categorical(correct,2)      # Converting to one hot
         # Reading test data
         df = pd.read_csv(test_path, usecols=range(n_input))
         test_input = np.array(df.values)
         test_input = test_input.astype(np.float32, copy=False)
         if not(type2):
             df = pd.read_csv(test_path, usecols=(n_input,))
             temp = [elem[0] for elem in df.values]
```

Here we are reading training data where we read data from our extensive dataset. We use a correlation train for the same which uses keras.

Keras provides a numpy utility library, which provides functions to perform actions on numpy arrays. Using the method to_categorical(), a numpy array (or) a vector which has integers that represent different categories, can be converted into a numpy array (or) a matrix which has binary values and has columns equal to the number of categories in the data.



```
[20]            correct = np.array(temp)
                corr_test = keras.utils.to_categorical(correct,2)       # Converting to one hot
            if not(type2):
                return train_input, corr_train, test_input, corr_test
            else:
                return train_input, corr_train, test_input

        ops.reset_default_graph()
        # Parameters
        learning_rate = 0.001
        training_epochs = 1000
        display_step = 1

        # Network Parameters
        n_hidden_1 = 7 # 1st layer number of neurons
        n_hidden_2 = 10 # 2nd layer number of neurons
        n_hidden_3 = 30 # 3rd layer
        n_classes = 2 # no. of classes (genuine or forged)

        # tf Graph input
        X = tf.placeholder("float", [None, n_input])
        Y = tf.placeholder("float", [None, n_classes])
```

Correlation is an indication about the changes between two variables. We can plot correlation matrices to show which variable is having a high or low correlation in respect to another variable.

Parameters:

learning_rate = 0.001

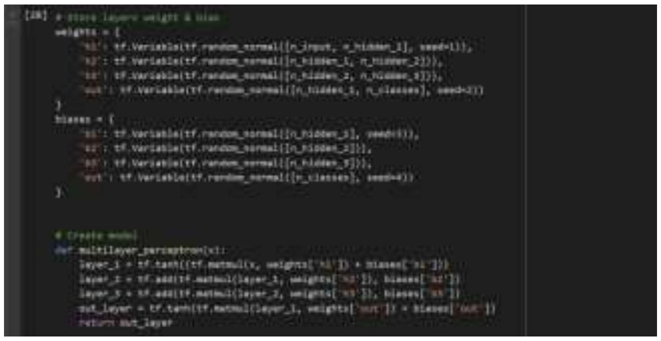
In machine learning and statistics, the learning rate is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function.

It is important to define the number of passes to optimise the resources and the model.

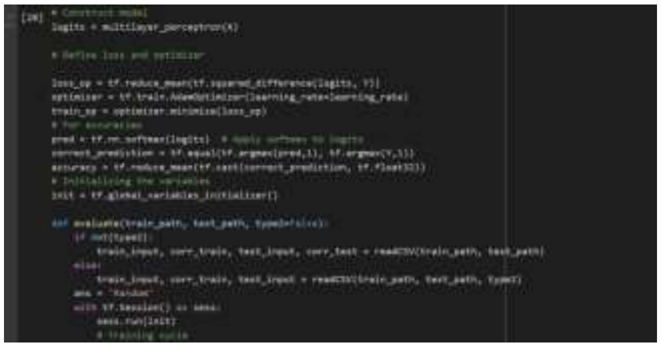
Store layers weight & bias:

Each of these levels has its own set of weights and biases.

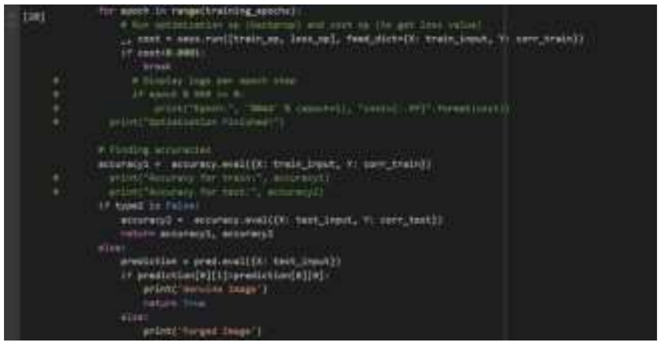
The weights and bias are added to the input when it is passed between neurons. Weights identify the degree of the interaction between two variables.



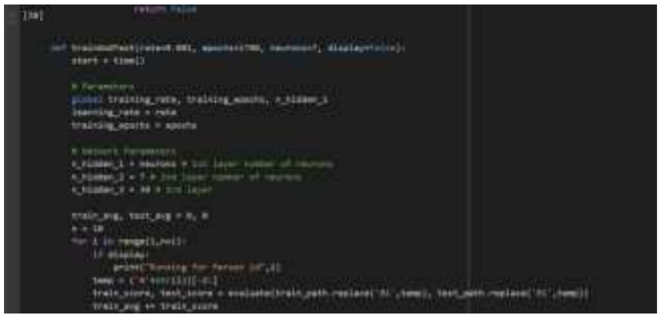
```
[20] # Store layers weight & bias
     weights = {
         'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1], seed=1)),
         'h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2])),
         'h3': tf.Variable(tf.random_normal([n_hidden_2, n_hidden_3])),
         'out': tf.Variable(tf.random_normal([n_hidden_1, n_classes], seed=2))
     }
     biases = {
         'b1': tf.Variable(tf.random_normal([n_hidden_1], seed=3)),
         'b2': tf.Variable(tf.random_normal([n_hidden_2])),
         'b3': tf.Variable(tf.random_normal([n_hidden_3])),
         'out': tf.Variable(tf.random_normal([n_classes], seed=4))
     }


     # Create model
     def multilayer_perceptron(x):
         layer_1 = tf.tanh((tf.matmul(x, weights['h1']) + biases['b1']))
         layer_2 = tf.add(tf.matmul(layer_1, weights['h2']), biases['b2'])
         layer_3 = tf.add(tf.matmul(layer_2, weights['h3']), biases['b3'])
         out_layer = tf.tanh(tf.matmul(layer_1, weights['out']) + biases['out'])
         return out_layer
```



```
[20] # Construct model
     logits = multilayer_perceptron(X)

     # Define loss and optimizer

     loss_op = tf.reduce_mean(tf.squared_difference(logits, Y))
     optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
     train_op = optimizer.minimize(loss_op)
     # For accuracies
     pred = tf.nn.softmax(logits)  # Apply softmax to logits
     correct_prediction = tf.equal(tf.argmax(pred,1), tf.argmax(Y,1))
     accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
     # Initializing the variables
     init = tf.global_variables_initializer()

     def evaluate(train_path, test_path, type2=False):
         if not(type2):
             train_input, corr_train, test_input, corr_test = readCSV(train_path, test_path)
         else:
             train_input, corr_train, test_input = readCSV(train_path, test_path, type2)
         ans = 'Random'
         with tf.Session() as sess:
             sess.run(init)
             # Training cycle
```



```
[20]         for epoch in range(training_epochs):
                 # Run optimization op (backprop) and cost op (to get loss value)
                 _, cost = sess.run([train_op, loss_op], feed_dict={X: train_input, Y: corr_train})
                 if cost<0.0001:
                     break
     #               # Display logs per epoch step
     #               if epoch % 999 == 0:
     #                   print("Epoch:", '%04d' % (epoch+1), "cost={:.9f}".format(cost))
     #           print("Optimization Finished!")

             # Finding accuracies
             accuracy1 =  accuracy.eval({X: train_input, Y: corr_train})
     #           print("Accuracy for train:", accuracy1)
     #           print("Accuracy for test:", accuracy2)
             if type2 is False:
                 accuracy2 =  accuracy.eval({X: test_input, Y: corr_test})
                 return accuracy1, accuracy2
             else:
                 prediction = pred.eval({X: test_input})
                 if prediction[0][1]>prediction[0][0]:
                     print('Genuine Image')
                     return True
                 else:
                     print('Forged Image')
```

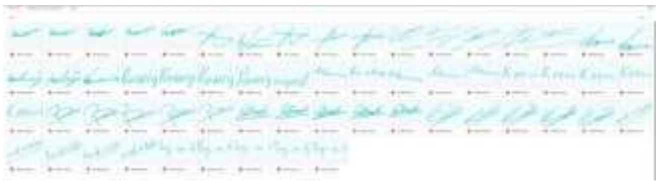


```
[20]                 return False


     def trainAndTest(rate=0.001, epochs=1700, neurons=7, display=False):
         start = time()

         # Parameters
         global training_rate, training_epochs, n_hidden_1
         learning_rate = rate
         training_epochs = epochs

         # Network Parameters
         n_hidden_1 = neurons # 1st layer number of neurons
         n_hidden_2 = 7 # 2nd layer number of neurons
         n_hidden_3 = 30 # 3rd layer

         train_avg, test_avg = 0, 0
         n = 10
         for i in range(1,n+1):
             if display:
                 print("Running for Person id",i)
             temp = ('0'+str(i))[-2:]
             train_score, test_score = evaluate(train_path.replace('01',temp), test_path.replace('01',temp))
             train_avg += train_score
```

4. Result and Output

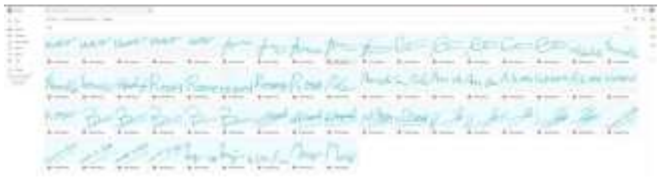
Dataset for real signatures:

(4 IDs of group members and remaining 8 IDs from Kaggle) Each ID has 5 signatures.

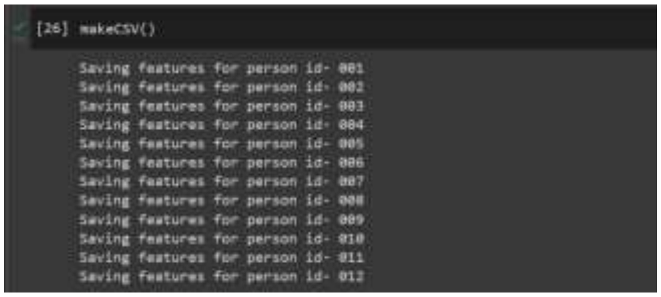


Dataset for forged signatures:

(4 IDs of group members and remaining 8 IDs from Kaggle) Each ID has 5 signatures.



Saving features for all 12 ids stored:

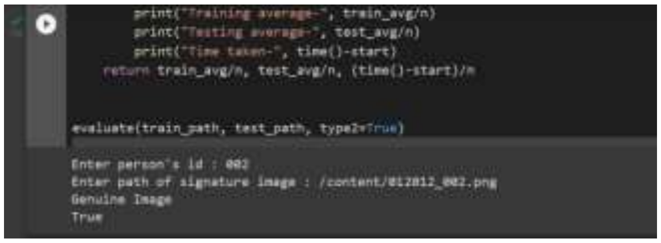


```
[26] makeCSV()

     Saving features for person id- 001
     Saving features for person id- 002
     Saving features for person id- 003
     Saving features for person id- 004
     Saving features for person id- 005
     Saving features for person id- 006
     Saving features for person id- 007
     Saving features for person id- 008
     Saving features for person id- 009
     Saving features for person id- 010
     Saving features for person id- 011
     Saving features for person id- 012
```

Input 1: (Real signature)

Output:

(Displays result as Genuine image)

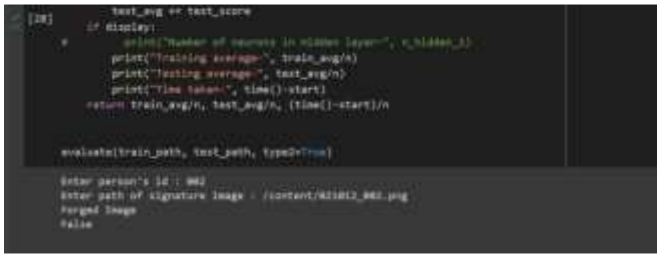


```
        print("Training average-", train_avg/n)
        print("Testing average-", test_avg/n)
        print("Time taken-", time()-start)
    return train_avg/n, test_avg/n, (time()-start)/n


evaluate(train_path, test_path, type2=True)

Enter person's id : 002
Enter path of signature image : /content/012012_002.png
Genuine Image
True
```

Input 2: (Forged signature)

Output

(Displays result as Forged Image)



```
[28]        test_avg += test_score
        if display:
#               print("Number of neurons in hidden layer-", n_hidden_1)
            print("Training average-", train_avg/n)
            print("Testing average-", test_avg/n)
            print("Time taken-", time()-start)
        return train_avg/n, test_avg/n, (time()-start)/n


    evaluate(train_path, test_path, type2=True)

    Enter person's id : 002
    Enter path of signature image : /content/021012_002.png
    Forged Image
    False
```

5. Conclusion

In this project we implemented offline verification of signatures by using different geometric measures. We developed an efficient system for signature verification and forgery detection based on image processing techniques using python libraries like TensorFlow. A solution is presented where the model is trained with a dataset of signatures, and predictions are made as to whether a provided signature is genuine or forged. As a result, we were able to distinguish between genuine and forged handwritten documents using pre-processing, feature extraction, and training the model with genuine and forged image datasets.

This project will help provide financial institutions the right solutions to detect forgeries quickly and accurately. Henceforth we hope this project is put into real world use and we are easily able to verify the authenticity of any documents thus minimizing the risk of fraud and providing to peace of mind to all parties involved.

REFERENCES

[1] S Jerome Gideon, Anurag Kandulna, Aron Abhishek Kujur, A Diana, Kumudha Raimond,Handwritten Signature Forgery Detection using Convolutional Neural Networks,Procedia Computer Science,Volume 143,2018,Pages 978-987,ISSN 1877-0509.

[2] Madasu Hanmandlu, Mohd. Hafizuddin Mohd. Yusof, Vamsi Krishna Madasu,Off-line signature verification and forgery detection using fuzzy modeling,Pattern Recognition,Volume 38, Issue 3,2005,Pages 341-356,ISSN 0031-3203,

[3] Zhang, KJ., Zhang, WW. & Li, D. Improving the security of arbitrated quantum signature against the forgery attack. Quantum Inf Process 12, 2655– 2669 (2013).

[4] S. Elliott and A. Hunt, "Dynamic signature forgery and signature strength perception assessment," in IEEE Aerospace and Electronic Systems Magazine, vol. 23, no. 6, pp. 13-18, June 2008, doi: 10.1109/MAES.2008.4558003.

[5] Journal of Forensic Sciences, Vol. , No. , 1993, pp. -, https://doi.org/. ISSN.

[6] L. E. Martinez, C. M. Travieso, J. B. Alonso and M. A. Ferrer, "Parameterization of a forgery handwritten signature verification system using SVM," 38th Annual 2004 International Carnahan Conference on Security Technology, 2004., 2004, pp. 193-196, doi: 10.1109/CCST.2004.1405391.

[7] Vamsi Krishna Madasu (Queensland University of Technology, Australia) and Brian C. Lovell (NICTA Limited (Queensland Laboratory) and University of Queensland, Australia.

[8] Hassaïne A., Al-Maadeed S. (2012) An Online Signature Verification System for Forgery and Disguise Detection. In: Huang T., Zeng Z., Li C., Leung C.S. (eds) Neural Information Processing. ICONIP 2012. Lecture Notes in Computer Science, vol 7666. Springer, Berlin, Heidelberg

[9] Dhvani Patel, Nehal Ghosalkar, Aruna Pavate, 2017, Offline Signature Verification using Artificial Neural Network, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) ICIATE – 2017 (Volume 5 – Issue 01)

[10] Chaurasia A., Agarwal H., Vishwakarma A., Dwivedi A., Sharma A. (2021) Signature Forgery Recognition Using CNN. In: Ranganathan G., Chen J., Rocha Á. (eds) Inventive Communication and Computational Technologies. Lecture Notes in Networks and Systems, vol 145. Springer, Singapore

AUTHOR INFORMATION

Rajveer Kalsi (20MIS1161), Student, School of Computer Science and Engineering, Vellore Institute of Technology - Chennai.

Kshitij Kiran Thakare (20MIS1136), Student, School of Computer Science and Engineering, Vellore Institute of Technology - Chennai.

Soni Singh (20MIS1137), Student, School of Computer Science and Engineering, Vellore Institute of Technology - Chennai.

Shashank Vinayak Burhade (20MIS1050), Student, School of Computer Science and Engineering, Vellore Institute of Technology - Chennai.