

A CENTER FOR INTER-DISCIPLINARY RESEARCH
2020-21

TITLE

“SIGNATURE VERIFICATION”



GOKARAJU RANGARAJU
INSTITUTE OF ENGINEERING AND TECHNOLOGY
AUTONOMOUS

Advanced Academic Center

(A Center For Inter-Disciplinary Research)

This is to certify that the project titled

“SIGNATURE VERIFICATION”

is a bonafide work carried out by the following students in partial fulfilment of the requirements for Advanced Academic Center intern, submitted to the chair, AAC during the academic year 2020-21.

NAME	ROLL NO.	BRANCH
MOHAMMED YASEEN ALI	19241A0532	CSE
DUDALA ESHWAR	19241A0478	ECE
BOBBERLA NEHA	19241A0468	ECE

NAME	ROLL NO	BRANCH
AKSHITHA THOTLA	19241A04H6	ECE
THOTA AAMUKTHA SAI SREE	19241A0452	ECE
SATYA ANVESH RANGANADHAM	19241A0548	CSE

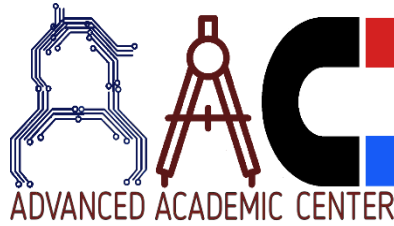
This work was not submitted or published earlier for any study

Dr/Ms./Mr.

Project Supervisor

Dr.B.R.K.Reddy
Program Coordinator

Dr.Ramamurthy Suri
Associate Dean,AAC



ACKNOWLEDGEMENTS

We express our deep sense of gratitude to our respected Director, Gokaraju Rangaraju Institute of Engineering and Technology, for the valuable guidance and for permitting us to carry out this project.

With immense pleasure, we extend our appreciation to our respected Principal, for permitting us to carry out this project.

We are thankful to the Associate Dean, Advanced Academic Centre, for providing us an appropriate environment required for the project completion.

We are grateful to our project supervisor who spared valuable time to influence us with their novel insights.

We are indebted to all the above mentioned people without whom we would not have concluded the project.

ABSTRACT

Signature verification is a common task in forensic document analysis. It is one of determining whether a questioned signature matches known signature samples. From the viewpoint of automating the task it can be viewed as one that involves machine learning from a population of signatures. There are two types of learning to be accomplished. In the first, the training set consists of genuine and forgeries from a general population. In the second there are genuine signatures in a given case. The two learning tasks are called person-independent (or general) learning and person-dependent (or special) learning. General learning is from a population of genuine and forged signatures of several individuals, where the differences between genuine and forgeries across all individuals are learnt. The general learning model allows a questioned signature to be compared to a single genuine signature. In special learning, a person's signature is learnt from multiple samples of only that person's signature— where within-person similarities are learnt. When a sufficient number of samples are available, special learning performs better than general learning (5% higher accuracy). With special learning, verification accuracy increases with the number of samples.

INTRODUCTION

Signature verification provides automated signature verification software that protects against fraud and reduces costs. Since all signatures are unique, they are relied upon for verification by financial organizations, businesses and governments to authorize transactions and documents. It verifies signatures on both print documents and online using mobile devices and terminals for account applications, check processing, loan origination, vote-by-mail, legal documents and much more. Accurate signature verification is crucial since forgery and fraud can cost organizations money, time and their reputation.

Forgery Protection

Identifies and protects against all types of forgeries including random (signature does not match the name of the authorized signee); blind (correct name but incorrect writing style) and even skilled forgeries (signature closely resembles the signature on file).

Print and Online

Automated signature verification works for signatures collected from scanned documents and those captured online on pads, tablets, smartphones and terminals.

Cost Effective

Perform signature verification on all checks, regardless of amount, without limitations due to cost; or on any other application regardless of volume. With high accuracy rates and adjustable thresholds, only questionable signatures are routed for human verification. This enables organizations to allow more time to analyze for potential fraud or to assess more signatures for verification, while they improve the overall accuracy of the verification process.

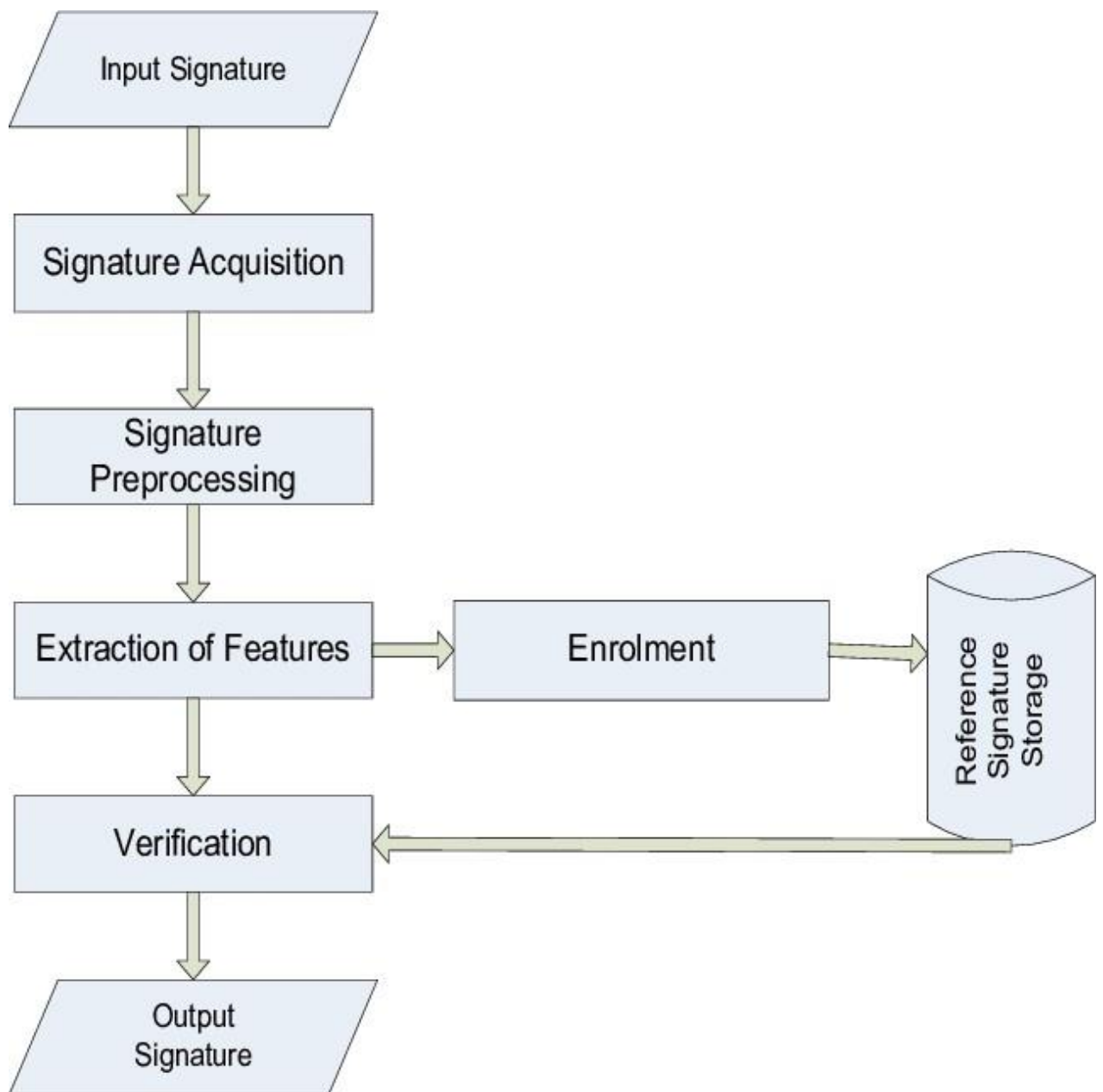
Fast, Accurate, Works 24/7

Human verification is limited by shortcomings that include level of expertise, fatigue, mood and working conditions. People can experience diminished accuracy over time, making more mistakes at the end of a day or shift than at the beginning. Automated signature verification can verify much faster, work 24/7 and produce consistent results at a much lower cost.

Highly Effective

Detects characteristics of a signature that are indistinguishable to the human eye for high fraud detection accuracy rates. Analyzes typical signature features such as comparison of geometric shapes, fragments, and trajectories. When used for online verification, it analyzes pressure, speed and tension. Multiple software engines work together to produce reliable verification output.

Work Flow and Methodology

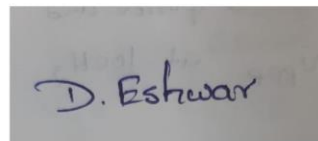


Methodology

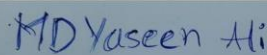
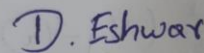
Gathering Information

Basically in this method we collect the scanned images of signature of different person .These images are stored in a database which we are going to use in training of our software, but not for actual matching. In our proposed work we have to use an interface with scanner for getting an image and storing it in desired database. The size of each image ($n \times n$) and each image can represent as array.

Real

A handwritten signature in blue ink that reads "MD Yaseen Ali".A handwritten signature in blue ink that reads "D. Eshwar".

Forged

A handwritten signature in blue ink that reads "MD Yaseen Ali", appearing as a copy of the real signature.A handwritten signature in blue ink that reads "D. Eshwar", appearing as a copy of the real signature.

Path Define

Here we define the path for forged and real images

Pre processing the image

Image preprocessing are the steps taken to format images before they are used by model training and inference. This includes, but is not limited to, resizing, orientation and color corrections.

Feature extraction and saving features

Feature extraction is **a part of the dimensionality reduction process**, in which, an initial set of the raw data is divided and reduced to more manageable groups. So when you want to process it will be easier. The most important characteristic of these large data sets is that they have a large number of variables. Then saving the features and their paths is called feature saving.

TF Model

In machine learning, a model is a **function with learnable parameters** that maps an input to an output. The optimal parameters are obtained by training the model on data. A well-trained model will provide an accurate mapping from the input to the desired output. In TensorFlow.

Code:

```
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
import pandas as pd
import numpy as np
from time import time

import keras
from keras.utils import np_utils
from tensorflow.python.framework import ops
import tensorflow.compat.v1 as tf

tf.disable_v2_behavior()

Original_Signatures = "F:\\Aac dataset\\Dataset\\real/"
Fake_Signatures = "F:\\Aac dataset\\Dataset\\forged/"

def ColorToGray(img):
    Grayimage = np.zeros((img.shape[0], img.shape[1]))
    for i in range(len(img)):
        for j in range(len(img[i])):
            Grayimage[i][j] = np.average(img[i][j])
    return Grayimage
```

```
def GrayToBinary(img):
```

```
    blur_radius = 0.8
```

```
    img = ndimage.gaussian_filter(img, blur_radius)
```

```
    ThresholdFreq = threshold_otsu(img)
```

```
    BinaryImg = img > ThresholdFreq
```

```
    BinaryImg = np.logical_not(BinaryImg)
```

```
    return BinaryImg
```

```
def PreProccesing(path, img=None, display=True):
```

```
    if img is None:
```

```
        img = mpimg.imread(path)
```

```
    if display:
```

```
        plt.imshow(img)
```

```
        plt.show()
```

```
    Grey = ColorToGray(img)
```

```
    if display:
```

```
        plt.imshow(Grey, cmap = matplotlib.cm.Greys_r)
```

```
        plt.show()
```

```
    BinaryImg = GrayToBinary(Grey)
```

```
    if display:
```

```
        plt.imshow(BinaryImg, cmap = matplotlib.cm.Greys_r)
```

```
        plt.show()
```

```
    r, c = np.where(BinaryImg==1)
```

```
    SignatureImage = BinaryImg[r.min(): r.max(), c.min(): c.max()]
```

```
    if display:
```

```
        plt.imshow(SignatureImage, cmap = matplotlib.cm.Greys_r)
```

```
        plt.show()
```

```
    return SignatureImage
```

Feature Extraction:

```
def Calculating_Ratio(img):
```

```
    pixels = 0
```

```
    for i in range(len(img)):
```

```
        for j in range(len(img[0])):
```

```
            if img[i][j]==True:
```

```
                pixels += 1
```

```
    total_pixels = img.shape[0] * img.shape[1]
```

```
    return pixels/total_pixels
```

```
def Calculating_EccentricityAndSolidity(img):
```

```
    region = regionprops(img.astype("int8"))
```

```
    return region[0].eccentricity, region[0].solidity
```

```
def Calculating_SkewKurtosis(img):
```

```
    h,w = img.shape
```

```
    x = range(w)
```

```
    y = range(h)
```

```
    xp = np.sum(img,axis=0)
```

```
    yp = np.sum(img,axis=1)
```

```
    cx = np.sum(x*xp)/np.sum(xp)
```

```
    cy = np.sum(y*yp)/np.sum(yp)
```

```
    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))
```

```
    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)
```

```
x4 = (x-cx)**4
```

```
y4 = (y-cy)**4
```

```
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)
```

```
def Calculating_Centroid(img):
```

```
    NumberOfWhites = 0
```

```
    a = np.array([0,0])
```

```
    for i in range(len(img)):
```

```
        for j in range(len(img[0])):
```

```
            if img[i][j]==True:
```

```
                b = np.array([i,j])
```

```
                a = np.add(a,b)
```

```
                NumberOfWhites += 1
```

```
    rowcols = np.array([img.shape[0], img.shape[1]])
```

```
    centroid = a/NumberOfWhites
```

```
    centroid = centroid/rowcols
```

```
    return centroid[0], centroid[1]
```

```
def Feature_Extraction(path, img=None, display=False):
```

```
    if img is None:
```

```
        img = mpimg.imread(path)
```

```
    img = PreProccesing(path, display=display)
```

```
    ratio = Calculating_Ratio(img)
```

```
    centroid = Calculating_Centroid(img)
```

```
    eccentricity, solidity = Calculating_EccentricityAndSolidity(img)
```

```
    skewness, kurtosis = Calculating_SkewKurtosis(img)
```

```
    feature_tuple = (ratio, centroid, eccentricity, solidity, skewness, kurtosis)
```

```
    features = (feature_tuple[0], feature_tuple[1][0], feature_tuple[1][1], feature_tuple[2], feature_tuple[3], feature_tuple[4][0], feature_tuple[4][1], feature_tuple[5][0], feature_tuple[5][1])
```

```
    return features
```

Features Manipulation :

```
def makeCSV():
    if not(os.path.exists('F:\\Aac dataset\\Dataset\\Features')):
        os.mkdir('F:\\Aac dataset\\Dataset\\Features')
        print('New folder "Features" created')
    if not(os.path.exists('F:\\Aac dataset\\Dataset\\Features\\Training')):
        os.mkdir('F:\\Aac dataset\\Dataset\\Features\\Training')
        print('New folder "Features/Training" created')
    if not(os.path.exists('F:\\Aac dataset\\Dataset\\Features\\Testing')):
        os.mkdir('F:\\Aac dataset\\Dataset\\Features\\Testing')
        print('New folder "Features/Testing" created')
    for person in range(1,13):
        per = ('00'+str(person))[-3:]
        print('Saving features for person id-',per)

        with open('F:\\Aac dataset\\Dataset\\Features\\Training\\training_'+per+'.c
sv', 'w') as handle:
            handle.write('ratio,cent_y,cent_x,eccentricity,solidity,skew_x,skew_y,k
urt_x,kurt_y,output\n')
            for i in range(0,3):
                source = os.path.join(Original_Signatures, per+per+'_00'+str(i)+'.png
')
                features = Feature_Extraction(path=source)
                handle.write(','.join(map(str, features))+',1\n')
            for i in range(0,3):
                source = os.path.join(Fake_Signatures, '021'+per+'_00'+str(i)+'.png')
                features = Feature_Extraction(path=source)
                handle.write(','.join(map(str, features))+',0\n')
```

```

with open('F:\\Aac dataset\\Dataset\\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')
    for i in range(3, 5):
        source = os.path.join(Original_Signatures, per+per+'_00'+str(i)+'.png')
        features = Feature_Extraction(path=source)
        handle.write(','.join(map(str, features))+',1\n')
    for i in range(3,5):
        source = os.path.join(Fake_Signatures, '021'+per+'_00'+str(i)+'.png')
        features = Feature_Extraction(path=source)
        handle.write(','.join(map(str, features))+',0\n')

makeCSV()

```

Tensor Flow Model :

```

def WritingFeatures(path):
    ExtractedFeatures = Feature_Extraction(path)
    if not(os.path.exists('F:\\Aac dataset\\Dataset\\TestFeatures')):
        os.mkdir('F:\\Aac dataset\\Dataset\\TestFeatures')
    with open('F:\\Aac dataset\\Dataset\\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, ExtractedFeatures))+'\n')

```



```
n_input = 9
```

```
train_person_id = input("Enter person's id : ")
```

```
test_image_path = input("Enter path of signature image : ")
```

```
train_path = 'F:\\Aac dataset\\Dataset\\Features\\Training\\training_'+train_person_id+'.csv'
```

```
WritingFeatures(test_image_path)
```

```
test_path = 'F:\\Aac dataset\\Dataset\\TestFeatures\\testcsv.csv'
```

```
def readCSV(train_path, test_path, type2=False):
```

```
    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)
```

```
    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.np_utils.to_categorical(correct,2)
```

```
    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]
```

```
        correct = np.array(temp)
```

```
        corr_test = keras.utils.np_utils.to_categorical(correct,2)
```

```
    if not(type2):
```

```
        return train_input, corr_train, test_input, corr_test
```

```
    else:
```

```
        return train_input, corr_train, test_input
```

```
ops.reset_default_graph()
```

```
learning_rate = 0.001
```

```
training_epochs = 1000
```

```
display_step = 1
```

```
n_hidden_1 = 7
```

```
n_hidden_2 = 10
```

```
n_hidden_3 = 30
```

```
n_classes = 2
```

```
X = tf.compat.v1.placeholder(dtype = tf.float32 ,shape= [None, n_input])
```

```
Y = tf.compat.v1.placeholder(dtype = tf.float32 ,shape= [None, n_classes])
```

```
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))
```

```
}
```

```
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
```

```
logits = multilayer_perceptron(X)
```

```
loss_op = tf.reduce_mean(tf.squared_difference(logits, Y))
```

```
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
```

```
train_op = optimizer.minimize(loss_op)
```

```
pred = tf.nn.softmax(logits)
```

```
correct_prediction = tf.equal(tf.argmax(pred,1), tf.argmax(Y,1))
```

```
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

```
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)
```

```
for epoch in range(training_epochs):
```

```
    _, cost = sess.run([train_op, loss_op], feed_dict={X: train_input, Y:corr_train})
```

```
        if cost<0.0001:
```

```
            break
```

```
    accuracy1 = accuracy.eval({X: train_input, Y: corr_train})
```

```
    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')
```

```
            return False
```

```
evaluate(train_path, test_path, type2=True)
```

Output:

Enter person's id : 012

Enter path of signature image : F:\\Aac dataset\\Dataset\\forged\\021012_000.png

Forged Image

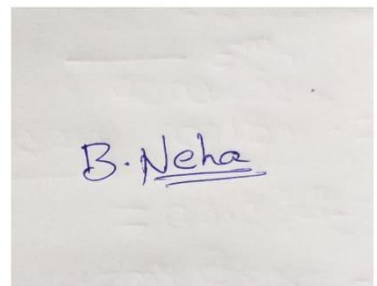
False

Real

Akshitha Aankthe B. Neha

Forged

Akshitha Aankthe



Future Developments

1. Development and scientific progress in biometric security applications based on handwritten text (capital letters, cursive letters, etc.) , where the goal is not the classical optical character recognition (OCR). The goal is to identify the author of the text in two different modes: text dependent and text independent. Additional functionality is checking if the author of the text is forced to write a specific text or if he is free to write whatever he wants.

2. While signature-based recognition systems are quite mature, the complementary information of the combination between signature and text provided by the same author as well as the possibility of crossed recognition has not been deeply analyzed yet.

3. Development of new algorithms able to identify the author of a drawing in a similar way to what is done with signatures or handwriting. In the same way that a specific simple pattern can be used to unlock the smartphone , a specific drawing (invented or copied) could hold the same functionality. Pattern unlocks consist of a grid of dots on a device's lock screen which users connect uniquely to gain access to the phone. However, the possibilities are much reduced when compared with handwritten drawings.

4. Development and scientific progress in biometric health applications based on handwritten text and drawings . This is a promising research line to detect healthy aging, evaluate the effect of prescribed drugs on a specific disease (such as apomorphine on Parkinson's disease), etc.

5. Analysis of the potential use of handwritten signatures for diagnosing diseases. Although the signature tends to be a mechanical movement with almost no cognitive effort to perform it, it has potential use in health applications.

6. Stress detection given handwritten tasks has potential implications on security applications, e.g., coercion detection .

7. Drug substance abuse can have legal implications for professional activities. It would be cheap and non-invasive to conduct preliminary tests based on handwriting tasks.

8. Development of algorithms to anonymize handwritten samples in order to hide the identity of the user while preserving the diagnostic capability of the samples .

9. Development of algorithms to anonymize handwritten samples in order to hide the health information of the user while preserving the identification capability of the samples

References :

Google;

You tube;

Research Paper;

[286027506.pdf \(core.ac.uk\);](#)

<https://www.sciencedirect.com/science/article/pii/S1877050918320301>