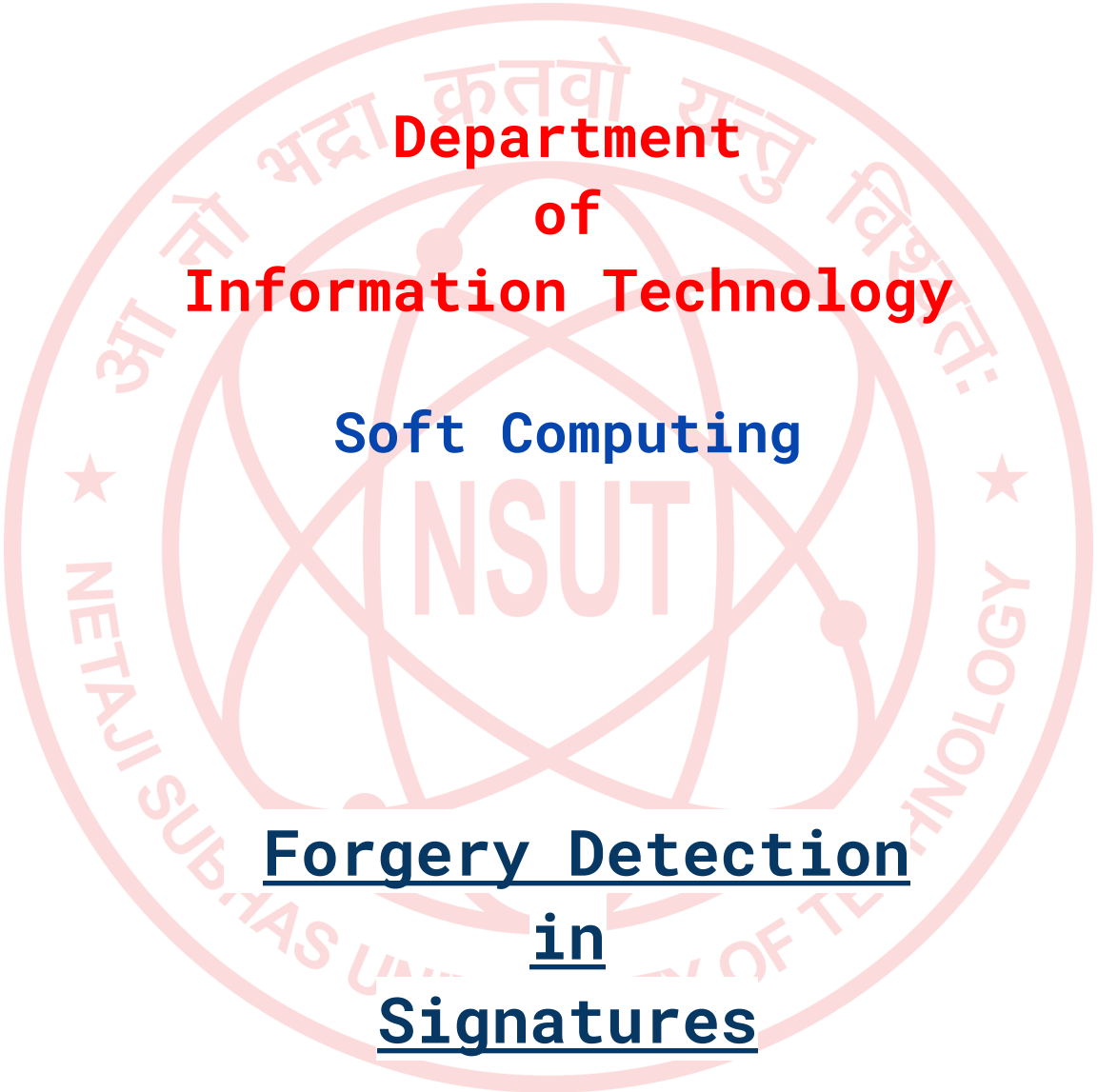
NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY
NEW DELHI



Department
of
Information Technology

Soft Computing

Forgery Detection
in
Signatures

Name: Manoj Kumar
Roll No: 2019UIT3028

INDEX

ACKNOWLEDGEMENT

I would like to present my special gratitude towards my mentor **Dr. Akshi Kumar** ma'am for giving me this golden opportunity to work on this project on the topic "**Forgery Detection in Signatures using Convolutional Neural Networks"** which was a very good learning experience . I am grateful for her help and guidance throughout the project.

Secondly I would also like to thank my teammates who helped me a lot in finalizing this project and finishing it within the limited time frame.

INTRODUCTION.

A **signature** is a handwritten (and often stylized) depiction of someone's name, nickname, or even a simple "X" or other mark that a person writes on documents as a proof of identity and intent.

Signature forgery refers to the act of falsely replicating another person’s signature.

Automatic signature forgery detection is a challenging task that has niche yet significant potential due to its applications in document verification, legal documents, financial cheques, originality of famous arts etc.

The painting is considered a replica of Picasso’s art as the signature was found to be forged. The original painting costs millions of dollars.

However, the problem of signature forgery detection for a group of people has been less extensively studied,due to the difficulty in well labeled and organized data. Usually,even genuine signatures have slight variations.

A number of characteristics can suggest to an examiner that a signature has been forged, mostly stemming from the forgery focusing on accuracy rather than fluency. These include:

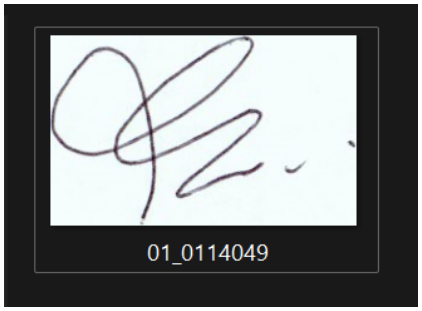
- Shaky handwriting
- Pen lifts
- Signs of retouching
- Letter proportions
- Very close similarity between two or more signatures

Signature Forgery Detection is done by highly skilled experts, and hourly fees can range from $400-$600 per hour of work. Since, there’s such a high cost and lack of experts, automation of this task can solve a lot of problems.

The focus is on differentiating between genuine differences and genuine-forgery differences. The learning problem is stated as learning a binary classification problem where the input consists of the difference between a pair of signatures. The verification task is performed by comparing the questioned signature against each known signature.

Images of a known Signature (Original)

Now, Our model will be questioned about a signature which we don't know if it's forged or original.



Questioned Signature

Now, Based on the knowledge acquired during learning, the model will predict 1 ("Forged") or 0 ("Original").

In brief, we made a model to predict whether a signature is Forged or Not.

METHODOLOGY

a. **Objective**

To design a CNN model that can classify whether a signature is forged or not.

b. **Dataset Used :**

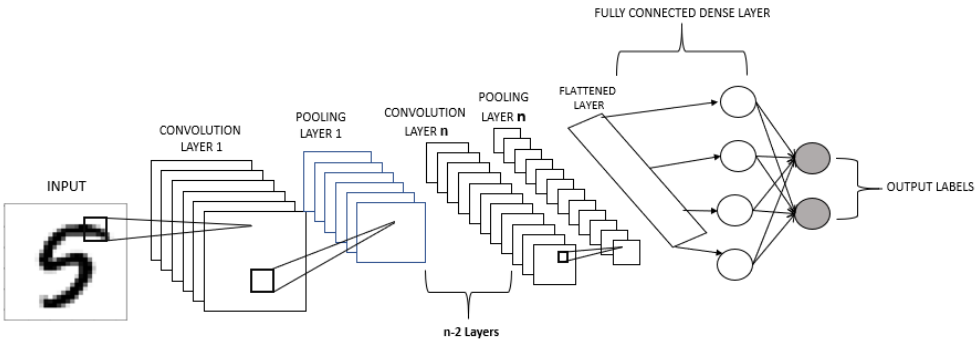
All the data are extracted from ICDAR 2011 Signature Dataset (Dutch) and organized perfectly for user usage. In the dataset the directory number says the name of the user and it is classified into two : Genuine with its own user number and Fraud with the user number + "_forg".

c. **Proposed Solution :**

To tackle this problem of signature forgery, the method that we suggest is the use of a **Convolutional Neural Network Model.**

A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable

weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. A convolutional neural network has convolution layers followed by a fully connected neural network.



The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.
The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such

fields overlap to cover the entire visual area.

d. **Approach**

In this work, the signature images are pre-processed in a batch-by-batch manner.

After these signatures are preprocessed, it is stored in a file directory structure which the keras python library can work with.

To build a powerful image classifier using very little training data, image augmentation is usually required to boost the performance of deep networks. Image augmentation artificially creates training images through different ways of processing or combination of multiple processing, such as random rotation, shifts, shear and flips, etc.

Due to it’s mentioned advantages, the project makes use of Image augmentation.
An augmented image generator is created using ImageDataGenerator API in Keras.

ImageDataGenerator generates batches of image data with real-time data augmentation.

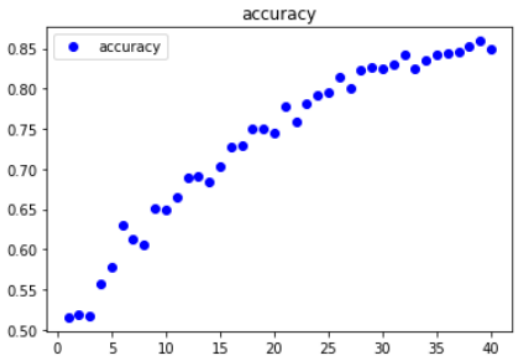
Then the CNN has been implemented in python using the Keras with the TensorFlow backend to learn the patterns associated with the signatures.

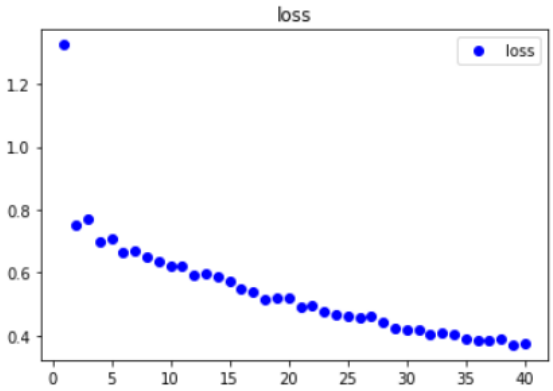
Then the model derived has been verified using accuracy and loss metrics to see how well the model has fit the data.

Finally, the model has been tested by using a signature from a holdout set to see if the predictions are correct.

RESULTS

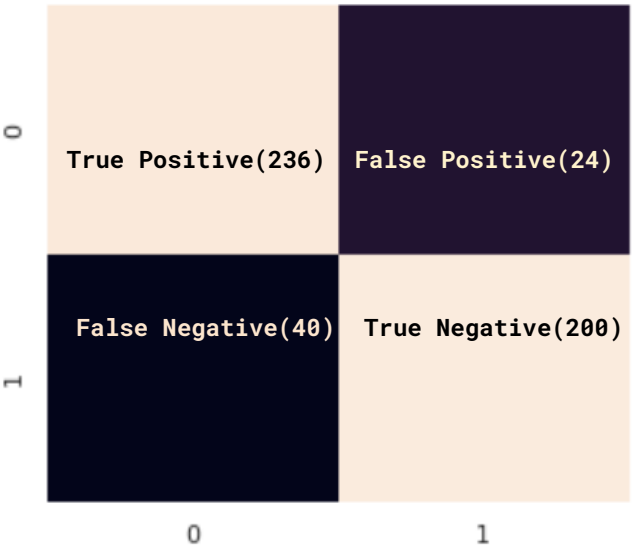
The model has successfully managed to predict whether a signature is forged or not with an accuracy of **87.26%**.
In order to avoid overfitting, some random factors were introduced in training samples like rotating images, shifting images in left-right, up-down to make the learning examples as close to real world variations. Hence, We have tried to model robust enough to handle real world data with a little trade-off with accuracy numbers.
The Model was trained for 40 Epochs, throughout which, the accuracy of the model kept increasing. The graph of accuracy vs Epoch number is as follows:





Even though the model was able to achieve an accuracy of **87.26%**, it cannot be said that this alone can replace the manual signature forgery detection. A study was conducted by the German

Research Centre for Artificial Intelligence(DFKI),titled “Man vs. Machine: A Comparative Analysis for Forensic Signature Verification". A detailed analysis we performed in order to compare the performance of human experts/FHEs against automated systems with respect to signature verification. The analysis was concluded as: “Both the human experts as well as automated systems/machines encountered difficulties incorrectly classifying disguised signatures. This is probably because of limited disguised training data availability. However,the results provided are encouraging and we hope that automated systems will become better with time and with access to more forensically relevant data especially involving disguised behaviors.”



Confusion matrix

Hence, the model’s accuracy levels closely rivals the accuracy of humans, but before it can replace the manual signature forgery detection it definitely has to cover a long way both in terms of training data and reliability.

CONCLUSION

The system recognizes and identifies whether the signature is forged or not with acceptable accuracy.

The popularity pattern is trained on Convolutional Neural Networks (CNN) that works well with the dataset of pictures storing the extracted features. Negligible error or misclassification is required in such sensitive applications.

Experimental results show that the stated system **achieves 87.26%** accuracy on the validation set.

Overall, I think this project has been a success, as it was a reasonable attempt at a real-world problem, i.e. forgery detection in signature. Even though it wasn't quite able to achieve state of the art accuracy for the given task, the project manages to make a meaningful contribution to the ever growing field of forgery detection.

Future scope

A robust and reliable signature recognition and verification system with maximum accuracy possible is very important for many purposes like enforcement, security management, and lots of business processes.

It can be used as an intermediate tool to authenticate several documents like cheques, legal records, certificates, etc and in the process might replace actual human presence in such places given the model is trained over a wide dataset effectively. Similar methodology and approach can be used to tackle similar problems like painting forgery, handwriting recognition, etc.
More efficient models can be built, taking this project as a base and building upon it.

APPENDIX

CODE

```
from keras.preprocessing.image import ImageDataGenerator
import keras
import os
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
```

```
!mkdir Train
!mkdir Test

!mkdir Train/Fake
!mkdir Train/Real

!mkdir Test/Fake
!mkdir Test/Real
```

```
# Repositioning the train data so that keras can read it.

PATH = "/kaggle/input/signature-verification-dataset/sign_data/train/"

for i in os.listdir(PATH):

    contol = i.split("_")

    try:

        if contol[1]=="forg":

            os.system("cp -r {} Train/Fake".format(PATH+i))

    except:

        os.system("cp -r {} Train/Real".format(PATH+i))
```

```
# Rearrange the test data in such a way that keras can read it.

PATH = "/kaggle/input/signature-verification-dataset/sign_data/test/"

for i in os.listdir(PATH):

    contol = i.split("_")

    try:

        if contol[1]=="forg":

            os.system("cp -r {} Test/Fake".format(PATH+i))

    except:

        os.system("cp -r {} Test/Real".format(PATH+i))
```

```
# locations

train_dir = os.path.join("/kaggle/working/Train")

test_dir = os.path.join("/kaggle/working/Test")
```

```
train_datagen = ImageDataGenerator(

      # rescaling pixels between 0,1
      rescale=1./255,

      # Angle of random rotation of images in degrees (0-180)
      rotation_range=40,

      # horizontal and vertical scrolling ratios of images
      width_shift_range=0.2,

      # horizontal and vertical scrolling ratios of images
      height_shift_range=0.2,

      # sprain operation
      shear_range=0.2,

      # zoom operation
      zoom_range=0.2,

      # rotate image vertically
      horizontal_flip=True,

      # excess after processing
      # determines how the image points are filled
      fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)
```

```
train_generator = train_datagen.flow_from_directory(

        # target directory
        train_dir,

        # all images will be resized as (150x150)
        target_size=(200, 200),

        # batch or stack size
        batch_size=64,

        # binary tags required
        # because we are using binary_crossentropy
        class_mode='binary')
```

```
test_generator = test_datagen.flow_from_directory(

        test_dir,

        target_size=(200, 200),

        batch_size=64,

        class_mode='binary')
```

```
plt.imshow(train_generator[0][0][5])
print("Label : ",train_generator[0][1][5])
```

```
plt.imshow(train_generator[0][0][60])
print("Label : ",train_generator[0][1][60])
```

```
class MyModel(tf.keras.Model):

  def __init__(self):
    super(MyModel, self).__init__()

    self.cnn1 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu', input_shape=(3,200,200))
    self.cnn2 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu')
    self.cnn3 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu')

    self.flatten = tf.keras.layers.Flatten()

    self.dense1 = tf.keras.layers.Dense(512, activation='relu')
    self.dense2 = tf.keras.layers.Dense(1, activation='sigmoid')

  def call(self, inputs):

    x = self.cnn1(inputs)
    x = self.cnn2(x)
    x = self.cnn3(x)
    x = self.flatten(x)
    x = self.dense1(x)
    x = self.dense2(x)

    return x

model = MyModel()
```

```
input_shape = (None, 200, 200, 3)
model.build(input_shape)
model.summary()
```

```
model.compile(
    # loss function
    loss="binary_crossentropy",

    # Optimization:
    # Considering the loss created by the data, which is the input of our network
    # self-update mechanism
    optimizer=tf.keras.optimizers.RMSprop(lr=2e-5),

    # metrics to follow during training and testing.
    metrics=["acc"])
```

```
# We will get the acc, loss, val_acc, val_loss values from the variable named history.
history = model.fit_generator(

    # training data
    train_generator,

    # the number of samples it will run through until the loop finishes (stack to get)
    steps_per_epoch=train_generator.samples//train_generator.batch_size,

    # number of cycles
    epochs= 40,

    verbose=2)
```

```
# Training achievement score
acc = history.history["acc"]

# training loss score
loss = history.history["loss"]

# We will plot graphs according to the number of epochs.
epochs = range(1, len(acc) + 1)

# We had training data drawn for itself..
plt.plot(epochs, acc, "bo", label="Training achievement")

# the title of graph
plt.title("Training achievement")

plt.legend()

plt.figure()

# we had training data drawn for itself.
plt.plot(epochs, loss, "bo", label="Training loss")


# the title of our graph
plt.title("Training loss")

plt.legend()

# display on screen
plt.show()
```

```
# Loss and verification with test data
model.evaluate(test_generator)
```

```
plt.imshow(test_generator[0][0][5])
print("Label : ",test_generator[0][1][5])

test_input = test_generator[0][0][5]
test_input = np.expand_dims(test_input,axis=0)
test_input = np.expand_dims(test_input,axis=0)
test_input = np.expand_dims(test_input,axis=0)

pred = model.predict(test_input)

if pred>=0.5:
    pred = 1

else:

    pred = 0

print("Predict : ",float(pred))
```

```
plt.imshow(test_generator[0][0][30])
print("Label : ",test_generator[0][1][30])

test_input = test_generator[0][0][30]
test_input = np.expand_dims(test_input,axis=0)
test_input = np.expand_dims(test_input,axis=0)
test_input = np.expand_dims(test_input,axis=0)

pred = model.predict(test_input)

if pred>=0.5:
    pred = 1

else:

    pred = 0

print("Predict : ",float(pred))
```

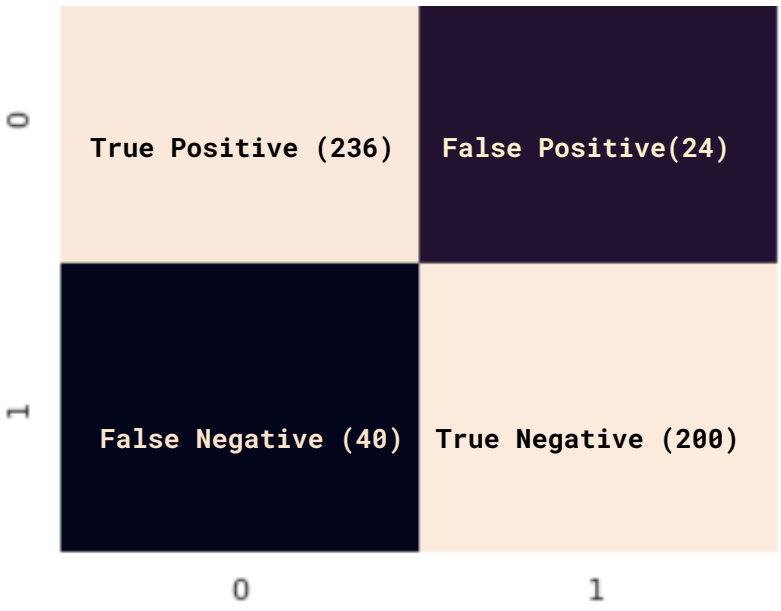
```
from sklearn.metrics import confusion_matrix
test_steps_per_epoch = np.math.ceil(test_generator.samples / test_generator.batch_size)
Y_pred = model.predict_generator(test_generator)
y_pred = np.argmax(Y_pred, axis=1)

print(Y_pred)
binaryYPred=[]
for i in range(0,len(Y_pred)):
    if Y_pred[i]>=0.5:
        binaryYPred.append(1.0)
    else:binaryYPred.append(0.0)
print(binaryYPred)
t=[]
for i in range(0,len(test_generator)):
    for j in range(0,len(test_generator[i][1])):
        t.append(test_generator[i][1][j])


print(len(t))
y_true = t

print(confusion_matrix(binaryYPred, y_true, labels=[1,0]))
```

```
print(confusion_matrix(binaryYPred, y_true, labels=[1,0]))
c_m=confusion_matrix(binaryYPred, y_true, labels=[1,0])
import seaborn as sns; sns.set_theme()
ax=sns.heatmap(c_m)
```



Confusion matrix

BIBLIOGRAPHY

- https://cedar.buffalo.edu/~srihari/papers/ICGVIP2006-sig.pdf
- https://www.sciencedirect.com/science/article/pii/S1877050918320301
- https://arxiv.org/pdf/1705.05787.pdf
- Principle Of Soft Computing 2nd Edition By S.N.Sivanandam And S.N.Deepa
- https://en.wikipedia.org/wiki/Convolutional_neural_network

***