National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Shahil Shaik
Student ID: x18131743

School of Computing
National College of Ireland

Supervisor: Dr. Vladimir Milosavljevic

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Shahil Shaik
Student ID:	x18131743
Programme:	Data Analytics
Year:	2019
Module:	MSc Research Project
Supervisor:	Dr. Vladimir Milosavljevic
Submission Due Date:	12/12/2019
Project Title:	Configuration Manual
Word Count:	944
Page Count:	14

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:	
Date:	2nd February 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

Shahil Shaik
x18131743

1 Introduction

This configuration manual consists of the information regarding the system setup, Software requirements, Hardware configuration used for execution of the model. A step wise implementation of the code is described with the help of images of the code along with brief explanation. Few images of the output are also provided for better understanding of the code implementation. All the code and details provided are for the project "OpenPose based Gait Recognition using triplet loss Architecture". (**? ? ? ?**).

2 System Setup

This section discusses about the list of software and hardware utilized to implement this research project.

2.1 Software

- Anaconda
- Spyder
- Python
- TensorFlow

2.2 Hardware

- Processor: Intel CORE i7@ 270 GHz
- RAM: 16GB
- System type: Windows 10 64-bit Operating System, x64based processor
- Hard Disk: 1 TB
- Graphic Card: NVIDIA Ge force 1060ti 6GB

3 Project development

Implementation of this project includes several steps such as data filtering, Key point extraction, data pre-processing, Manual feature extraction, Triplet loss architecture, KNN algorithm and Evaluation.

```python
#Import required libraries
import cv2
import time
import numpy as np
import os
import matplotlib.pyplot as plt
import math
import re
import glob
import pickle
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.datasets import mnist

## for Model definition/training
from keras.models import Model, load_model
from keras.layers import Input, Flatten, Dense, concatenate,  Dropout
from keras.optimizers import Adam

from keras.utils import plot_model
from keras.callbacks import ModelCheckpoint

## required for semi-hard triplet loss:
from tensorflow.python.ops import array_ops
from tensorflow.python.ops import math_ops
from tensorflow.python.framework import dtypes
import tensorflow as tf
```

Figure 1: Import required libraries

3.1 Data Filtering

CASIA-B data set provided by CBSR is used for this project. It has videos of 124 subjects in 11 views and 3 modes. Only Normal mode 90-degree views are used for this project. Glob library of python is used for filtering of these videos.Figure1 Figure2

```python
import glob
import shutil
dest_dir = r"D:\Thesis\al902"
i=0
for file in glob.glob(r'D:\Thesis\DatasetB-2\DatasetB-2\video\***-nm-*-090.avi'):
    i=i+1
    print(i,file)
    shutil.copy(file, dest_dir)
```

Figure 2: Data Filtering

3.2 Data Extraction

OpenPose model is loaded using the OpenCV library and is initialized by COCO weights. Using this model key points for every frame in every video are estimated. Apart from these key points certain other manual features are also calculated such as angle between limbs using the angle function.Figure3 - Figure7

```python
#Get the list of videos in to a file
flist=glob.glob(r'D:\Thesis\al902\*')
input_source = flist[0]

#Read video frame by frame using opencv videocapture
cap = cv2.VideoCapture(input_source)
hasFrame, frame = cap.read()
vid_writer = cv2.VideoWriter('output.avi',cv2.VideoWriter_fourcc('M','J','P','G'), 10, (frame.shape[1],frame.shape[0]))

#load the COCO OpenCV weights
net = cv2.dnn.readNetFromCaffe(protoFile, weightsFile)

#Run OpenPose Video captue for every frame in every and store the co-ordinate as list.
#Write the coordinate included frames in a video

pot=[]
n=0
nin=[]
while cv2.waitKey(1) < 0:
    t = time.time()
    hasFrame, frame = cap.read()
    frameCopy = np.copy(frame)
    if not hasFrame:
        pot.append(input_source)
        nin.append(pot)
        n=n+1
        if n>371:
            cv2.waitKey()
            break
        input_source = flist[n]
        print(input_source)
        cap = cv2.VideoCapture(input_source)
        hasFrame, frame = cap.read()
        pot=[]
        with open("tmp.txt", "wb") as fp:
            pickle.dump(nin, fp)

        #cv2.waitKey()
        #break
```

Figure 3: Keypoint Extraction:1

```python
    frameWidth = frame.shape[1]
    frameHeight = frame.shape[0]

    inpBlob = cv2.dnn.blobFromImage(frame, 1.0 / 255, (inWidth, inHeight),
                              (0, 0, 0), swapRB=False, crop=False)
    net.setInput(inpBlob)
    output = net.forward()

    H = output.shape[2]
    W = output.shape[3]
    # Empty list to store the detected keypoints
    points = []

    for i in range(nPoints):
        # confidence map of corresponding body's part.
        probMap = output[0, i, :, :]

        # Find global maxima of the probMap.
        minVal, prob, minLoc, point = cv2.minMaxLoc(probMap)

        # Scale the point to fit on the original image
        x = (frameWidth * point[0]) / W
        y = (frameHeight * point[1]) / H

        if prob > threshold :
            cv2.circle(frameCopy, (int(x), int(y)), 8, (0, 255, 255), thickness=-1, lineType=cv2.FILLED)
            cv2.putText(frameCopy, "{}".format(i), (int(x), int(y)), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, lineType=cv2.LINE_AA)

            # Add the point to the list if the probability is greater than the threshold
            points.append((int(x), int(y)))
        else :
            points.append((None,None))
    #Calculate different angles
    if points[11][0] and points[9][0] and points[12][0]:
            ang=getAngle(points[9], points[11], points[12])
            if ang>180:
                ang=360-ang

            points.append(ang)
    else:
        points.append(None)
```

Figure 4: Keypoint Extraction:2

```
    if points[11][0] and points[5][0] and points[6][0]:

            ang=getAngle(points[11], points[5], points[6])
            if ang>180:
                ang=360-ang

            points.append(ang)
    else:
        points.append(None)
    if points[11][0] and points[1][0]:
        p1=points[11]
        p2=points[1]
        l=math.sqrt( ((p1[0]-p2[0])**2)+((p1[1]-p2[1])**2) )

        points.append(l)
    else:
        points.append(None)
    if points[11][0] and points[12][0]:
        p1=points[11]
        p2=points[12]
        l=math.sqrt( ((p1[0]-p2[0])**2)+((p1[1]-p2[1])**2) )
        points.append(l)
    else:
        points.append(None)
    if points[5][0] and points[6][0]:
        p1=points[5]
        p2=points[6]
        l=math.sqrt( ((p1[0]-p2[0])**2)+((p1[1]-p2[1])**2) )

        points.append(l)
    else:
        points.append(None)
    pot.append(points)
```

Figure 5: Keypoint Extraction:3

```
    # Draw Skeleton
    for pair in POSE_PAIRS:
        partA = pair[0]
        partB = pair[1]

        if points[partA][0] and points[partB][0]:
            cv2.line(frame, points[partA], points[partB], (0, 255, 255), 1, lineType=cv2.LINE_AA)
            cv2.ellipse(frame, points[partA], (4, 4), 0, 0, 360, (255, 255, 255), cv2.FILLED)
            cv2.ellipse(frame, points[partB], (4, 4), 0, 0, 360, (255, 255, 255), cv2.FILLED)
            cv2.putText(frame, str(partA), points[partA], cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255),2,cv2.LINE_AA)
            cv2.putText(frame, str(partB), points[partB], cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255),2,cv2.LINE_AA)
            ###cv2.circle(frame, points[partA], 8, (0, 0, 255), thickness=-2, lineType=cv2.FILLED)
            #cv2.circle(frame, points[partB], 8, (0, 0, 255), thickness=-2, lineType=cv2.FILLED)

    #cv2.putText(frame, "time taken = {:.2f} sec".format(time.time() - t), (50, 50), cv2.FONT_HERSHEY_COMPLEX, .8, (255, 50,
    # cv2.putText(frame, "OpenPose using OpenCV", (50, 50), cv2.FONT_HERSHEY_COMPLEX, 1, (255, 50, 0), 2, lineType=cv2.LINE_A
    # cv2.imshow('Output-Keypoints', frameCopy)
    #cv2.imshow('Output-Skeleton', frame)
    #cv2.waitKey()

    vid_writer.write(frame)

vid_writer.release()
```

Figure 6: Keypoint Extraction:4

```
#Dump the coordinate list in a pickle file and load when required
with open("full.txt", "wb") as fp:
    pickle.dump(nin, fp)#Pickling
with open("full.txt", "rb") as fp:
    b = pickle.load(fp)
```

Figure 7: Save output as pickle file

3.3 Manual Feature Extraction

Using the extracted key points various features are defined and calculated for each individual. Features such as length of the limbs, Max angle between Limbs, Length of the Gait stride and amplitude of gait cycle are calculated.Figure8

```
#Calculate various manual features using the cordinates for every video and save as a csv
j=0
for potf in b:
    print(potf[-1])
    p1=[potf[x][1][1]for x in range(0,(len(potf)-1))]
    p1=[int(x) for x in p1 if x is not None]
    p1s=pd.Series(p1).value_counts()
    p1m=p1s.index[0]
    p1d=np.median(p1)
    #p1d=p1s.index[0]-p1s.index[1]

    p11=[potf[x][11][1]for x in range(0,(len(potf)-1))]
    p11=[int(x) for x in p11 if x is not None]
    p11s=pd.Series(p11).value_counts()
    p11m=p11s.index[0]
    p11d=np.median(p11)
    #if len(p11s) >1 :
        #p11d=p11s.index[0]-p11s.index[1]
    #else :
    #    p11d=0


    p12=[potf[x][12][1]for x in range(0,(len(potf)-1))]
    p12=[int(x) for x in p12 if x is not None]
    p12s=pd.Series(p12).value_counts()
    p12m=p1s.index[0]
    p12d=np.median(p12)
    #p12d=p12s.index[0]-p12s.index[1]

    p13=[potf[x][13][1]for x in range(0,(len(potf)-1))]
    p13=[int(x) for x in p13 if x is not None]
    p13s=pd.Series(p13).value_counts()
    p13m=p13s.index[0]
    p13d=np.median(p13)
    #p13d=p13s.index[0]-p13s.index[1]
```

Figure 8: Manual Feature extraction:1

Calculated Manual features for every video are stored as a row in a dataframe. Dataframe is saved as csv for future use.Figure10

```
p19=[potf[x][19]for x in range(0,(len(potf)-1))]
p19=[int(x) for x in p19 if x is not None]
p19s=pd.Series(p19).value_counts()
p19m=p19s.index[0]
p19d=np.median(p19)
#p19d=p19s.index[0]-p19s.index[1]

p20=[potf[x][20]for x in range(0,(len(potf)-1))]
p20=[int(x) for x in p20 if x is not None]
p20s=pd.Series(p20).value_counts()
p20m=p20s.index[0]
p20d=np.median(p20)
#if len(p20s)>1:
#    p20d=p20s.index[0]-p20s.index[1]
#else:
#    p20d=0


p21=[potf[x][21]for x in range(0,(len(potf)-1))]
p21=[int(x) for x in p21 if x is not None]
p21s=pd.Series(p21).value_counts()
p21m=p21s.index[0]
p21d=np.median(p21)
#p21d=p21s.index[0]-p21s.index[1]

p22=[potf[x][22]for x in range(0,(len(potf)-1))]
p22=[int(x) for x in p22 if x is not None]
p22s=pd.Series(p22).value_counts()
p22m=p22s.index[0]
p22d=np.median(p22)
#p22d=p22s.index[0]-p22s.index[1]

pnam=str(potf[-1])[16:19]#[18:20]

l1=[[p1m,p1d,p11m,p11d,p12m,p12d,p13m,p13d,p7m,p7d,p18m,p18d,p19m,p19d,p20m,p20d,p21m,p21d,p22m,p22d,pnam]]
if j==0:
    fin=pd.DataFrame(l1)
    j=1
else:
    fin=fin.append(l1)
```

Figure 9: Manual Feature extraction:2

```
fin.columns=['p1m','p1d','p11m','p11d','p12m','p12d','p13m','p13d','p7m','p7d','p18m','p18d','p19m','p19d','p20m','p20d','p21m']

fin.to_csv('finlist.csv',index=False)

fin=pd.read_csv('finlist.csv')
```

Figure 10: Save features as CSV file

3.4 Knn for Manual Features

A nearest neighbour algorithm is trained for the manual feature data. Data is initially split into train and test using sklearn package. Error rate for various k values are calculated and plotted for calculation of optimum K value. Finally, model is built and evaluated for optimum k value.Figure11

```
#Split in to train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,stratify=y, test_size=0.3,random_state=15)

from sklearn.neighbors import KNeighborsClassifier
# Calculating error for various K values to chose the optimum Kvalue
error = []
for i in range(3, 10):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))

plt.rc('axes', labelsize=18)
plt.figure(figsize=(12, 6))
plt.plot(range(3, 10), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value(triplet loss)')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
plt.show()
plt.savefig('img9.png')

#Build Knn with optimum K value and print the classification report

classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

from sklearn.metrics import accuracy_score

accuracy_score(y_test, y_pred)
```

Figure 11: Knn for manual features

3.5 Triplet Loss Architecture

Triplet loss architecture creates dynamic features of the data in form of a 64- dimensional vector. These embeddings of the same class or individual are clustered on training for triplet loss. For implementation of Triplet loss algorithm. Tutorial provided by TensorFlow is utilized and various functions required for calculation of triplet loss are developed.

A pairwise distance function gives the distance matrix which has distance between every pair in the input of the function.Figure12

Masked Maximum and Masked Minimum functions are used to compute the axis wise maximum and minimum respectively.Figure13

Triplet loss function is used to calculate the triplet loss for the given batch of embedding. This used above defined 3 functions for calculation.Figure14

A Base architecture function is created for the Creation of base architecture in the main triplet loss function. It consists of few dense and dropout layer and an initial flatten layer to reshape the input.Figure15

All the parameters are defined such as batch size, input shape , validation and test library. Main architecture is built with help of base architecture function and triplet loss function. This model is trained for 10000 epochs.Figure16 Figure17

```python
def pairwise_distance(feature, squared=False):
    """Computes the pairwise distance matrix with numerical stability.

    output[i, j] = || feature[i, :] - feature[j, :] ||_2

    Args:
      feature: 2-D Tensor of size [number of data, feature dimension].
      squared: Boolean, whether or not to square the pairwise distances.

    Returns:
      pairwise_distances: 2-D Tensor of size [number of data, number of data].
    """
    pairwise_distances_squared = math_ops.add(
        math_ops.reduce_sum(math_ops.square(feature), axis=[1], keepdims=True),
        math_ops.reduce_sum(
            math_ops.square(array_ops.transpose(feature)),
            axis=[0],
            keepdims=True)) - 2.0 * math_ops.matmul(feature,
                                                    array_ops.transpose(feature))

    # Deal with numerical inaccuracies. Set small negatives to zero.
    pairwise_distances_squared = math_ops.maximum(pairwise_distances_squared, 0.0)
    # Get the mask where the zero distances are at.
    error_mask = math_ops.less_equal(pairwise_distances_squared, 0.0)

    # Optionally take the sqrt.
    if squared:
        pairwise_distances = pairwise_distances_squared
    else:
        pairwise_distances = math_ops.sqrt(
            pairwise_distances_squared + math_ops.to_float(error_mask) * 1e-16)

    # Undo conditionally adding 1e-16.
    pairwise_distances = math_ops.multiply(
        pairwise_distances, math_ops.to_float(math_ops.logical_not(error_mask)))

    num_data = array_ops.shape(feature)[0]
    # Explicitly set diagonals to zero.
    mask_offdiagonals = array_ops.ones_like(pairwise_distances) - array_ops.diag(
        array_ops.ones([num_data]))
    pairwise_distances = math_ops.multiply(pairwise_distances, mask_offdiagonals)
    return pairwise_distances
```

Figure 12: Pairwise Distance Function

```
def masked_maximum(data, mask, dim=1):
    """Computes the axis wise maximum over chosen elements.

    Args:
      data: 2-D float `Tensor` of size [n, m].
      mask: 2-D Boolean `Tensor` of size [n, m].
      dim: The dimension over which to compute the maximum.

    Returns:
      masked_maximums: N-D `Tensor`.
        The maximized dimension is of size 1 after the operation.
    """
    axis_minimums = math_ops.reduce_min(data, dim, keepdims=True)
    masked_maximums = math_ops.reduce_max(
        math_ops.multiply(data - axis_minimums, mask), dim,
        keepdims=True) + axis_minimums
    return masked_maximums

def masked_minimum(data, mask, dim=1):
    """Computes the axis wise minimum over chosen elements.

    Args:
      data: 2-D float `Tensor` of size [n, m].
      mask: 2-D Boolean `Tensor` of size [n, m].
      dim: The dimension over which to compute the minimum.

    Returns:
      masked_minimums: N-D `Tensor`.
        The minimized dimension is of size 1 after the operation.
    """
    axis_maximums = math_ops.reduce_max(data, dim, keepdims=True)
    masked_minimums = math_ops.reduce_min(
        math_ops.multiply(data - axis_maximums, mask), dim,
        keepdims=True) + axis_maximums
    return masked_minimums
```

Figure 13: Masked maximum and minimum function

```python
def triplet_loss_adapted_from_tf(y_true, y_pred):
    del y_true
    margin = 1.
    labels = y_pred[:, :1]
    labels = tf.cast(labels, dtype='int32')
    embeddings = y_pred[:, 1:]
    ### Code from Tensorflow function [tf.contrib.losses.metric_learning.triplet_semiha
    # Build pairwise squared distance matrix.
    pdist_matrix = pairwise_distance(embeddings, squared=True)
    # Build pairwise binary adjacency matrix.
    adjacency = math_ops.equal(labels, array_ops.transpose(labels))
    # Invert so we can select negatives only.
    adjacency_not = math_ops.logical_not(adjacency)
    # global batch_size
    batch_size = array_ops.size(labels) # was 'array_ops.size(labels)'
    # Compute the mask.
    pdist_matrix_tile = array_ops.tile(pdist_matrix, [batch_size, 1])
    mask = math_ops.logical_and(
        array_ops.tile(adjacency_not, [batch_size, 1]),
        math_ops.greater(
            pdist_matrix_tile, array_ops.reshape(
                array_ops.transpose(pdist_matrix), [-1, 1])))
    mask_final = array_ops.reshape(
        math_ops.greater(
            math_ops.reduce_sum(
                math_ops.cast(mask, dtype=dtypes.float32), 1, keepdims=True),
            0.0), [batch_size, batch_size])
    mask_final = array_ops.transpose(mask_final)

    adjacency_not = math_ops.cast(adjacency_not, dtype=dtypes.float32)
    mask = math_ops.cast(mask, dtype=dtypes.float32)
    # negatives_outside: smallest D_an where D_an > D_ap.
    negatives_outside = array_ops.reshape(
        masked_minimum(pdist_matrix_tile, mask), [batch_size, batch_size])
    negatives_outside = array_ops.transpose(negatives_outside)
    # negatives_inside: largest D_an.
    negatives_inside = array_ops.tile(
        masked_maximum(pdist_matrix, adjacency_not), [1, batch_size])
    semi_hard_negatives = array_ops.where(
        mask_final, negatives_outside, negatives_inside)

    loss_mat = math_ops.add(margin, pdist_matrix - semi_hard_negatives)

    mask_positives = math_ops.cast(
        adjacency, dtype=dtypes.float32) - array_ops.diag(
        array_ops.ones([batch_size]))

    # In lifted-struct, the authors multiply 0.5 for upper triangular
    #   in semihard, they take all positive pairs except the diagonal.
    num_positives = math_ops.reduce_sum(mask_positives)

    semi_hard_triplet_loss_distance = math_ops.truediv(
        math_ops.reduce_sum(
            math_ops.maximum(
                math_ops.multiply(loss_mat, mask_positives), 0.0)),
        num_positives,
        name='triplet_semihard_loss')

    ### Code from Tensorflow function semi-hard triplet loss ENDS here.
    return semi_hard_triplet_loss_distance
```

Figure 14: Triplet loss function

```python
def create_base_network(image_input_shape, embedding_size):
    """
    Base network to be shared (eq. to feature extraction).
    """
    input_image = Input(shape=image_input_shape)

    x = Flatten()(input_image)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.1)(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.1)(x)
    x = Dense(embedding_size)(x)

    base_network = Model(inputs=input_image, outputs=x)
    plot_model(base_network, to_file='base_network.png', show_shapes=True, show_layer_names=True)
    return base_network
```

Figure 15: Base neural network function

```python
batch_size = 128
epochs = 10000
train_flag = True  # either     True or False

embedding_size = 64

no_of_components = 2  # for visualization -> PCA.fit_transform()

step = 10

# The data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test, y_train, y_test = train_test_split(new7, y,stratify=y, test_size=0.20,random_state=26)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 256.
x_test /= 256.
input_image_shape = (30, 28, 1)
x_val = x_train[:50, :, :]
y_val = y_train[:50]
```

Figure 16: Parameter Initialization

```
if train_flag == True:
    base_network = create_base_network(input_image_shape, embedding_size)

    input_images = Input(shape=input_image_shape, name='input_image') # input layer for images
    input_labels = Input(shape=(1,), name='input_label')    # input layer for labels
    embeddings = base_network([input_images])               # output of network -> embeddings
    labels_plus_embeddings = concatenate([input_labels, embeddings])  # concatenating the labels + embeddings

    # Defining a model with inputs (images, labels) and outputs (labels_plus_embeddings)
    model = Model(inputs=[input_images, input_labels],
                  outputs=labels_plus_embeddings)

    model.summary()
    #plot_model(model, to_file='model.png', show_shapes=True, show_layer_names=True)

    plot_model(base_network, to_file='base_network.png', show_shapes=True, show_layer_names=True)

    # train session
    opt = Adam(lr=0.0001)  # choose optimiser. RMS is good too!

    model.compile(loss=triplet_loss_adapted_from_tf,
                  optimizer=opt)

    filepath = "trip_loss2_ep{epoch:02d}_BS%d.hdf5" % batch_size
    checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=False, period=10000)
    callbacks_list = [checkpoint]

    # Uses 'dummy' embeddings + dummy gt labels. Will be removed as soon as loaded, to free memory
    dummy_gt_train = np.zeros((len(x_train), embedding_size + 1))
    dummy_gt_val = np.zeros((len(x_val), embedding_size + 1))

    x_train = np.reshape(x_train, (len(x_train), x_train.shape[1], x_train.shape[2], 1))
    x_val = np.reshape(x_val, (len(x_val), x_train.shape[1], x_train.shape[2], 1))

    H = model.fit(
        x=[x_train,y_train],
        y=dummy_gt_train,
        batch_size=batch_size,
        epochs=epochs,
        validation_data=([x_val, y_val], dummy_gt_val),
        callbacks=callbacks_list)

    plt.figure(figsize=(8,8))
    plt.plot(H.history['loss'], label='Training loss')
    plt.plot(H.history['val_loss'], label='validation loss')
    plt.legend()
    plt.title('Training loss')
    plt.savefig('img8.png')
    plt.show()
else:

    #####
    model = load_model('trip_loss2_ep10000_BS128.hdf5',
                                  custom_objects={'triplet_loss_adapted_from_tf':triplet_loss_adapted_from_tf})
```

Figure 17: Triplet loss Model

3.6 Knn for Triplet loss architecture

After training the triplet loss architecture, embedding vector is created for all the videos with the help of model. This data is split into train and test using sklearn package. Optimum K value is found by plotting the error rate for various K values. Finally, a Knn model with optimum k value is trained and evaluated. Figure18

```
X = testing_embeddings.predict(np.reshape(new8, (len(new8), 30, 28, 1)))
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,stratify=y, test_size=0.3,random_state=15)


# Calculating error for various K values to chose the optimum Kvalue
error = []
for i in range(3, 10):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))

plt.rc('axes', labelsize=18)
plt.figure(figsize=(12, 6))
plt.plot(range(3, 10), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value(triplet loss)')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
plt.savefig('img9.png')

#Build Knn with optimum K value and print the classification report
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=4)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

from sklearn.metrics import accuracy_score

accuracy_score(y_test, y_pred)
```

Figure 18: Knn for Triplet loss

4 Conclusion

In this configuration manual given software setup, system setup and code for the implementation of project is briefly explained. This information will help for better insights on the implementation of "OpenPose base Gait Recognition using Triplet Loss Architecture".

References

[1] https://github.com/AdrianUng/keras-triplet-loss-mnist