Project Report

On

Signature Forgery Detection Using Siamese Network: One Shot Learning

Submitted in partial fulfilment for the award of

Post Graduate Diploma in Artificial Intelligence From C-DAC, ACTS (Pune)

Guided by:

Mrs. Swapna E.

Presented by:

PRN	NAME
200240128012	Kanumuri Sri Naga Sai Ajith
200240128015	Mohith Kumar
200240128018	Padmaja Phanse
200240128023	Satbhai Aniket Bharatrao
200240128033	V Raghavendra Krishna Srujan

Centre of Development of Advanced Computing (C-DAC), Pune.

INDEX

CERTIFICATE

This is to certify that the project titled "**Signature Forgery Detection Using Siamese Network: One Shot Learning"** submitted to the **CDAC ACTS, Pune** by **Kanumuri Sri Naga Sai Ajith, Mohith Kumar, Padmaja Phanse, Satbhai Aniket Bharatrao, V Raghavendra Krishna Srujan is** a bonafide record of the work done by the students towards partial fulfillment of requirements for the award of **Post Graduate Diploma in Artificial Intelligence.**

ACKNOWLEDGEMENT

Completion of this project and thesis would not have been possible without the help of many people, to whom we are very thankful. First of all, we would like to convey our sincere thanks to our supervisor Mrs. Swapna E for her constant support and encouragement. Her motivation, guidance and support helped us a great deal to achieve this feat.

We would like to thank our course coordinator Mrs. Priyanka Ranade for her continuous support and encouragement.

We would like to thank Mr. Mahesh Bhagwat sir for guiding us in project related work. We wish a deep sense of gratitude and heartfelt thanks to management for providing excellent lab facilities and tools.

INTRODUCTION

In the situations where signature verification is critical, such as bank cheque validation, an automated system that verifies the signatures will be reliable. An immediately obvious way to make such a system is by training a machine learning model to classify signatures where each person is a label. If we design such a system, we will not be able to classify the signatures of new persons. One-shot learning solves this problem.

In One-shot learning, we need to keep at least one genuine signature of the person in the database whose signature has to be verified. The model compares the input signature with the signature present in the database and determines whether the input is forged or genuine. So the model does not classify signatures based on a predetermined list of persons. We can easily add a new person to the system by adding a genuine signature of the same person to the database.

In this approach, the real signatures in the database and the input signature are passed through the siamese network to obtain comparable and representative vectors. These two vectors are compared using Euclidean distance to determine the genuineness of the input signature. If the Euclidean distance is more than a threshold, the input signature is classified to be a forged signature, else it is classified to be a real signature.

PURPOSE

Biometrics technology is used in a wide variety of security applications. The aim of such systems is to recognize a person based on physiological or behavioural traits. In the first case, the recognition is based on measurements of biological traits, such as the fingerprint, face, iris, etc. The later case is concerned with behavioural traits such as voice and the handwritten signature.

Biometric systems are mainly employed in two scenarios: verification and identification. In the first case, a user of the system claims an identity, and provides the biometric sample. The role of the verification system is to check if the user is indeed who he or she claims to be. In the identification case, a user provides a biometric sample, and the objective is to identify it among all users enrolled in the system.

The handwritten signature is a particularly important type of biometric trait, mainly due to its ubiquitous use to verify a person's identity in legal, financial and administrative areas. One of the reasons for its widespread use is that the process to collect handwritten signatures is non-invasive, and people are familiar with the use of signatures in their daily life.

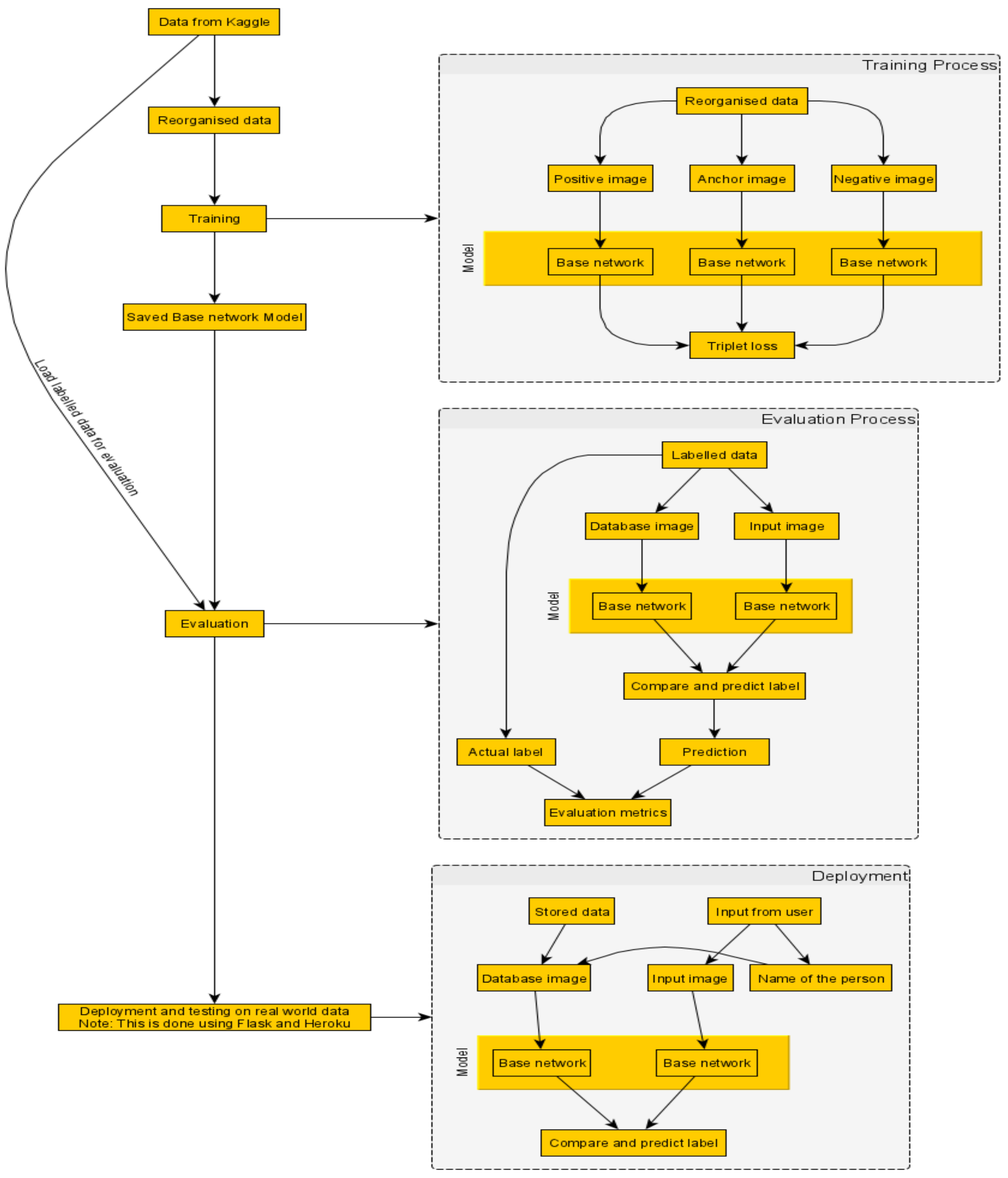
Signature verification systems aim to automatically discriminate if the biometric sample is indeed of a claimed individual. In other words, they are used to classify query signatures as genuine or forgeries. Deep Learning methods are applied for signatures. Such methods have demonstrated superior results in multiple benchmarks.

METHODOLOGY

Data obtained from Kaggle is organized into different directories where each directory represents a person. Each directory is, in turn, organized into two directories which contain genuine signatures and forged signatures respectively. This data was reorganized for training as described next. Each directory which contains genuine signatures was split into two directories each containing an equal number of images. During training, an image each from the three directories (genuine1, genuine2, forged) was loaded as a positive image, an anchor image, a negative image respectively. The siamese network was trained using the triplet loss function. The base network, through which each image is passed, was saved after training.

Data from Kaggle was used directly for evaluation as it was already organized suitably for evaluation. Pairs of images and their labels were obtained from CSV files included in Kaggle. The base network, which has been saved previously, was loaded and an evaluation model was constructed. That model was then evaluated using the true labels.

A web application was built using Flask and deployed using Heroku. Some images were stored beforehand to compare input images. The web application takes 2 inputs from the user, the signature image to be verified and the name of the person. A stored image is loaded based on the person name obtained from the user. Stored image and input image are then passed through the siamese network and compared to get a prediction. Finally, that prediction is shown to the user. The constructed web application is used to test the model using real-world data.


Data from Kaggle
Reorganised data
Training
Saved Base network Model
Load labelled data for evaluation
Evaluation
Deployment and testing on real world data
Note: This is done using Flask and Heroku
Training Process
Reorganised data
Positive image
Anchor image
Negative image
Model
Base network
Base network
Base network
Triplet loss
Evaluation Process
Labelled data
Database image
Input image
Model
Base network
Base network
Compare and predict label
Actual label
Prediction
Evaluation metrics
Deployment
Stored data
Input from user
Database image
Input image
Name of the person
Model
Base network
Base network
Compare and predict label

CODE

Image Preprocessing

Each original image in the train and test data were of size (350,900,3). We had to pass three such images in each iteration for training. Using original images would increase the computation complexity of the model. So, it's important to preprocess the image.

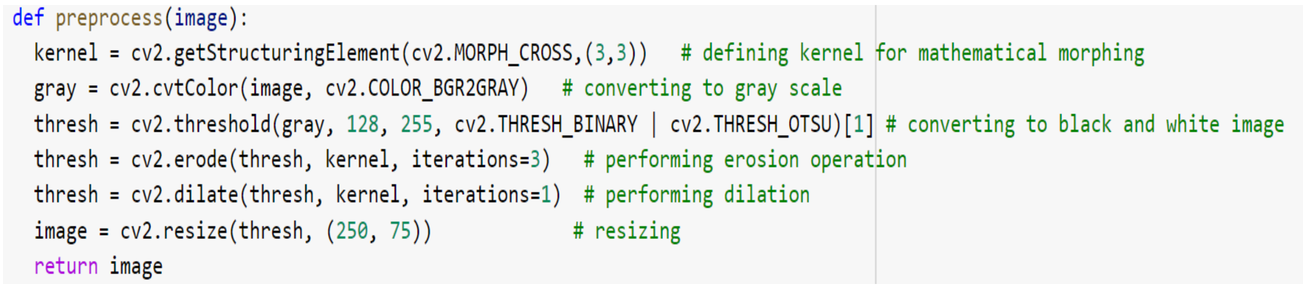
The preprocessing steps include:

1) Converting color image to gray scale image.

2) Thresholding on gray scale image to convert it to binary image.

3) Performing mathematical morphology operations erosion and dilation for a required number of iterations.

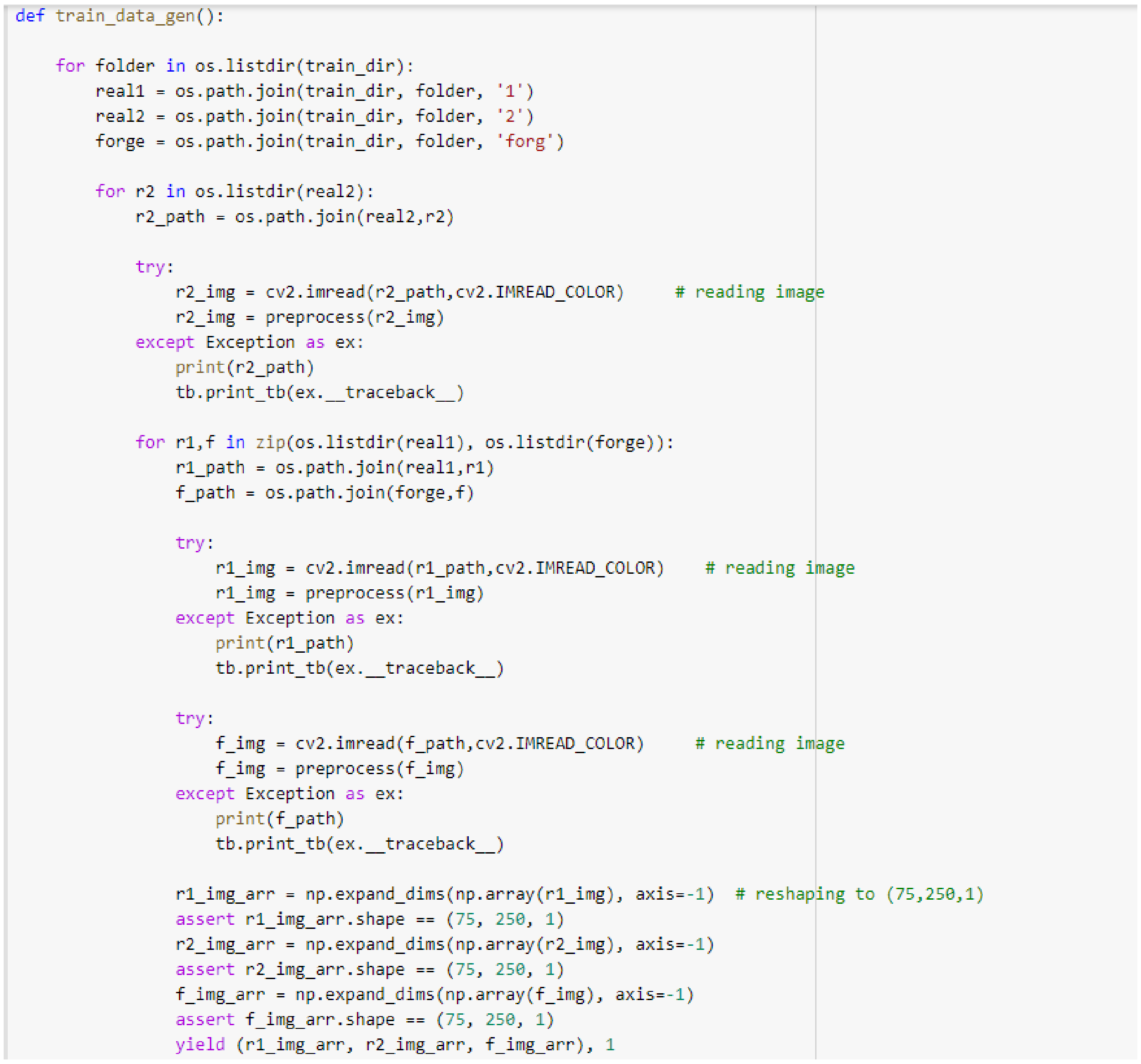
4) Resizing the image to (75, 250).



```
def preprocess(image):
  kernel = cv2.getStructuringElement(cv2.MORPH_CROSS,(3,3))   # defining kernel for mathematical morphing
  gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)   # converting to gray scale
  thresh = cv2.threshold(gray, 128, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1] # converting to black and white image
  thresh = cv2.erode(thresh, kernel, iterations=3)   # performing erosion operation
  thresh = cv2.dilate(thresh, kernel, iterations=1)  # performing dilation
  image = cv2.resize(thresh, (250, 75))             # resizing
  return image
```

Data Loading

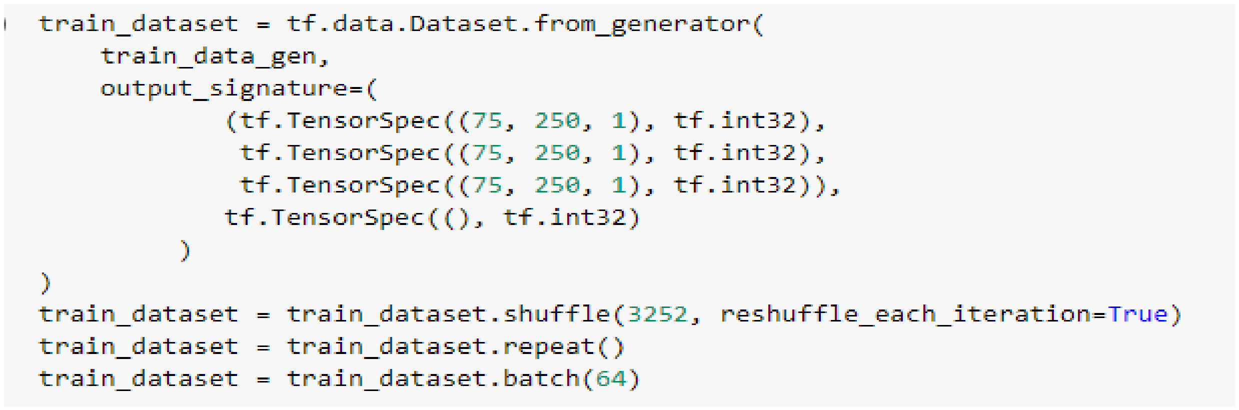
We have three sub folders in each person's folder. Real1, real2 and Forge. Each of these folders contain around 12-20 images. We need to pass three images per iteration to train the model. So, we took 1 image from the real2 folder and created all possible combinations with images in real1 and forge folders. This way we created around 3200 samples for training and 720 samples for testing.



```python
def train_data_gen():

    for folder in os.listdir(train_dir):
        real1 = os.path.join(train_dir, folder, '1')
        real2 = os.path.join(train_dir, folder, '2')
        forge = os.path.join(train_dir, folder, 'forg')

        for r2 in os.listdir(real2):
            r2_path = os.path.join(real2,r2)

            try:
                r2_img = cv2.imread(r2_path,cv2.IMREAD_COLOR)     # reading image
                r2_img = preprocess(r2_img)
            except Exception as ex:
                print(r2_path)
                tb.print_tb(ex.__traceback__)

            for r1,f in zip(os.listdir(real1), os.listdir(forge)):
                r1_path = os.path.join(real1,r1)
                f_path = os.path.join(forge,f)

                try:
                    r1_img = cv2.imread(r1_path,cv2.IMREAD_COLOR)    # reading image
                    r1_img = preprocess(r1_img)
                except Exception as ex:
                    print(r1_path)
                    tb.print_tb(ex.__traceback__)

                try:
                    f_img = cv2.imread(f_path,cv2.IMREAD_COLOR)     # reading image
                    f_img = preprocess(f_img)
                except Exception as ex:
                    print(f_path)
                    tb.print_tb(ex.__traceback__)

                r1_img_arr = np.expand_dims(np.array(r1_img), axis=-1)  # reshaping to (75,250,1)
                assert r1_img_arr.shape == (75, 250, 1)
                r2_img_arr = np.expand_dims(np.array(r2_img), axis=-1)
                assert r2_img_arr.shape == (75, 250, 1)
                f_img_arr = np.expand_dims(np.array(f_img), axis=-1)
                assert f_img_arr.shape == (75, 250, 1)
                yield (r1_img_arr, r2_img_arr, f_img_arr), 1
```

Creating dataset

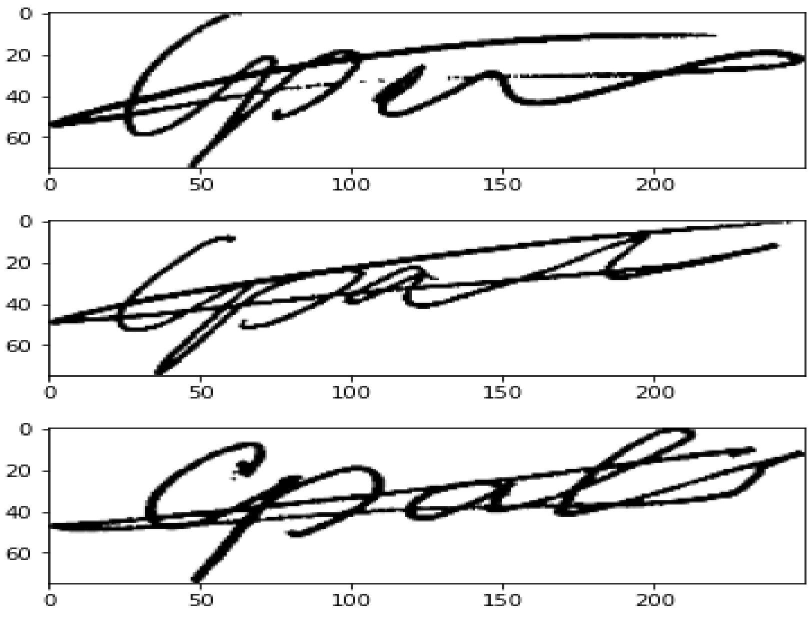
Since we are passing three images for each iteration, we have to create a Tensorflow dataset from the generator. For this we used a function from Tensorflow.



```
train_dataset = tf.data.Dataset.from_generator(
    train_data_gen,
    output_signature=(
            (tf.TensorSpec((75, 250, 1), tf.int32),
             tf.TensorSpec((75, 250, 1), tf.int32),
             tf.TensorSpec((75, 250, 1), tf.int32)),
            tf.TensorSpec((), tf.int32)
         )
)
train_dataset = train_dataset.shuffle(3252, reshuffle_each_iteration=True)
train_dataset = train_dataset.repeat()
train_dataset = train_dataset.batch(64)
```

Triplet Loss

In the below picture, the first image is the anchor image, the second image is a positive image and the third image is a forged image.



We can train the network by taking an anchor image and comparing it with both a positive sample and a negative sample. The dissimilarity between the anchor image and positive image must be low and the dissimilarity between the anchor image and the negative image must be high.

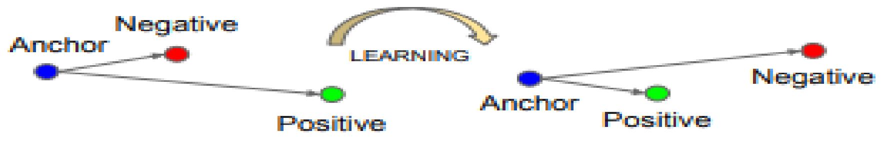
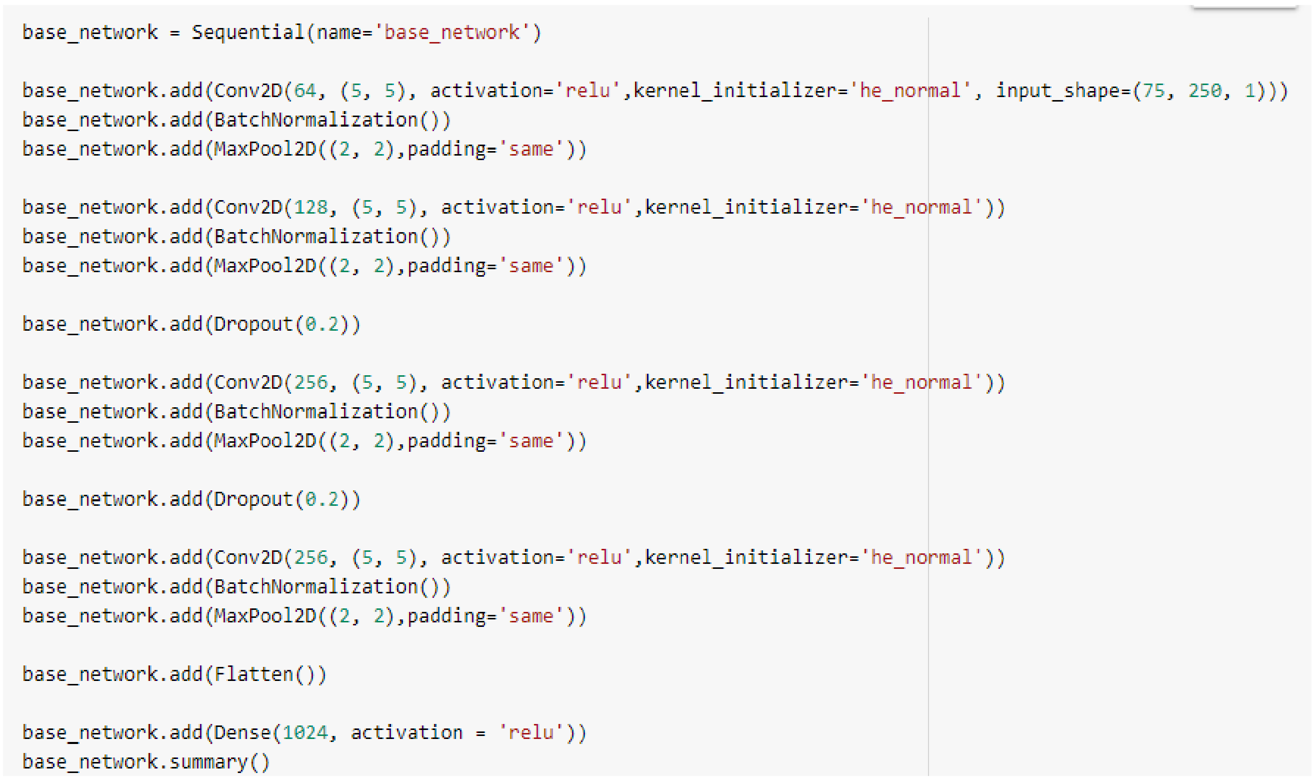


Figure 3. The **Triplet Loss** minimizes the distance between an *anchor* and a *positive*, both of which have the same identity, and maximizes the distance between the *anchor* and a *negative* of a different identity.

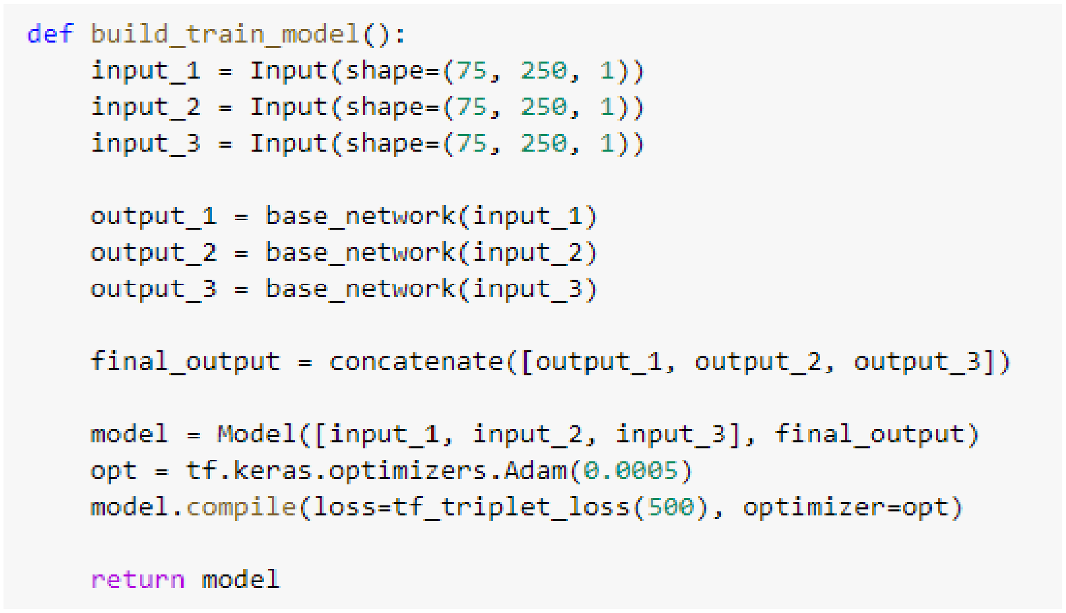
Code for triplet loss:

```
def tf_triplet_loss(alpha):
    def triplet_loss(y_true, y_pred):
        anchor, positive, negative = tf.split(y_pred, 3, axis=-1)
        pos_dist = tf.math.squared_difference(anchor, positive)
        neg_dist = tf.math.squared_difference(anchor, negative)
        loss = tf.maximum(pos_dist - neg_dist + alpha, tf.zeros_like(pos_dist))
        return loss
    return triplet_loss
```

Model Building



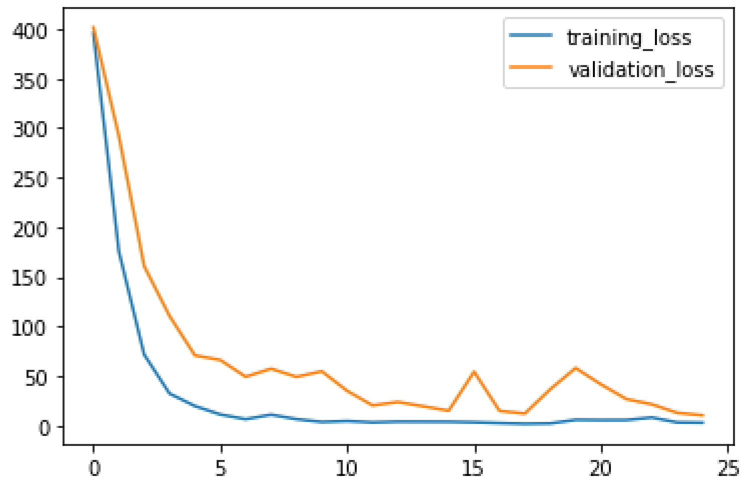
```
base_network = Sequential(name='base_network')

base_network.add(Conv2D(64, (5, 5), activation='relu',kernel_initializer='he_normal', input_shape=(75, 250, 1)))
base_network.add(BatchNormalization())
base_network.add(MaxPool2D((2, 2),padding='same'))

base_network.add(Conv2D(128, (5, 5), activation='relu',kernel_initializer='he_normal'))
base_network.add(BatchNormalization())
base_network.add(MaxPool2D((2, 2),padding='same'))

base_network.add(Dropout(0.2))

base_network.add(Conv2D(256, (5, 5), activation='relu',kernel_initializer='he_normal'))
base_network.add(BatchNormalization())
base_network.add(MaxPool2D((2, 2),padding='same'))

base_network.add(Dropout(0.2))

base_network.add(Conv2D(256, (5, 5), activation='relu',kernel_initializer='he_normal'))
base_network.add(BatchNormalization())
base_network.add(MaxPool2D((2, 2),padding='same'))

base_network.add(Flatten())

base_network.add(Dense(1024, activation = 'relu'))
base_network.summary()
```



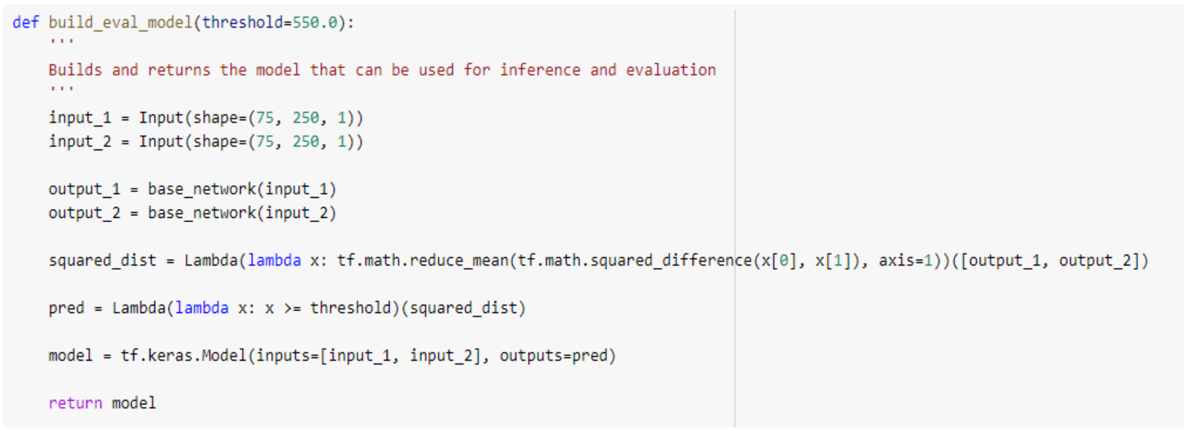
```python
def build_train_model():
    input_1 = Input(shape=(75, 250, 1))
    input_2 = Input(shape=(75, 250, 1))
    input_3 = Input(shape=(75, 250, 1))

    output_1 = base_network(input_1)
    output_2 = base_network(input_2)
    output_3 = base_network(input_3)

    final_output = concatenate([output_1, output_2, output_3])

    model = Model([input_1, input_2, input_3], final_output)
    opt = tf.keras.optimizers.Adam(0.0005)
    model.compile(loss=tf_triplet_loss(500), optimizer=opt)

    return model
```

Training Loss

After trying out many hyperparameters, optimal hyperparameters are batch size=64, optimizer= Adam, learning rate = 0.0005, epochs=25, triplet loss margin = 500



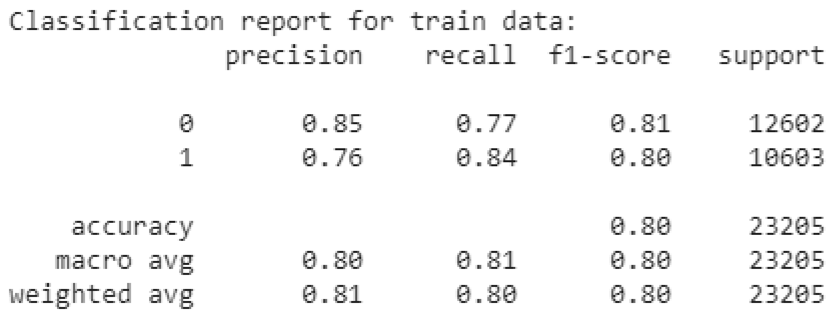
Model Evaluation



```
def build_eval_model(threshold=550.0):
    '''
    Builds and returns the model that can be used for inference and evaluation
    '''
    input_1 = Input(shape=(75, 250, 1))
    input_2 = Input(shape=(75, 250, 1))

    output_1 = base_network(input_1)
    output_2 = base_network(input_2)

    squared_dist = Lambda(lambda x: tf.math.reduce_mean(tf.math.squared_difference(x[0], x[1]), axis=1))([output_1, output_2])

    pred = Lambda(lambda x: x >= threshold)(squared_dist)

    model = tf.keras.Model(inputs=[input_1, input_2], outputs=pred)

    return model
```

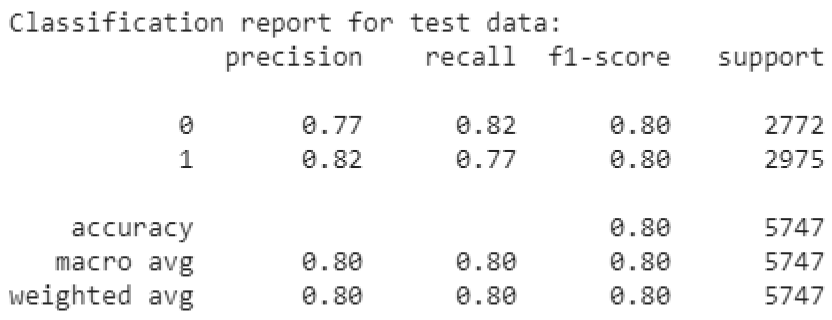
After the model is built, the model is saved in .h5 format. In a new environment, the saved model is loaded and is evaluated using training and testing data. Here we have coded a function to load the data. The data contains two images in each sample, one is a real image and the other could be a real/forged image along with a label. We are passing these two images at a time and we will get the similarity measure of both the images. If the measure is more than a specified threshold then the input image is classified as a forged signature else it is classified as a real signature.

Classification report of training data:



```
Classification report for train data:
              precision    recall  f1-score   support

           0       0.85      0.77      0.81     12602
           1       0.76      0.84      0.80     10603

    accuracy                           0.80     23205
   macro avg       0.80      0.81      0.80     23205
weighted avg       0.81      0.80      0.80     23205
```

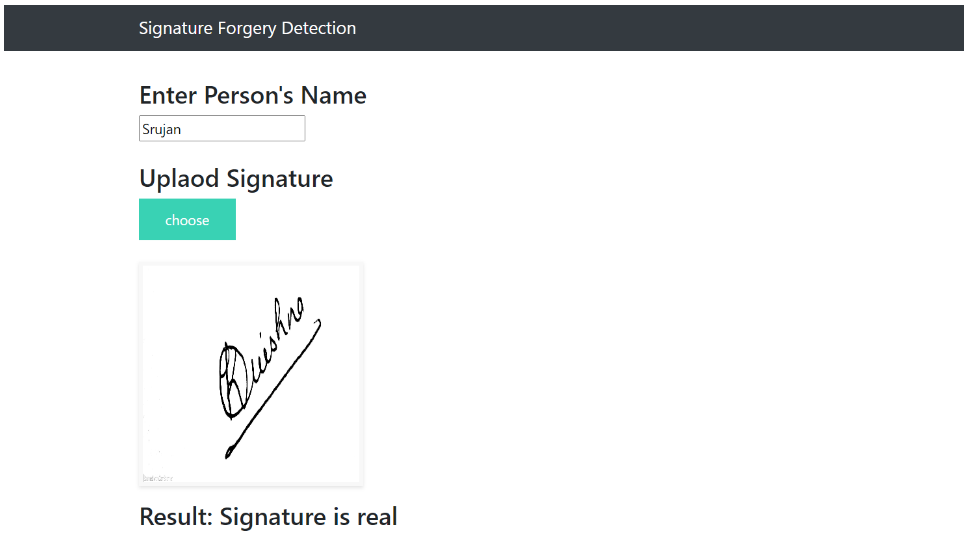
Classification report of testing data:



```
Classification report for test data:
              precision    recall  f1-score   support

           0       0.77      0.82      0.80      2772
           1       0.82      0.77      0.80      2975

    accuracy                           0.80      5747
   macro avg       0.80      0.80      0.80      5747
weighted avg       0.80      0.80      0.80      5747
```

Live Testing Using Web Application

We have tested it using two people's signatures which were not included in training the model. The model correctly classified the images as real and forged.

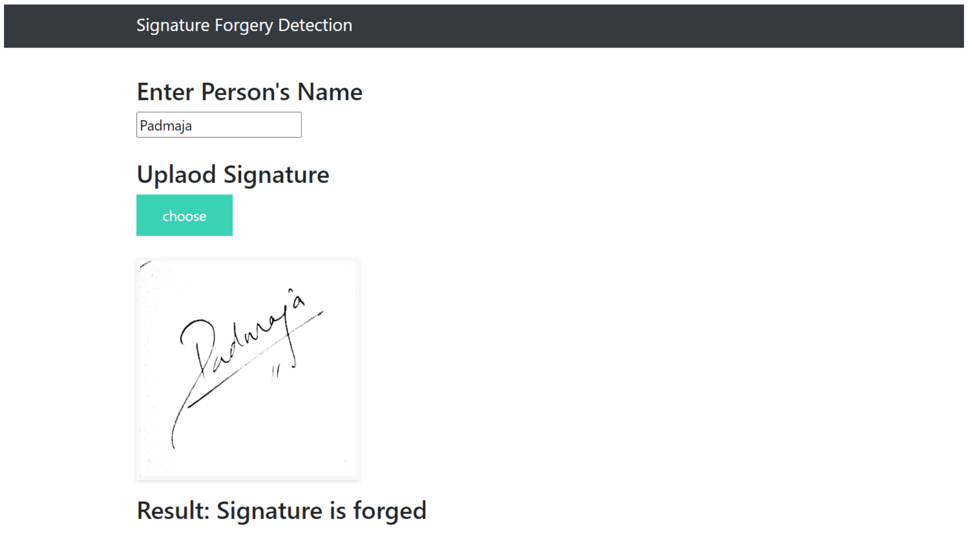

Signature Forgery Detection

Enter Person's Name

Srujan

Uplaod Signature

Result: Signature is real


Signature Forgery Detection

Enter Person's Name

Padmaja

Uplaod Signature

Result: Signature is forged

REFERENCES

- https://www.kaggle.com/kayademirs/signature
- Signature Verification using a "Siamese" Time Delay Neural Network - https://proceedings.neurips.cc/paper/1993/file/288cc0ff022877bd3df94bc9360b9c5d-Paper.pdf
- FaceNet: A Unified Embedding for Face Recognition and Clustering - https://arxiv.org/abs/1503.03832
- How to choose your loss when designing a Siamese Neural Net : Contrastive, Triplet or Quadruplet? - https://towardsdatascience.com/how-to-choose-your-loss-when-designing-a-siamese-neural-net-contrastive-triplet-or-quadruplet-ecba11944ec
- Siamese Network Keras for Image and Text similarity - https://medium.com/@prabhnoor0212/siamese-network-keras-31a3a8f37d04