# Configuration Manual

MSc Research Project
Programme Name

Anthony Effiok
x19108788

School of Computing
National College of Ireland

Supervisor: Dr. Hicham Rifai

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

| | |
|---|---|
| **Student Name:** | Anthony Eyo Effiok |
| **Student ID:** | X19108788 |
| **Programme:** | Msc Data Analytics **Year:** 2020 |
| **Module:** | Research Project |
| **Lecturer:** | Dr. Hicham Rifai |
| **Submission Due Date:** | $17^{TH}$ August 2020 |
| **Project Title:** | **NiohSign: A Siamese neural network approach for Signature Authentication** |
| **Word Count:** | 685 **Page Count:** 11 |

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

| | |
|---|---|
| **Signature:** | Anthony Eyo Effiok |
| **Date:** | $17^{TH}$ August 2020 |

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

| | |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies) | □ |
| **Attach a Moodle submission receipt of the online project submission,** to each project (including multiple copies). | □ |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | □ |

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Configuration Manual


## Anthony Effiok
x19108788


## 1 Introduction

This configuration manual aims to provide all information in regards to the system configuration, the programming language used and the complete program code used for the research in the project: "NiohSign: A Siamese neural network approach for Signature Authentication"

## 2 System Configuration

### 2.1 Hardware

- OS: Mac OS
- RAM: 8GB
- Processor: Inter core i5
- Hard Disk: 250GB

### 2.2 Software

- Jupyter Notebook - Python

## 3 Project Development

The data required for the project is a signature dataset known as BHSig260 [1]. This data set comprises of signatures appended by 260 individuals of which 100 are in Bengali and 160 are in Hindi. The signatures appended in Hindi are used to test the performance of the model created in this project.

Figure 1 shows the importation of required libraries for the execution of the project namely the keras libraries.

[1] https://goo.gl/9QfByd

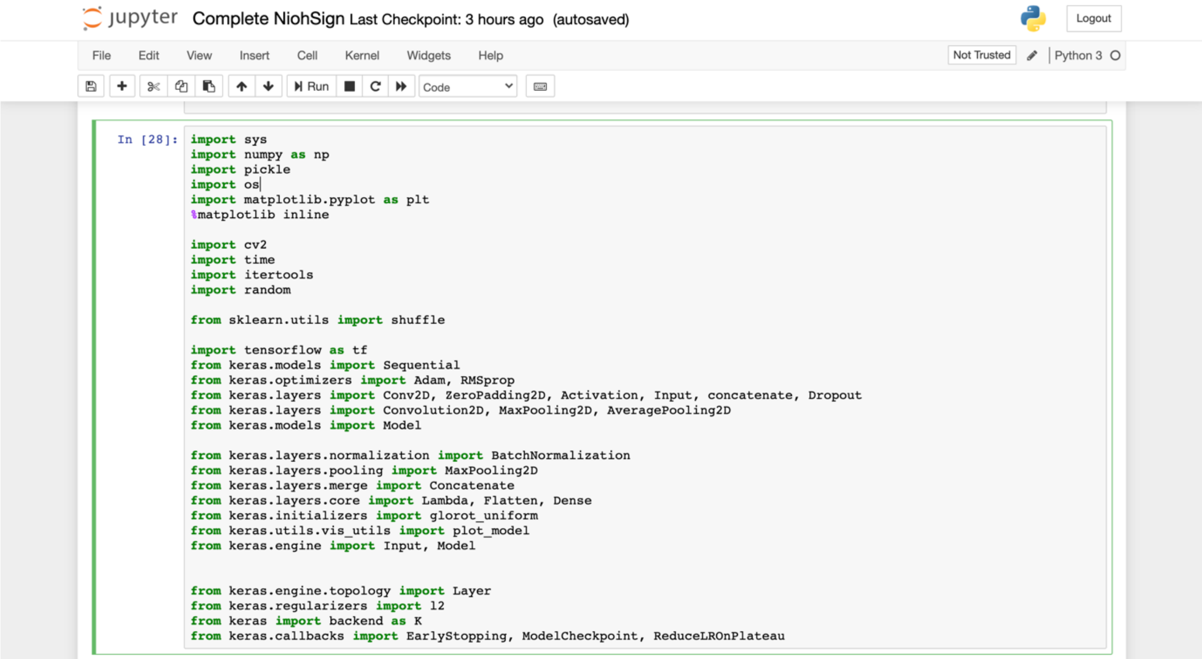



Fig 1: Importing required libraries.

Figure 2 below specifies the location of the dataset along with checking the contents of the dataset.

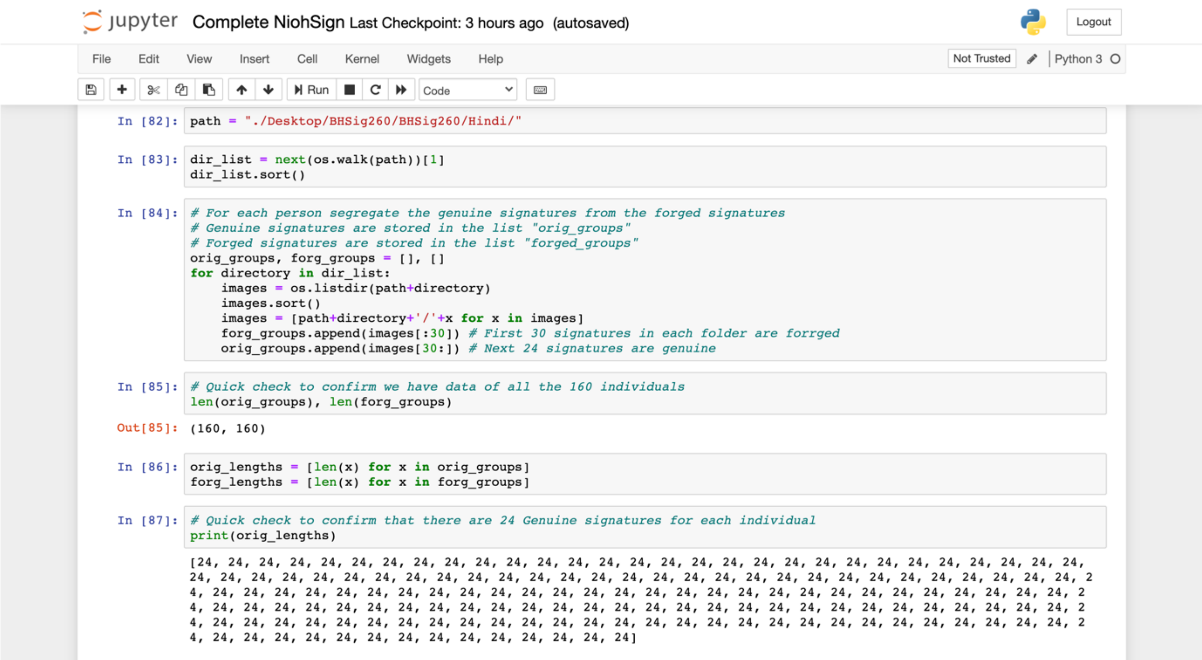



Fig 2: Path Specification and Data exploration

Figure 3 & 4 show further exploration of the data through the visualization of the signatures


Fig 3: Further Data exploration

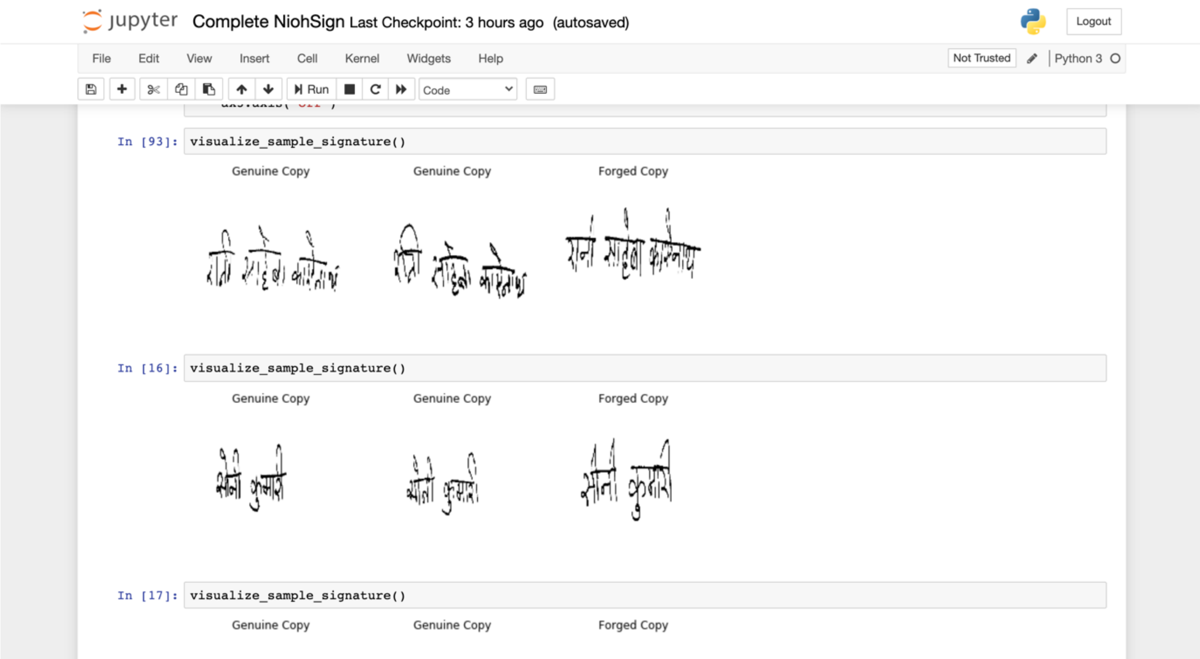



Fig 4: Signature Visualization

Figure 5 shows the generation of the signatures as inputs for the model by splitting them into pairwise of genuine – genuine and genuine- forged signatures.

jupyter Complete NiohSign Last Checkpoint: 3 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3

```
In [94]: def generate_batch(orig_groups, forg_groups, batch_size = 32):
             '''Function to generate a batch of data with batch_size number of data points
             Half of the data points will be Genuine-Genuine pairs and half will be Genuine-Forged pairs'''
             while True:
                 orig_pairs = []
                 forg_pairs = []
                 gen_gen_labels = []
                 gen_for_labels = []
                 all_pairs = []
                 all_labels = []

                 # Here we create pairs of Genuine-Genuine image names and Genuine-Forged image names
                 # For every person we have 24 genuine signatures, hence we have
                 # 24 choose 2 = 276 Genuine-Genuine image pairs for one person.
                 # To make Genuine-Forged pairs, we pair every Genuine signature of a person
                 # with 12 randomly sampled Forged signatures of the same person.
                 # Thus we make 24 * 12 = 300 Genuine-Forged image pairs for one person.
                 # In all we have 120 person's data in the training data.
                 # Total no. of Genuine-Genuine pairs = 120 * 276 = 33120
                 # Total number of Genuine-Forged pairs = 120 * 300 = 36000
                 # Total no. of data points = 33120 + 36000 = 69120
                 for orig, forg in zip(orig_groups, forg_groups):
                     orig_pairs.extend(list(itertools.combinations(orig, 2)))
                     for i in range(len(forg)):
                         forg_pairs.extend(list(itertools.product(orig[i:i+1], random.sample(forg, 12))))

                 # Label for Genuine-Genuine pairs is 1
                 # Label for Genuine-Forged pairs is 0
                 gen_gen_labels = [1]*len(orig_pairs)
                 gen_for_labels = [0]*len(forg_pairs)

                 # Concatenate all the pairs together along with their labels and shuffle them
                 all_pairs = orig_pairs + forg_pairs
                 all_labels = gen_gen_labels + gen_for_labels
                 del orig_pairs, forg_pairs, gen_gen_labels, gen_for_labels
                 all_pairs, all_labels = shuffle(all_pairs, all_labels)
```

Fig 5: Pair wise Split

Figure 6, 7, 8 & 9 represent the InceptionResNetV2 network created for the Siamese network

jupyter Complete NiohSign Last Checkpoint: 3 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3

```
In [99]: # we reduce # filters by factor of 8 compared to original inception-v4
         nb_filters_reduction_factor = 8



         def inception_resnet_v2_stem(x):
              # in original inception-resnet-v2, conv stride is 2
             x = Convolution2D(32//nb_filters_reduction_factor, 3, 3, subsample=(1, 1), activation='relu',
                               init='he_normal', border_mode='valid', dim_ordering='tf')(x)
             x = Convolution2D(32//nb_filters_reduction_factor, 3, 3, subsample=(1, 1), activation='relu',
                               init='he_normal', border_mode='valid', dim_ordering='tf')(x)
             x = Convolution2D(64//nb_filters_reduction_factor, 3, 3, subsample=(1, 1), activation='relu',
                               init='he_normal', border_mode='same', dim_ordering='tf')(x)

                 # in original inception-resnet-v2, stride is 2
             a = MaxPooling2D((3, 3), strides=(1, 1), border_mode='valid', dim_ordering='tf')(x)
                 # in original inception-resnet-v2, conv stride is 2
             b = Convolution2D(96//nb_filters_reduction_factor, 3, 3, subsample=(1, 1), activation='relu',
                               init='he_normal', border_mode='valid', dim_ordering='tf')(x)
             x = concatenate([a, b], axis=-1)

             a = Convolution2D(64//nb_filters_reduction_factor, 1, 1, subsample=(1, 1), activation='relu',
                           init='he_normal', border_mode='same', dim_ordering='tf')(x)
             a = Convolution2D(96//nb_filters_reduction_factor, 3, 3, subsample=(1, 1), activation='relu',
                           init='he_normal', border_mode='valid', dim_ordering='tf')(a)
             b = Convolution2D(64//nb_filters_reduction_factor, 1, 1, subsample=(1, 1), activation='relu',
                           init='he_normal', border_mode='same', dim_ordering='tf')(x)
             b = Convolution2D(64//nb_filters_reduction_factor, 7, 1, subsample=(1, 1), activation='relu',
                           init='he_normal', border_mode='same', dim_ordering='tf')(b)
             b = Convolution2D(64//nb_filters_reduction_factor, 1, 7, subsample=(1, 1), activation='relu',
                           init='he_normal', border_mode='same', dim_ordering='tf')(b)
             b = Convolution2D(96//nb_filters_reduction_factor, 3, 3, subsample=(1, 1), activation='relu',
                           init='he_normal', border_mode='valid', dim_ordering='tf')(b)
             x = concatenate([a, b], axis=-1)
```

Fig 6: InceptionResNetV2 Network (1)

```
jupyter   Complete NiohSign Last Checkpoint: 3 hours ago  (autosaved)
    # in original inception-resnet-v2, conv stride should be 2
    a = Convolution2D(192//nb_filters_reduction_factor, 3, 3, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='valid', dim_ordering='tf')(x)
    # in original inception-resnet-v2, stride is 2
    b = MaxPooling2D((3, 3), strides=(1, 1), border_mode='valid', dim_ordering='tf')(x)
    x = concatenate([a, b], axis=-1)

    x = Activation('relu')(x)

    return x


def inception_resnet_v2_A(x):
    shortcut = x

    a = Convolution2D(32//nb_filters_reduction_factor, 1, 1, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='same', dim_ordering='tf')(x)

    b = Convolution2D(32//nb_filters_reduction_factor, 1, 1, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='same', dim_ordering='tf')(x)
    b = Convolution2D(32//nb_filters_reduction_factor, 3, 3, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='same', dim_ordering='tf')(b)

    c = Convolution2D(32//nb_filters_reduction_factor, 1, 1, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='same', dim_ordering='tf')(x)
    c = Convolution2D(48//nb_filters_reduction_factor, 3, 3, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='same', dim_ordering='tf')(c)
    c = Convolution2D(64//nb_filters_reduction_factor, 3, 3, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='same', dim_ordering='tf')(c)

    x = concatenate([a, b, c],axis=-1)
    x = Convolution2D(384//nb_filters_reduction_factor, 1, 1, subsample=(1, 1), activation='linear',
                      init='he_normal', border_mode='same', dim_ordering='tf')(x)

    x = concatenate([shortcut, x])
    x = Activation('relu')(x)

    return x
```

Fig 7: InceptionResNetV2 Network (2)

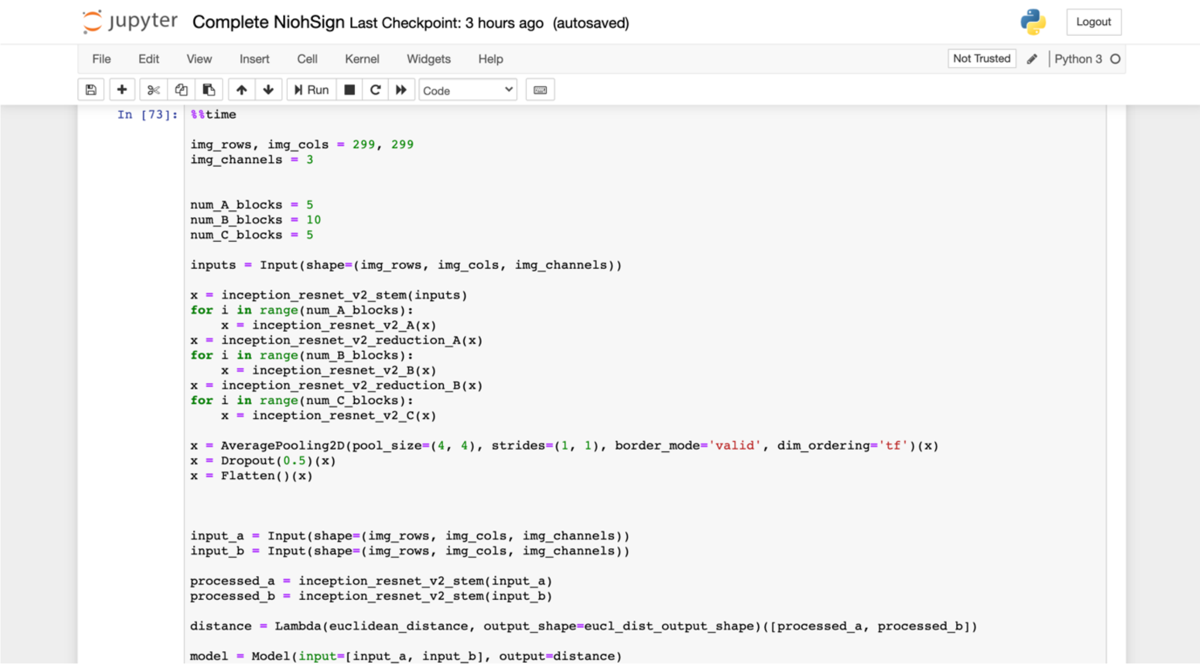
```
jupyter   Complete NiohSign Last Checkpoint: 3 hours ago  (autosaved)
def inception_resnet_v2_reduction_A(x):
    a = MaxPooling2D((3, 3), strides=(2, 2), border_mode='valid', dim_ordering='tf')(x)
    b = Convolution2D(384//nb_filters_reduction_factor, 3, 3, subsample=(2, 2), activation='relu',
                      init='he_normal', border_mode='valid', dim_ordering='tf')(x)
    c = Convolution2D(256//nb_filters_reduction_factor, 1, 1, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='same', dim_ordering='tf')(x)
    c = Convolution2D(256//nb_filters_reduction_factor, 3, 3, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='same', dim_ordering='tf')(c)
    c = Convolution2D(384//nb_filters_reduction_factor, 3, 3, subsample=(2, 2), activation='relu',
                      init='he_normal', border_mode='valid', dim_ordering='tf')(c)

    x = concatenate([a, b, c], axis=-1)

    return x


def inception_resnet_v2_B(x):
    shortcut = x

    a = Convolution2D(192//nb_filters_reduction_factor, 1, 1, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='same', dim_ordering='tf')(x)

    b = Convolution2D(128//nb_filters_reduction_factor, 1, 1, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='same', dim_ordering='tf')(x)
    b = Convolution2D(160//nb_filters_reduction_factor, 1, 7, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='same', dim_ordering='tf')(b)
    b = Convolution2D(192//nb_filters_reduction_factor, 7, 1, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='same', dim_ordering='tf')(b)

    x = concatenate([a, b], axis=-1)
    x = Convolution2D(1154//nb_filters_reduction_factor, 1, 1, subsample=(1, 1), activation='linear',
                      init='he_normal', border_mode='same', dim_ordering='tf')(x)

    x = concatenate([shortcut, x])
    x = Activation('relu')(x)

    return x
```

Fig 8: InceptionResNetV2 Network (3)

```
def inception_resnet_v2_reduction_B(x):
    a = MaxPooling2D((3, 3), strides=(2, 2), border_mode='valid', dim_ordering='tf')(x)
    b = Convolution2D(256//nb_filters_reduction_factor, 1, 1, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='same', dim_ordering='tf')(x)
    b = Convolution2D(288//nb_filters_reduction_factor, 3, 3, subsample=(2, 2), activation='relu',
                      init='he_normal', border_mode='valid', dim_ordering='tf')(b)
    c = Convolution2D(256//nb_filters_reduction_factor, 1, 1, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='same', dim_ordering='tf')(x)
    c = Convolution2D(288//nb_filters_reduction_factor, 3, 3, subsample=(2, 2), activation='relu',
                      init='he_normal', border_mode='valid', dim_ordering='tf')(c)
    d = Convolution2D(256//nb_filters_reduction_factor, 1, 1, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='same', dim_ordering='tf')(x)
    d = Convolution2D(288//nb_filters_reduction_factor, 3, 3, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='same', dim_ordering='tf')(d)
    d = Convolution2D(320//nb_filters_reduction_factor, 3, 3, subsample=(2, 2), activation='relu',
                      init='he_normal', border_mode='valid', dim_ordering='tf')(d)

    x = concatenate([a, b, c, d],axis=-1)

    return x

def inception_resnet_v2_C(x):
    shortcut = x

    a = Convolution2D(192//nb_filters_reduction_factor, 1, 1, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='same', dim_ordering='tf')(x)

    b = Convolution2D(192//nb_filters_reduction_factor, 1, 1, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='same', dim_ordering='tf')(x)
    b = Convolution2D(224//nb_filters_reduction_factor, 1, 3, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='same', dim_ordering='tf')(b)
    b = Convolution2D(256//nb_filters_reduction_factor, 3, 1, subsample=(1, 1), activation='relu',
                      init='he_normal', border_mode='same', dim_ordering='tf')(b)
```

Fig 9: InceptionResNetV2 Network (4)

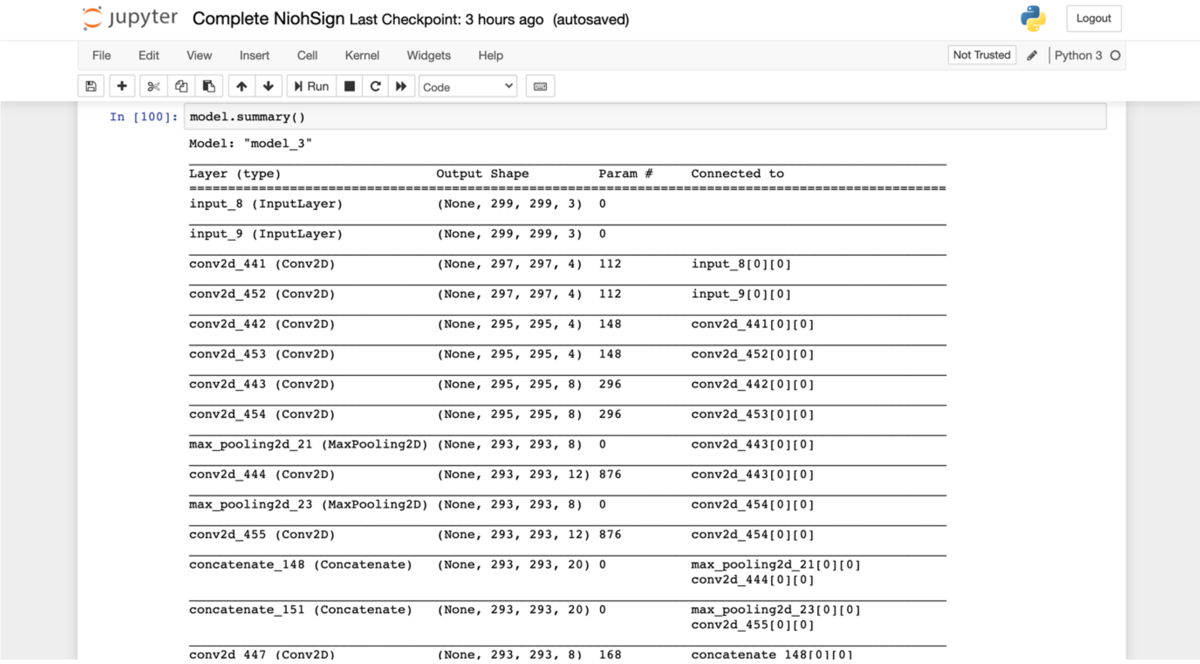
Fig 10 represents the steps taken to create the Siamese Network from the above base network.



```
In [73]: %%time

img_rows, img_cols = 299, 299
img_channels = 3


num_A_blocks = 5
num_B_blocks = 10
num_C_blocks = 5

inputs = Input(shape=(img_rows, img_cols, img_channels))

x = inception_resnet_v2_stem(inputs)
for i in range(num_A_blocks):
    x = inception_resnet_v2_A(x)
x = inception_resnet_v2_reduction_A(x)
for i in range(num_B_blocks):
    x = inception_resnet_v2_B(x)
x = inception_resnet_v2_reduction_B(x)
for i in range(num_C_blocks):
    x = inception_resnet_v2_C(x)

x = AveragePooling2D(pool_size=(4, 4), strides=(1, 1), border_mode='valid', dim_ordering='tf')(x)
x = Dropout(0.5)(x)
x = Flatten()(x)



input_a = Input(shape=(img_rows, img_cols, img_channels))
input_b = Input(shape=(img_rows, img_cols, img_channels))

processed_a = inception_resnet_v2_stem(input_a)
processed_b = inception_resnet_v2_stem(input_b)

distance = Lambda(euclidean_distance, output_shape=eucl_dist_output_shape)([processed_a, processed_b])

model = Model(input=[input_a, input_b], output=distance)
```

Fig 10: Creation of the Siamese network

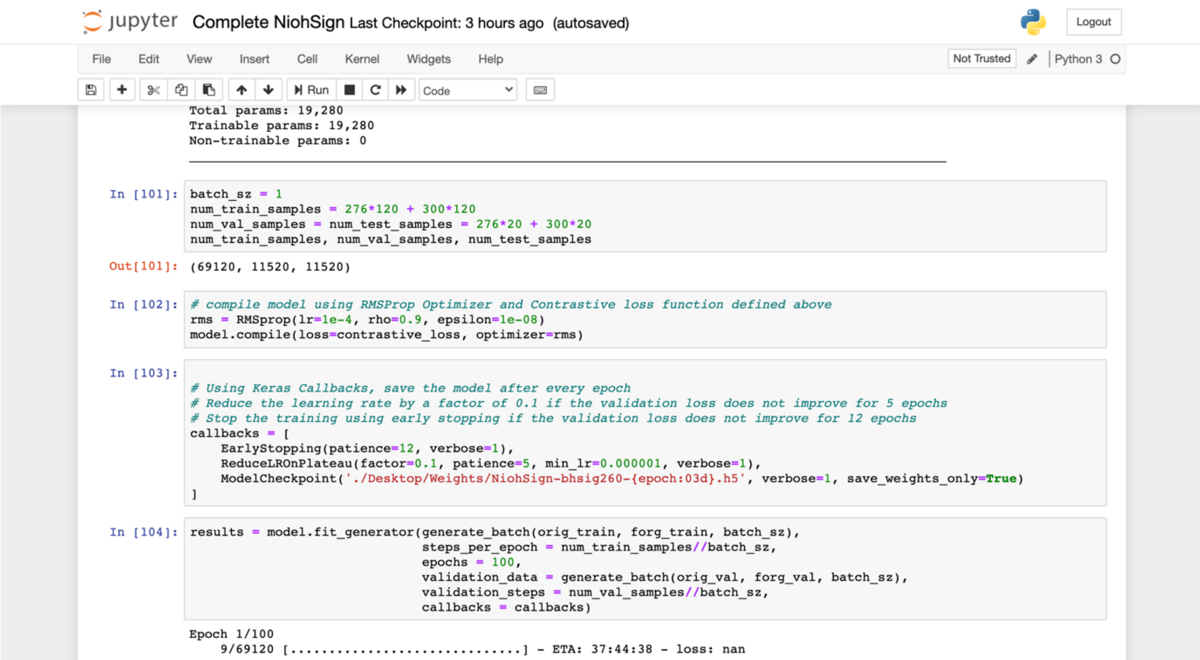
Figure 11 represents the summary of the created model



```
In [100]: model.summary()

Model: "model_3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_8 (InputLayer)            (None, 299, 299, 3)  0
__________________________________________________________________________________________________
input_9 (InputLayer)            (None, 299, 299, 3)  0
__________________________________________________________________________________________________
conv2d_441 (Conv2D)             (None, 297, 297, 4)  112         input_8[0][0]
__________________________________________________________________________________________________
conv2d_452 (Conv2D)             (None, 297, 297, 4)  112         input_9[0][0]
__________________________________________________________________________________________________
conv2d_442 (Conv2D)             (None, 295, 295, 4)  148         conv2d_441[0][0]
__________________________________________________________________________________________________
conv2d_453 (Conv2D)             (None, 295, 295, 4)  148         conv2d_452[0][0]
__________________________________________________________________________________________________
conv2d_443 (Conv2D)             (None, 295, 295, 8)  296         conv2d_442[0][0]
__________________________________________________________________________________________________
conv2d_454 (Conv2D)             (None, 295, 295, 8)  296         conv2d_453[0][0]
__________________________________________________________________________________________________
max_pooling2d_21 (MaxPooling2D) (None, 293, 293, 8)  0           conv2d_443[0][0]
__________________________________________________________________________________________________
conv2d_444 (Conv2D)             (None, 293, 293, 12) 876         conv2d_443[0][0]
__________________________________________________________________________________________________
max_pooling2d_23 (MaxPooling2D) (None, 293, 293, 8)  0           conv2d_454[0][0]
__________________________________________________________________________________________________
conv2d_455 (Conv2D)             (None, 293, 293, 12) 876         conv2d_454[0][0]
__________________________________________________________________________________________________
concatenate_148 (Concatenate)   (None, 293, 293, 20) 0           max_pooling2d_21[0][0]
                                                                 conv2d_444[0][0]
__________________________________________________________________________________________________
concatenate_151 (Concatenate)   (None, 293, 293, 20) 0           max_pooling2d_23[0][0]
                                                                 conv2d_455[0][0]
__________________________________________________________________________________________________
conv2d_447 (Conv2D)             (None, 293, 293, 8)  168         concatenate_148[0][0]
```

Fig 11: Model Summary

Figure 12 shows the steps taken to fit and run the model



```
Total params: 19,280
Trainable params: 19,280
Non-trainable params: 0
__________________________________________________________________________________________________

In [101]: batch_sz = 1
          num_train_samples = 276*120 + 300*120
          num_val_samples = num_test_samples = 276*20 + 300*20
          num_train_samples, num_val_samples, num_test_samples

Out[101]: (69120, 11520, 11520)

In [102]: # compile model using RMSProp Optimizer and Contrastive loss function defined above
          rms = RMSprop(lr=1e-4, rho=0.9, epsilon=1e-08)
          model.compile(loss=contrastive_loss, optimizer=rms)

In [103]: 
          # Using Keras Callbacks, save the model after every epoch
          # Reduce the learning rate by a factor of 0.1 if the validation loss does not improve for 5 epochs
          # Stop the training using early stopping if the validation loss does not improve for 12 epochs
          callbacks = [
              EarlyStopping(patience=12, verbose=1),
              ReduceLROnPlateau(factor=0.1, patience=5, min_lr=0.000001, verbose=1),
              ModelCheckpoint('./Desktop/Weights/NiohSign-bhsig260-{epoch:03d}.h5', verbose=1, save_weights_only=True)
          ]

In [104]: results = model.fit_generator(generate_batch(orig_train, forg_train, batch_sz),
                                        steps_per_epoch = num_train_samples//batch_sz,
                                        epochs = 100,
                                        validation_data = generate_batch(orig_val, forg_val, batch_sz),
                                        validation_steps = num_val_samples//batch_sz,
                                        callbacks = callbacks)

          Epoch 1/100
              9/69120 [..............................] - ETA: 37:44:38 - loss: nan
```

Fig 12: Running the Model

Figure 13 shows the steps to load the weights after running the model and the creation of the function for the accuracy and the threshold that will be used for the signatures.

jupyter Complete NiohSign Last Checkpoint: 3 hours ago (autosaved)

```
In [105]: def compute_accuracy_roc(predictions, labels):
              '''Compute ROC accuracy with a range of thresholds on distances.
              '''
              dmax = np.max(predictions)
              dmin = np.min(predictions)
              nsame = np.sum(labels == 1)
              ndiff = np.sum(labels == 0)

              step = 0.01
              max_acc = 0
              best_thresh = -1

              for d in np.arange(dmin, dmax+step, step):
                  idx1 = predictions.ravel() <= d
                  idx2 = predictions.ravel() > d

                  tpr = float(np.sum(labels[idx1] == 1)) / nsame
                  tnr = float(np.sum(labels[idx2] == 0)) / ndiff
                  acc = 0.5 * (tpr + tnr)
          #       print ('ROC', acc, tpr, tnr)

                  if (acc > max_acc):
                      max_acc, best_thresh = acc, d

              return max_acc, best_thresh
```

**Load the weights from the epoch which gave the best validation accuracy**

```
In [ ]: model.load_weights('./Weights/NiohSign-bhsig260-{epoch:03d}.h5')

In [69]: test_gen = generate_batch(orig_test, forg_test, 1)
         pred, tr_y = [], []
         for i in range(num_test_samples):
             (img1, img2), label = next(test_gen)
```

Fig 13: Threshold and accuracy calculation

Figure 14 shows accuracy and threshold along with the function for the prediction of the signatures.

jupyter Complete NiohSign Last Checkpoint: 3 hours ago (autosaved)

```
In [70]: tr_acc, threshold = compute_accuracy_roc(np.array(pred), np.array(tr_y))
         tr_acc, threshold

Out[70]: (0.817140551248915, 0.5500106516727101)
```

**Accuracy = 81.71% and Threshold = 0.55**

if the calculated differnce score is less than 0.55, we predict that the signature is Genuine and if the difference score is greater than 0.55, we predict that the signature is forged

```
In [75]: def predict_score():
             '''Predict distance score and classify test images as Genuine or Forged'''
             test_point, test_label = next(test_gen)
             img1, img2 = test_point[0], test_point[1]

             fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (10, 10))
             ax1.imshow(np.squeeze(img1), cmap='gray')
             ax2.imshow(np.squeeze(img2), cmap='gray')
             ax1.set_title('Genuine')
             if test_label == 1:
                 ax2.set_title('Genuine')
             else:
                 ax2.set_title('Forged')
             ax1.axis('off')
             ax2.axis('off')
             plt.show()
             result = model.predict([img1, img2])
             diff = result[0][0]
             print("Difference Score = ", diff)
             if diff > threshold:
                 print("Its a Forged Signature")
             else:
                 print("Its a Genuine Signature")
```

Fig 14: Prediction function

Figure 15 shows the output of a prediction which is representative of the ability of the network to classify signatures as authentic or forged


Fig 15: Final Outputs