National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Abhilash Rajkumar Kadhane
Student ID: x18203744

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Abhilash Rajkumar Kadhane
Student ID:	x18203744
Programme:	MSc in Data Analytics **Year:** 2019-20
Module:	MSc Research Project
Supervisor:	Dr. Catherine Mulwa
Submission Due Date:	……
Project Title:	Face Identification and Face Verification in Spherical Images
Word Count:	…… **Page Count**……

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: ……

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

Abhilash Rajkumar Kadhane
X18203744

1 Introduction

This configuration manual includes environment setup, system configuration :– hardware and software specifications and implementation steps for the tasks : a) data generation b) data pre-processing c) model fine-tuning d) model implementation and evaluation performed for the research project : Face identification and face verification in spherical images.

2 System Configuration

2.1 Hardware

- Machine model : Dell G3
- Processor : Intel(R) Core(TM) i7-8750H CPU @ 2.20 GHz 2.21GHz
- RAM : 8 GB RAM
- Graphics : NVIDIA GeForce GTX 1050 Ti

2.2 Software

- PyCharm COMMUNITY 2019.2
- Anacoda3 prompt
- Jupyter (Anaconda3)
- Google Colaboratory

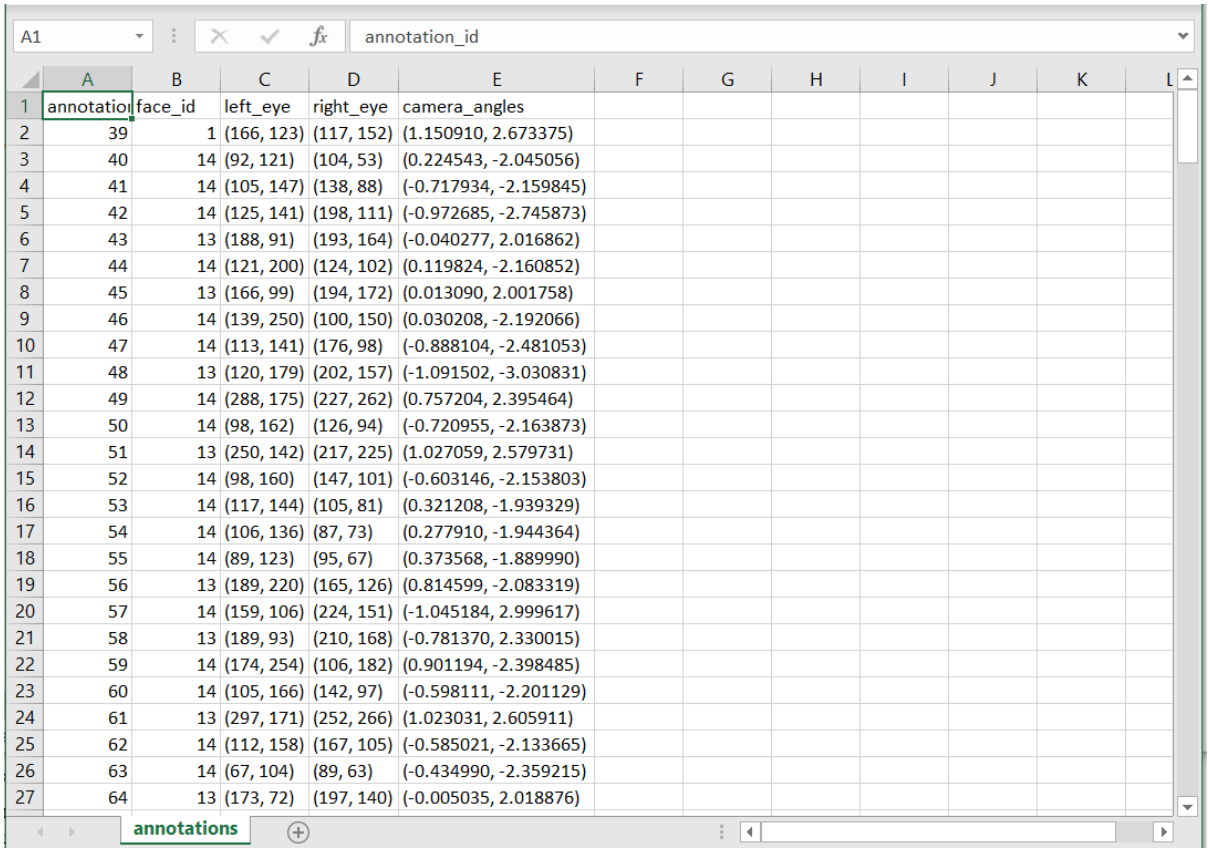
3 Project Development

This project had three major parts: a) Data creation b) model fine-tuning c) Data pre-processing and Model implementation (feature extraction) and classification. Details of these parts are discussed below

3.1 Data creation

Data generation from downloaded VGGFace2[1] dataset using annotations given with MOT-360 dataset[2]

[1] http://www.robots.ox.ac.uk/~vgg/data/vgg_face2/
[2] https://figshare.com/articles/MOT-360_Face/8120747



	A	B	C	D	E
1	annotatio	face_id	left_eye	right_eye	camera_angles
2	39	1	(166, 123)	(117, 152)	(1.150910, 2.673375)
3	40	14	(92, 121)	(104, 53)	(0.224543, -2.045056)
4	41	14	(105, 147)	(138, 88)	(-0.717934, -2.159845)
5	42	14	(125, 141)	(198, 111)	(-0.972685, -2.745873)
6	43	13	(188, 91)	(193, 164)	(-0.040277, 2.016862)
7	44	14	(121, 200)	(124, 102)	(0.119824, -2.160852)
8	45	13	(166, 99)	(194, 172)	(0.013090, 2.001758)
9	46	14	(139, 250)	(100, 150)	(0.030208, -2.192066)
10	47	14	(113, 141)	(176, 98)	(-0.888104, -2.481053)
11	48	13	(120, 179)	(202, 157)	(-1.091502, -3.030831)
12	49	14	(288, 175)	(227, 262)	(0.757204, 2.395464)
13	50	14	(98, 162)	(126, 94)	(-0.720955, -2.163873)
14	51	13	(250, 142)	(217, 225)	(1.027059, 2.579731)
15	52	14	(98, 160)	(147, 101)	(-0.603146, -2.153803)
16	53	14	(117, 144)	(105, 81)	(0.321208, -1.939329)
17	54	14	(106, 136)	(87, 73)	(0.277910, -1.944364)
18	55	14	(89, 123)	(95, 67)	(0.373568, -1.889990)
19	56	13	(189, 220)	(165, 126)	(0.814599, -2.083319)
20	57	14	(159, 106)	(224, 151)	(-1.045184, 2.999617)
21	58	13	(189, 93)	(210, 168)	(-0.781370, 2.330015)
22	59	14	(174, 254)	(106, 182)	(0.901194, -2.398485)
23	60	14	(105, 166)	(142, 97)	(-0.598111, -2.201129)
24	61	13	(297, 171)	(252, 266)	(1.023031, 2.605911)
25	62	14	(112, 158)	(167, 105)	(-0.585021, -2.133665)
26	63	14	(67, 104)	(89, 63)	(-0.434990, -2.359215)
27	64	13	(173, 72)	(197, 140)	(-0.005035, 2.018876)

Figure 1: annotations and metadata of MOT-360 dataset

3.1.1 Environment setup

Platforms used for data generation : Ubuntu (Linux Bash Shell on Windows 10)

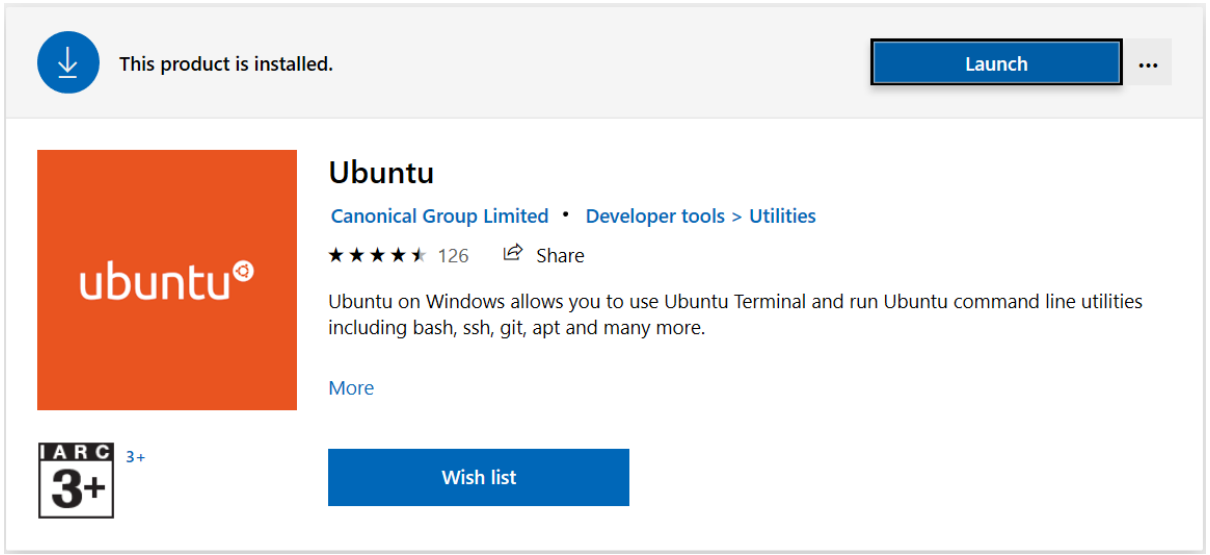


Figure 2: Linux Bash Shell on Windows 10

Linux Bash shell for convenience and running data generation scripts on windows.
Install python dependencies for data generation script: dlib, numpy, panda, scikit-image

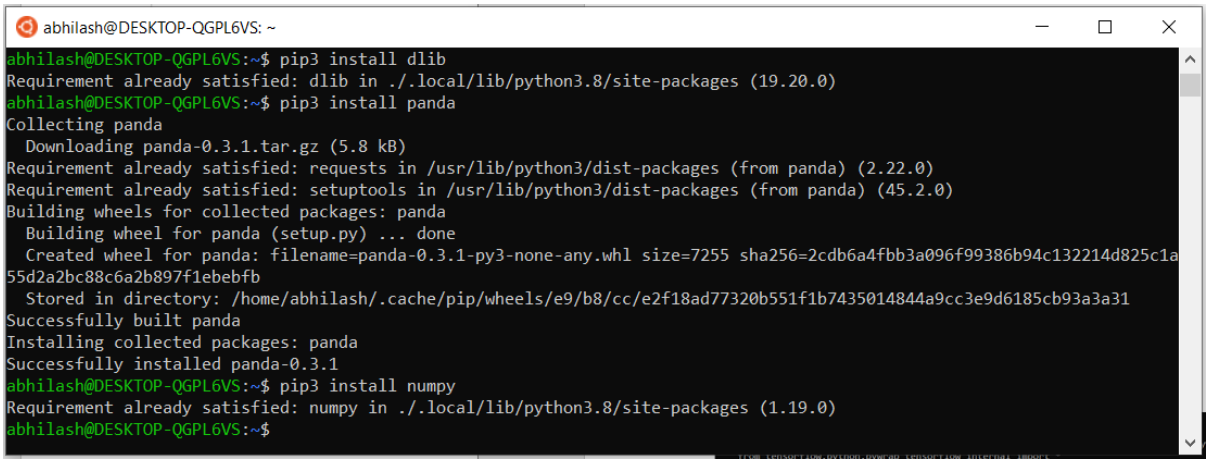


Figure 3: data generation script dependencies

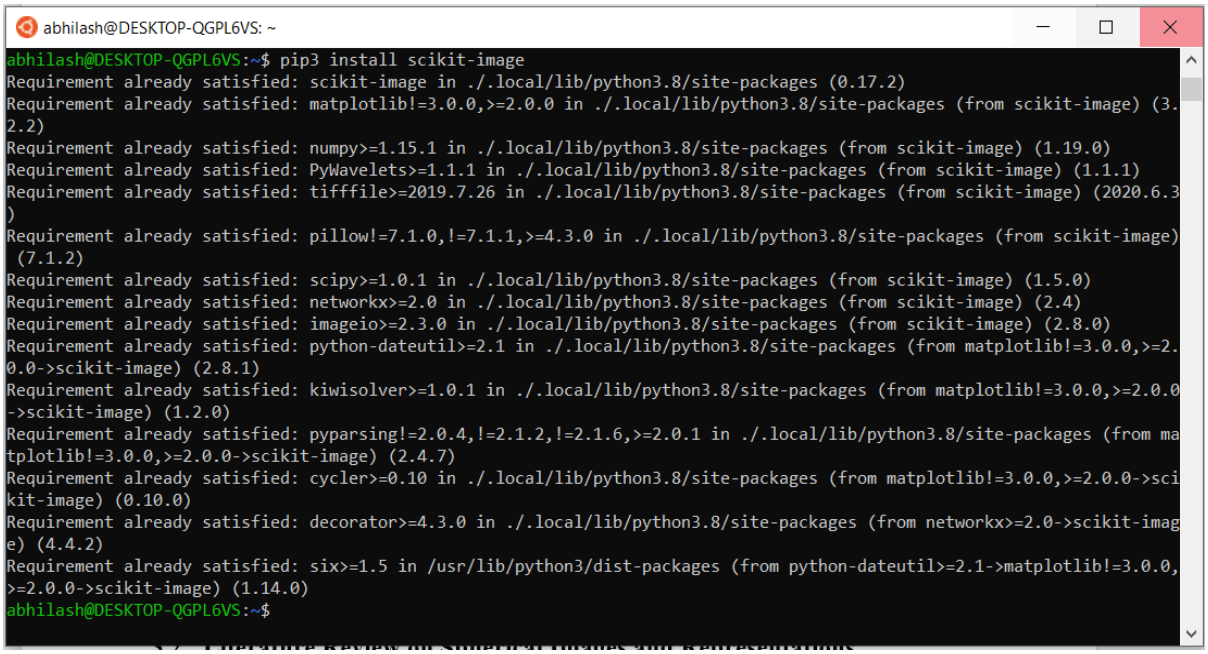


Figure 4: data generation script dependencies

3.1.2 Data generation Code Structure

Code structure of Data generation scripts includes scripts 1) to generate landmarks 2) to project image on sphere 3) to get equirectangular representation of projected image

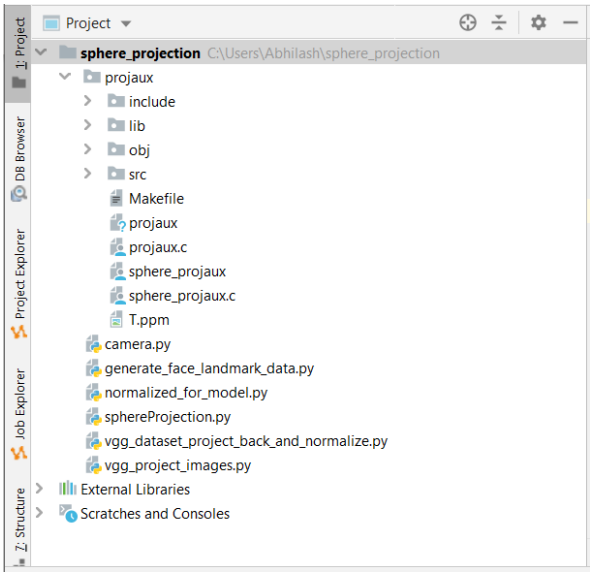


Figure 5: data generation code structure

3.1.3 Data generation using scripts

1) Execute generate_face_landmark_data.py to create metadata containing image details (path, file name, landmarks)
    a. Set path of predictor 'shape_predictor_68_face_landmarks.dat'[3] in the script
    b. Set path of dataset in the script
    c. Set path of output metadata file path in the script

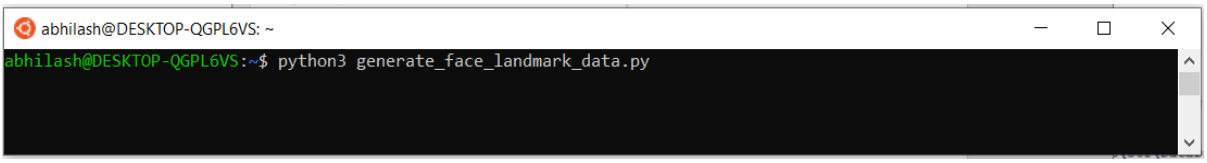


Figure 6: Generate face landmarks

2) Execute vgg_project_images.py to generate equirectangular version of VGGFace2 dataset
    a. Set metadata file path created using generate_face_landmark.py
    b. Set output directory name to save generated data

[3] https://drive.google.com/file/d/16Gzll1q2yp5JBoGGPhzaOpUc6FnQjTUh/view?usp=sharing

```
Running for image 278451 / 278461
Running for image 278452 / 278461
Running for image 278453 / 278461
Running for image 278454 / 278461
Running for image 278455 / 278461
Running for image 278456 / 278461
Running for image 278457 / 278461
Running for image 278458 / 278461
Running for image 278459 / 278461
Running for image 278460 / 278461
Running for image 278461 / 278461
Start time : 2020-07-06 17:06:21.306594
End time : 2020-07-08 21:27:49.940833
abhilash@DESKTOP-QGPL6VS:/mnt/c/Users/Abhilash/sphere_projection$
```

Figure 7: data generation detail

3.2 Model Fine-tuning

ResNet-50 pre-trained model was selected for fine-tuning on Google Colab.
Dataset uploaded in batches and pre-processed using script dataset_operations.ipynb

1) Split dataset into 80% training and 20% testing

```
import os
import shutil

source='drive/My Drive/VGG2_equi_training_data'

for (dirpath, dirnames, filenames) in os.walk(source):
    for dir in dirnames:
        if not os.path.exists('drive/My Drive/VGG2_equi_testing_data/%s' % (dir)):
            os.makedirs('drive/My Drive/VGG2_equi_testing_data/%s' % (dir))
        files = os.listdir('drive/My Drive/VGG2_equi_training_data/%s' % (dir))
        for k, f in enumerate(files):
            if not k % 20:
                shutil.move('drive/My Drive/VGG2_equi_training_data/%s/' % (dir) + f, 'drive/My Drive/VGG2_equi_testing_data/%s' % (dir))
```

Figure 8: Split dataset train and test

2) Remove classes with low count of images (handling class imbalance)

```
[ ] import os
    import shutil

    labels = []
    source='drive/My Drive/VGG2_equi_training_data'
    for (dirpath, dirnames, filenames) in os.walk(source):
        for dir in dirnames:
          labels.append(dir)
          dir_path = os.path.join(dirpath, dir)
          file_count = len([name for name in os.listdir(dir_path) if os.path.isfile(os.path.join(dir_path, name))])
          if file_count < 45:
            print('{0}>>>>{1}'.format(dir_path,file_count))
            # shutil.rmtree(dir_path)
```

```
print(len(labels))
```

```
1067
```

```
[ ] source='drive/My Drive/VGG2_equi_testing_data'
    for (dirpath, dirnames, filenames) in os.walk(source):
        for dir in dirnames:
          if dir not in labels:
            dir_path = os.path.join(dirpath, dir)
            print(dir_path)
            # shutil.rmtree(dir_path)
```

Figure 9: Remove classes with very few samples

3) Create metadata and pre-processing functionality (includes face detection, face crop, grey scaling, image resizing)

```
[ ]  def load_image(path):
         img = cv2.imread(path, 1)
         # OpenCV loads images with color channels
         # in BGR order. So we need to reverse them
         return img[...,::-1]

     # Initialize the OpenFace face alignment utility for image pre-processing
     alignment = AlignDlib('drive/My Drive/shape_predictor_68_face_landmarks.dat')

     def align_image(img):
         return alignment.align(224, img, alignment.getLargestFaceBoundingBox(img),
                                landmarkIndices=AlignDlib.OUTER_EYES_AND_NOSE)


     class IdentityMetadata():
         def __init__(self, base, name, file):
             # dataset base directory
             self.base = base
             # identity name
             self.name = name
             # image file name
             self.file = file

         def __repr__(self):
             return self.image_path()

         def image_path(self):
             return os.path.join(self.base, self.name, self.file)

     def load_metadata(path):
         metadata = []
         for i in sorted(os.listdir(path)):
             for f in sorted(os.listdir(os.path.join(path, i))):
                 # Check file extension. Allow only jpg/jpeg' files.
                 ext = os.path.splitext(f)[1]
                 if ext == '.jpg' or ext == '.jpeg' or ext == '.png':
                     metadata.append(IdentityMetadata(path, i, f))
         return np.array(metadata)

     metadata = load_metadata('drive/My Drive/VGG2_equi_training_data_2')
```

Figure 10: create metadata and define pre-processing function

4) Pre-process dataset and create pre-processed images dataset separately, remove defected samples

```
[ ]  for i, m in enumerate(metadata):
         img = load_image(m.image_path())
         img = align_image(img)
         if img is None:
             pass
         else:
             face_pixels = img.astype('float32')
             if not os.path.exists('drive/My Drive/cropped_VGG2_equi_training_data_3/%s' % (m.name)):
                 os.makedirs('drive/My Drive/cropped_VGG2_equi_training_data_3/%s' % (m.name))
             cv2.imwrite('drive/My Drive/cropped_VGG2_equi_training_data_3/{0}/{1}.png'.format(m.name, m.file), img)
```

```
# Code to check 0 size files
import os
import shutil

source='drive/My Drive/cropped_VGG2_equi_training_data'
for (dirpath, dirnames, filenames) in os.walk(source):
    for dir in dirnames:
      dir_path = os.path.join(dirpath, dir)
      for name in os.listdir(dir_path):
        if os.path.getsize(os.path.join(dir_path, name)) == 0:
          print(os.path.join(dir_path, name))
```

```
drive/My Drive/cropped_VGG2_equi_training_data/n000056/0152_01.jpg.png
```

Figure 11: pre-process dataset and remove defected samples

5) Check TPU availability and tensorflow library

```
[ ] import tensorflow as tf
    print(tf.__version__)

    import os
    try:
      device_name = os.environ['COLAB_TPU_ADDR']
      TPU_ADDRESS = 'grpc://' + device_name
      print('Found TPU at: {}'.format(TPU_ADDRESS))

    except KeyError:
      print('TPU not found')
```

```
2.3.0
Found TPU at: grpc://10.26.157.98:8470
```

Figure 12: Check TPU availability

6) Install and import required libraries

```
[ ] !pip install keras_vggface
    !pip install keras_applications
    import numpy as np
    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras.models import Sequential,Model
    from tensorflow.keras.layers import ZeroPadding2D,Convolution2D,MaxPooling2D,GlobalAveragePooling2D
    from tensorflow.keras.layers import Dense,Dropout,Softmax,Flatten,Activation,BatchNormalization
    from tensorflow.keras.preprocessing.image import load_img,img_to_array
    from tensorflow.keras.applications.imagenet_utils import preprocess_input
    import tensorflow.keras.backend as K
    from keras_vggface.vggface import VGGFace
```

```
Collecting keras_vggface
  Downloading https://files.pythonhosted.org/packages/2f/7d/5f0319ebdc09ac1a2272364fa9583f5067b6f8aff93fbbf8835d81cbaad7/keras_vggface-0.6-py3-none-any.whl
Requirement already satisfied: scipy>=0.14 in /usr/local/lib/python3.6/dist-packages (from keras_vggface) (1.4.1)
Requirement already satisfied: h5py in /usr/local/lib/python3.6/dist-packages (from keras_vggface) (2.10.0)
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.6/dist-packages (from keras_vggface) (1.18.5)
Requirement already satisfied: pillow in /usr/local/lib/python3.6/dist-packages (from keras_vggface) (7.0.0)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.6/dist-packages (from keras_vggface) (1.15.0)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.6/dist-packages (from keras_vggface) (3.13)
Requirement already satisfied: keras in /usr/local/lib/python3.6/dist-packages (from keras_vggface) (2.4.3)
Installing collected packages: keras-vggface
Successfully installed keras-vggface-0.6
Collecting keras_applications
  Downloading https://files.pythonhosted.org/packages/71/e3/19762fdfc62877ae9102edf6342d71b28fbfd9dea3d2f96a882ce099b03f/Keras_Applications-1.0.8-py3-none-a
     |████████████████████████████████| 51kB 1.8MB/s
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.6/dist-packages (from keras_applications) (1.18.5)
Requirement already satisfied: h5py in /usr/local/lib/python3.6/dist-packages (from keras_applications) (2.10.0)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from h5py->keras_applications) (1.15.0)
Installing collected packages: keras-applications
Successfully installed keras-applications-1.0.8
```

Figure 13: Install and import required libraries

7) Import ResNet-50 model and modify

```
[ ] # create a vggface2 model
    model = VGGFace(model='resnet50')
```

```
Downloading data from https://github.com/rcmalli/keras-vggface/releases/download/v2.0/rcmalli_vggface_tf_resnet50.h5
165445632/165439116 [==============================] - 2s 0us/step
```

```
[ ] model.summary()
```

```
num_classes = 194
for layer in model.layers:
  layer.trainable = False

def new_model(bottom_model, num_classes):

  top_model = bottom_model.output
  top_model = Dense(1024, activation='relu')(top_model)
  top_model = Dense(1024, activation='relu')(top_model)
  top_model = Dense(512, activation='relu')(top_model)
  top_model = Dense(num_classes, activation='softmax')(top_model)
  return top_model

vgg_face=Model(inputs=model.layers[0].input,outputs=model.layers[-2].output)
face_model = new_model(vgg_face, num_classes)
vgg_face = Model(inputs = vgg_face.input, outputs = face_model)
vgg_face.summary()
```

Figure 14: Import ResNet-50 model and modify

8) Datagenerator to for training data and testing data

```
[ ] from keras.preprocessing.image import ImageDataGenerator
    batch_size = 64
    nrow,ncol = 224, 224

    train_data_dir = 'drive/My Drive/cropped_VGG2_equi_training_data'
    train_datagen = ImageDataGenerator()
    train_generator = train_datagen.flow_from_directory(
                          train_data_dir,
                          target_size=(nrow,ncol),
                          batch_size=batch_size,
                          class_mode='categorical')
```

```
Found 13416 images belonging to 194 classes.
```

```
[ ] test_data_dir = 'drive/My Drive/cropped_VGG2_equi_testing_data'
    test_datagen = ImageDataGenerator()
    test_generator = test_datagen.flow_from_directory(
                          test_data_dir,
                          target_size=(nrow,ncol),
                          batch_size=batch_size,
                          class_mode='categorical')
```

```
Found 802 images belonging to 194 classes.
```

Figure 15: image data generator for train and test data

9) Compile model, set hyperparameters and callback

```
[ ] from keras.optimizers import SGD, RMSprop
    from keras.callbacks import ModelCheckpoint, EarlyStopping

    checkpoint = ModelCheckpoint("drive/My Drive/VGG_Equi_Face.h5",
                                 monitor="val_loss",
                                 mode="min",
                                 save_best_only = True,
                                 verbose=1)

    earlystop = EarlyStopping(monitor='val_loss',
                              min_delta = 0,
                              patience = 3,
                              verbose = 1,
                              restore_best_weights = True)

    callbacks = [earlystop, checkpoint]

    vgg_face.compile(loss = 'categorical_crossentropy',
                            optimizer = RMSprop(lr=1e-4, momentum=0.9),
                            experimental_steps_per_execution = 50,
                            metrics = ['accuracy'])

    steps_per_epoch =  train_generator.n // batch_size
    validation_steps =  test_generator.n // batch_size

    nepochs = 5
```

Figure 16: compile model and set required parameters

10) Train model by calling fit_generator

```
[ ] vgg_face.fit_generator(
        train_generator,
        steps_per_epoch=steps_per_epoch,
        epochs=nepochs,
        callbacks=callbacks,
        validation_data=test_generator,
        validation_steps=validation_steps)
```

```
poch 1/5
09/209 [==============================] - ETA: 0s - loss: 1.6985 - accuracy: 0.6344
poch 00001: val_loss improved from inf to 1.02069, saving model to drive/My Drive/VGG_Equi_Face.h5
09/209 [==============================] - 1750s 8s/step - loss: 1.6985 - accuracy: 0.6344 - val_loss: 1.0207 - val_accuracy: 0.7695
poch 2/5
09/209 [==============================] - ETA: 0s - loss: 0.6798 - accuracy: 0.8307
poch 00002: val_loss improved from 1.02069 to 0.74840, saving model to drive/My Drive/VGG_Equi_Face.h5
09/209 [==============================] - 1505s 7s/step - loss: 0.6798 - accuracy: 0.8307 - val_loss: 0.7484 - val_accuracy: 0.8281
```

Figure 17: train model

11) Save model and weights to load and fine-tune next batch of data

```
[ ] vgg_face.save_weights('drive/My Drive/VGG_Face_weights_1/VGG_Face_weights_1')
```

```
[ ] vgg_face.save('drive/My Drive/VGG_Face_fine_tuned_1/VGG_Face_fine_tuned_1')
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/training/tracking/tracking.py:111: Model.state_updates (fr
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied automatically.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/training/tracking/tracking.py:111: Layer.updates (from ter
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied automatically.
INFO:tensorflow:Assets written to: drive/My Drive/VGG_Face_fine_tuned_1/VGG_Face_fine_tuned_1/assets
```

Figure 18: Save model and weights

3.3 Data Pre-processing and Model Implementation

Data pre-processing and model implementation done on Jupyter notebook on local machine

1) Jupyter version details

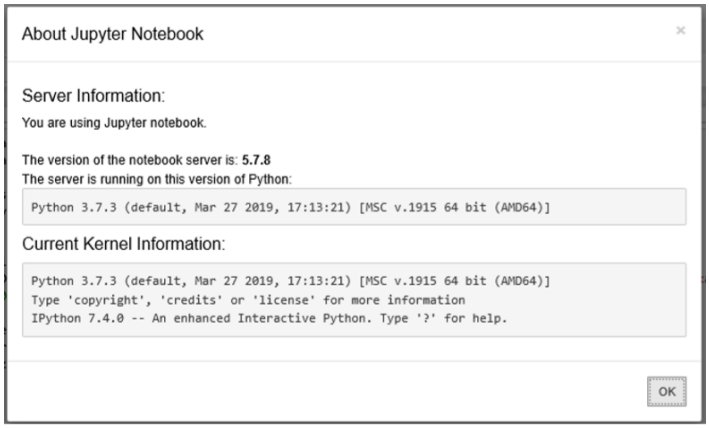


Figure 19: Jupyter details

2) Conda virtual environment setup and dependencies resolution by installing requirement.txt

   a) Create conda virtual environment

   b) Install dlib in activated virtual environment

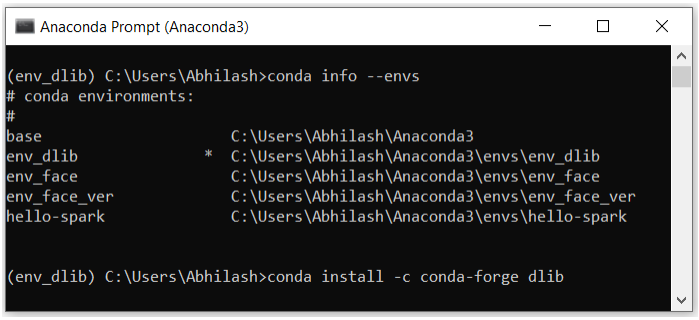


Figure 20: Activate conda virtual env and install dlib

   c) Common dependency issues

      I. Tensorflow version 1.9 issue

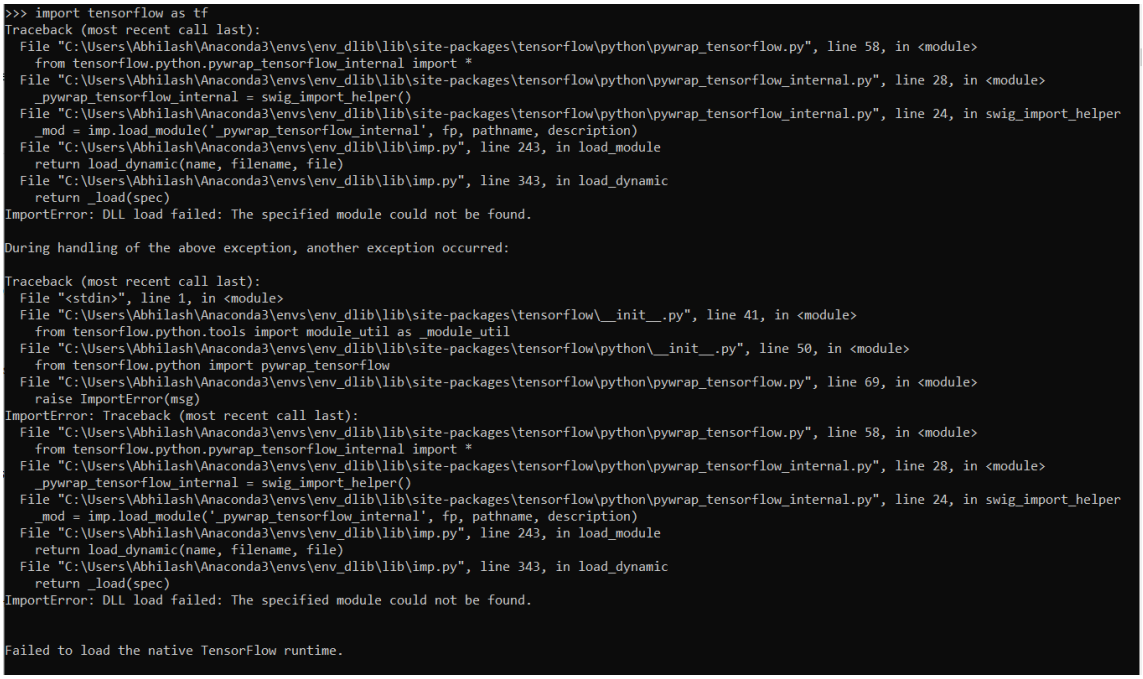


Figure 21: Tensorflow bug

Install Tensorflow version 1.6

Commands:

pip install tensorflow==1.6
pip install --upgrade tornado==5.1.1

II. Numpy issue with conda

Command:

pip install --upgrade --force-reinstall numpy

III. Matplotlib issue

Command

pip install --upgrade matplotlib

3) Implementation of OpenFace model for face recognition

a) Import model and load weights ‘nh4.small2.v1.h5’[4]

```
# create model
from model import create_model
from src.common import *
from scipy.io import loadmat
from sklearn import metrics
from sklearn.decomposition import PCA
from sklearn.externals import joblib
from src.joint_bayesian import *
```

```
# load weights
nn4_small2_pretrained = create_model()
nn4_small2_pretrained.load_weights('weights/nn4.small2.v1.h5')
```

Figure 22: Import OpenFace and load weights

b) Data pre-processing and create metadata

```
[ ]  def load_image(path):
         img = cv2.imread(path, 1)
         # OpenCV loads images with color channels
         # in BGR order. So we need to reverse them
         return img[...,::-1]

     # Initialize the OpenFace face alignment utility for image pre-processing
     alignment = AlignDlib('drive/My Drive/shape_predictor_68_face_landmarks.dat')

     def align_image(img):
         return alignment.align(224, img, alignment.getLargestFaceBoundingBox(img),
                                landmarkIndices=AlignDlib.OUTER_EYES_AND_NOSE)


     class IdentityMetadata():
         def __init__(self, base, name, file):
             # dataset base directory
             self.base = base
             # identity name
             self.name = name
             # image file name
             self.file = file

         def __repr__(self):
             return self.image_path()

         def image_path(self):
             return os.path.join(self.base, self.name, self.file)

     def load_metadata(path):
         metadata = []
         for i in sorted(os.listdir(path)):
             for f in sorted(os.listdir(os.path.join(path, i))):
                 # Check file extension. Allow only jpg/jpeg' files.
                 ext = os.path.splitext(f)[1]
                 if ext == '.jpg' or ext == '.jpeg' or ext == '.png':
                     metadata.append(IdentityMetadata(path, i, f))
         return np.array(metadata)

     metadata = load_metadata('drive/My Drive/VGG2_equi_training_data_2')
```

Figure 23: Data pre-processing

[4] https://drive.google.com/file/d/1_ULIEGsMSf_fSp9TJQDinKlf-1vgU5r6/view?usp=sharing

c) Extract feature using OpenFace and save in a binary format file for later usage

```
embedded = []
metadata_curated = []

for i, m in enumerate(metadata):
    img = load_image(m.image_path())
    img = align_image(img)
    if img is None:
        pass
    else:
        # scale RGB values to interval [0,1]
        img = (img / 255.).astype(np.float32)
        metadata_curated.append(IdentityMetadata(m.base, m.name, m.file))
        # obtain embedding vector for image
        embedded.append(nn4_small2_pretrained.predict(np.expand_dims(img, axis=0))[0])
metadata_curated = np.array(metadata_curated)
embedded = np.array(embedded)
```

```
data_to_pkl(embedded, "OpenFace_MOT360_fetures.pkl")
```

```
Saving data to file(OpenFace_MOT360_fetures.pkl).

True
```

Figure 24: Feature extraction using OpenFace

d) Analysis of extracted features : identity cluster, distance threshold and distance distribution

```
from sklearn.manifold import TSNE

targets = np.array([m.name for m in metadata_curated])
X_embedded = TSNE(n_components=2).fit_transform(embedded)

for i, t in enumerate(set(targets)):
    idx = targets == t
    plt.scatter(X_embedded[idx, 0], X_embedded[idx, 1], label=t)

plt.legend(bbox_to_anchor=(1, 1));
```

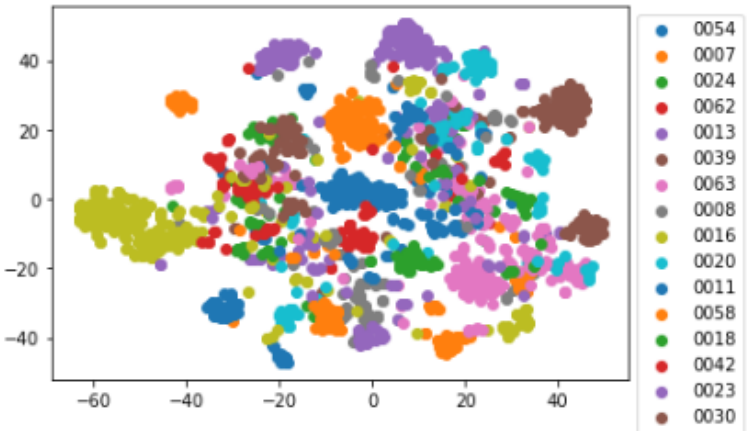


Figure 25: identity clusters

```python
from sklearn.metrics import f1_score, accuracy_score

def distance(emb1, emb2):
    return np.sum(np.square(emb1 - emb2))

distances = [] # squared L2 distance between pairs
identical = [] # 1 if same identity, 0 otherwise

num = len(metadata_curated)

for i in range(num - 1):
    for j in range(i + 1, num):
        distances.append(distance(embedded[i], embedded[j]))
        identical.append(1 if metadata_curated[i].name == metadata_curated[j].name else 0)

distances = np.array(distances)
identical = np.array(identical)

thresholds = np.arange(0.3, 1.0, 0.01)

f1_scores = [f1_score(identical, distances < t) for t in thresholds]
acc_scores = [accuracy_score(identical, distances < t) for t in thresholds]

opt_idx = np.argmax(f1_scores)
# Threshold at maximal F1 score
opt_tau = thresholds[opt_idx]
# Accuracy at maximal F1 score
opt_acc = accuracy_score(identical, distances < opt_tau)

# Plot F1 score and accuracy as function of distance threshold
plt.plot(thresholds, f1_scores, label='F1 score');
plt.plot(thresholds, acc_scores, label='Accuracy');
plt.axvline(x=opt_tau, linestyle='--', lw=1, c='lightgrey', label='Threshold')
plt.title(f'Accuracy at threshold {opt_tau:.2f} = {opt_acc:.3f}');
plt.xlabel('Distance threshold')
plt.legend();
```

Figure 26: Distance threshold calculation

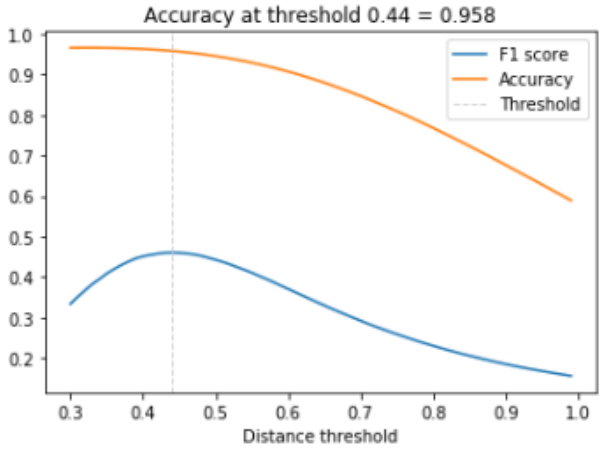


Figure 27: Distance threshold graph

```
dist_pos = distances[identical == 1]
dist_neg = distances[identical == 0]

plt.figure(figsize=(12,4))

plt.subplot(121)
plt.hist(dist_pos)
plt.axvline(x=opt_tau, linestyle='--', lw=1, c='lightgrey', label='Threshold')
plt.title('Distances (pos. pairs)')
plt.legend();

plt.subplot(122)
plt.hist(dist_neg)
plt.axvline(x=opt_tau, linestyle='--', lw=1, c='lightgrey', label='Threshold')
plt.title('Distances (neg. pairs)')
plt.legend();
```

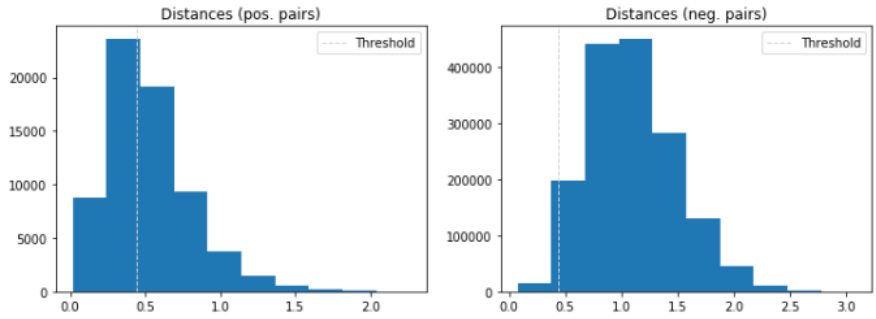


Figure 28: Distance distribution

e) Classification using KNN and SVM

```
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC

targets = np.array([m.name for m in metadata_curated])

encoder = LabelEncoder()
encoder.fit(targets)

# Numerical encoding of identities
y = encoder.transform(targets)

train_idx = np.random.randint(y.shape[0], size=int(len(y)*0.8))
test_idx = np.random.randint(y.shape[0], size=int(len(y)*0.2))

y_train = y[train_idx]
y_test = y[test_idx]

X_train = embedded[train_idx]
X_test = embedded[test_idx]

knn = KNeighborsClassifier(n_neighbors=1, metric='euclidean')
svc = LinearSVC()

knn.fit(X_train, y_train)
svc.fit(X_train, y_train)

acc_knn = accuracy_score(y_test, knn.predict(X_test))
acc_svc = accuracy_score(y_test, svc.predict(X_test))

print(f'KNN accuracy = {acc_knn}, SVM accuracy = {acc_svc}')
```

```
KNN accuracy = 0.8922651933701657, SVM accuracy = 0.8535911602209945
```

Figure 29: Face identification using classifiers

f) Classification using Joint Bayesian

Split dataset for training and testing

```
targets = np.array([m.name for m in metadata_curated])

encoder = LabelEncoder()
encoder.fit(targets)

# Numerical encoding of identities
y = encoder.transform(targets)

train_idx = np.random.randint(y.shape[0], size=int(len(y)*0.7))
test_idx = np.random.randint(y.shape[0], size=int(len(y)*0.3))

y_train = y[train_idx]
y_test = y[test_idx]

X_train = embedded[train_idx]
X_test = embedded[test_idx]
```

Figure 30: Split dataset train and test

Create positive and negative pairs of face images for face verification

```
test_pairs = y_test.tolist()
IntraPersonPair = []
ExtraPersonPair = []
for i in range(len(test_pairs)):
    for j in range(i+1, len(test_pairs)):
        if test_pairs[i] == test_pairs[j]:
            IntraPersonPair.append([i,j])
        else:
            ExtraPersonPair.append([i,j])
```

```
print(len(IntraPersonPair))
print(len(ExtraPersonPair))
IntraPersonPair = IntraPersonPair[:6000]
ExtraPersonPair = ExtraPersonPair[:6000]
```

```
6291
141405
```

Figure 31: Create pairs for face verification

Train Joint Bayesian classifier for face verification

```
def excute_train(data, label):

    # pca training.
    pca = PCA_Train(data, 128)
    data_pca = pca.transform(data)

    JointBayesian_Train(data_pca, label)
```

```
excute_train(X_train, y_train)
```

```
2020-08-15, 04:34:01  PCA done.
(1269, 128)
prepare done, maxNumberInOneClass= 143
2020-08-15, 04:34:02  Iterations-0: 0.0015399922708238208
Saving data to file(./G.pkl).
Saving data to file(./A.pkl).
2020-08-15, 04:34:03  Iterations-1: 0.001373748295804448
Saving data to file(./G.pkl).
Saving data to file(./A.pkl).
2020-08-15, 04:34:03  Iterations-2: 0.0012750262843178207
```

Figure 32: Train Joint Bayesian on train data

Test Joint Bayesian with face image pairs and test data

```
with open("A.pkl", "rb") as f:
    A = pickle.load(f)
with open("G.pkl", "rb") as f:
    G = pickle.load(f)
clt_pca = joblib.load("pca_model.m")
data = clt_pca.transform(X_test)
data_to_pkl(data, "pca_lfw.pkl")
```

```
Saving data to file(pca_lfw.pkl).

True
```

```
dist_Intra = get_ratios(A, G, IntraPersonPair, data)
dist_Extra = get_ratios(A, G, ExtraPersonPair, data)

dist_all = dist_Intra + dist_Extra
dist_all = np.asarray(dist_all)
label    = np.append(np.repeat(1, len(dist_Intra)), np.repeat(0, len(dist_Extra)))

data_to_pkl({"distance": dist_all, "label": label}, "result.pkl")
```

```
Saving data to file(result.pkl).

True
```

Figure 33: Test Joint Bayesian classifier

Get the results of face verification

```
excute_performance("result.pkl", -16.9, -16.6, 0.01)
```

```
avg / total       0.87      0.84      0.84     12000

threshold:  -16.619999999999955
              precision    recall  f1-score   support

       False       0.77      0.98      0.86      6000
        True       0.97      0.71      0.82      6000

avg / total       0.87      0.84      0.84     12000

threshold:  -16.609999999999953
              precision    recall  f1-score   support

       False       0.77      0.98      0.86      6000
        True       0.97      0.71      0.82      6000

avg / total       0.87      0.84      0.84     12000
```

Figure 34: Results face verification

4) Implementation of ResNet-50 model for face recognition

a) Import model and load weights (weights get loaded at the time of creating model)

```
#coding=utf-8
import numpy as np
import os.path
import cv2
import keras_vggface
from matplotlib import pyplot
from PIL import Image
from align import AlignDlib
from numpy import expand_dims
from keras_vggface.utils import preprocess_input
from sklearn.preprocessing import LabelEncoder
from keras_vggface.vggface import VGGFace
import sys
from src.common import *
from scipy.io import loadmat
from sklearn import metrics
from sklearn.decomposition import PCA
from sklearn.externals import joblib
from src.joint_bayesian import *
from sklearn.metrics import f1_score, accuracy_score
```

```
# create a vggface2 model
model = VGGFace(model='resnet50')
```

Figure 35: Import ResNet-50

b) Data pre-processing and create metadata
Same as OpenFace model (section 3.b)

c) Extract feature using ResNet-50 and save in a binary format file

```
embedded = np.empty(8631)
metadata_curated = []

for i, m in enumerate(metadata):
    img = load_image(m.image_path())
    img = align_image(img)
    if img is None:
        pass
    else:
        img = img.astype('float32')
        samples = expand_dims(img, axis=0)
        # prepare the face for the model, e.g. center pixels
        samples = preprocess_input(samples, version=2)
        metadata_curated.append(IdentityMetadata(m.base, m.name, m.file))
        embedded = np.vstack([embedded , model.predict(samples)])
        embedding = model.predict(samples)
metadata_curated = np.array(metadata_curated)
embed = np.delete(embedded, 0, 0)
```

```
data_to_pkl(embed, "ResNet-50_MOT360_fetures.pkl")
```

Figure 36: Feature extraction using ResNet-50

d) Analysis of extracted features : identity cluster

```
from sklearn.manifold import TSNE

targets = np.array([m.name for m in metadata_curated])
X_embedded = TSNE(n_components=2).fit_transform(embed)

for i, t in enumerate(set(targets)):
    idx = targets == t
    pyplot.scatter(X_embedded[idx, 0], X_embedded[idx, 1], label=t)

pyplot.legend(bbox_to_anchor=(1, 1));
```

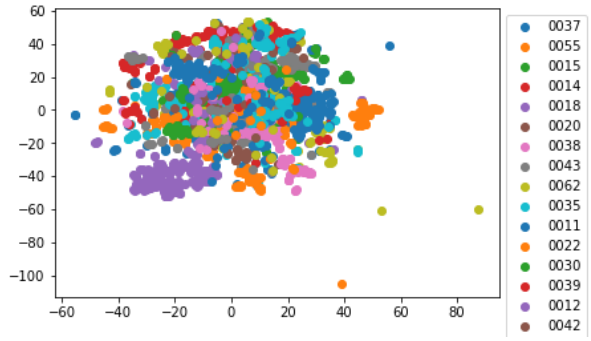


Figure 37: Identity cluster on ResNet-50

e) Classification using KNN and SVM

```
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC

targets = np.array([m.name for m in metadata_curated])

encoder = LabelEncoder()
encoder.fit(targets)

# Numerical encoding of identities
y = encoder.transform(targets)

with open("data/train_idx_identification.pkl", "rb") as f:
    train_idx = pickle.load(f)

with open("data/test_idx_identification.pkl", "rb") as f:
    test_idx = pickle.load(f)

y_train = y[train_idx]
y_test = y[test_idx]

X_train = embed[train_idx]
X_test = embed[test_idx]

knn = KNeighborsClassifier(n_neighbors=1, metric='euclidean')
svc = LinearSVC()

knn.fit(X_train, y_train)
svc.fit(X_train, y_train)

acc_knn = accuracy_score(y_test, knn.predict(X_test))
acc_svc = accuracy_score(y_test, svc.predict(X_test))

print(f'KNN accuracy = {acc_knn}, SVM accuracy = {acc_svc}')
```

```
KNN accuracy = 0.7928176795580111, SVM accuracy = 0.6353591160220995
```

Figure 38: Face identification using ResNet-50

f) Classification using Joint Bayesian
Steps for performing classification using Joint Bayesian are same as Steps in section 3.f
Results of face verification using ResNet-50

```
In [80]: excute_performance("result.pkl", -16.9, -16.6, 0.01)

avg / total       0.77      0.71      0.70     12000

threshold:  -16.619999999999955
             precision    recall  f1-score   support

      False       0.89      0.48      0.63      6000
       True       0.64      0.94      0.77      6000

avg / total       0.77      0.71      0.70     12000

threshold:  -16.609999999999953
             precision    recall  f1-score   support

      False       0.89      0.48      0.63      6000
       True       0.64      0.94      0.77      6000

avg / total       0.77      0.71      0.70     12000
```

Figure 39: FaceNet Face verification results

5) Implementation of FaceNet model for face recognition

a) Import model and load weights ‘facenet_keras.h5’[5]

```
#coding=utf-8
import sys
import numpy as np
from src.common import *
from scipy.io import loadmat
from sklearn import metrics
from sklearn.decomposition import PCA
from sklearn.externals import joblib
from src.joint_bayesian import *
import cv2
import os.path
from PIL import Image
from align import AlignDlib
from numpy import expand_dims
from matplotlib import pyplot
from keras.models import load_model
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import f1_score, accuracy_score
```

```
# load the model
Facenet_model = load_model('models/facenet_keras.h5')
# summarize input and output shape
print(Facenet_model.inputs)
print(Facenet_model.outputs)
```

```
[<tf.Tensor 'input_1_1:0' shape=(?, 160, 160, 3) dtype=float32>]
[<tf.Tensor 'Bottleneck_BatchNorm_1/cond/Merge:0' shape=(?, 128) dtype=float32>]
```

Figure 40: Import required libraries and model

b) Data pre-processing and create metadata
Same as OpenFace model (section 3.b)

c) Extract feature using ResNet-50 and save in a binary format file

```
embedded = []
metadata_curated = []

for i, m in enumerate(metadata):
    img = load_image(m.image_path())
    img = align_image(img)
    if img is None:
        pass
    else:
        # scale pixel values
        face_pixels = img.astype('float32')
        # standardize pixel values across channels (global)
        mean, std = face_pixels.mean(), face_pixels.std()
        face_pixels = (face_pixels - mean) / std
        metadata_curated.append(IdentityMetadata(m.base, m.name, m.file))
        # obtain embedding vector for image
        embedded.append(Facenet_model.predict(np.expand_dims(face_pixels, axis=0))[0])
metadata_curated = np.array(metadata_curated)
embedded = np.array(embedded)
```

```
data_to_pkl(embed, "FaceNet-50_MOT360_fetures.pkl")
```

Figure 41: Feature extraction using FaceNet

[5] https://drive.google.com/drive/folders/1pwQ3H4aJ8a6yyJHZkTwtjcL4wYWQb7bn

d) Analysis of extracted features : identity cluster

```
from sklearn.manifold import TSNE

targets = np.array([m.name for m in metadata_curated])
X_embedded = TSNE(n_components=2).fit_transform(embedded)

for i, t in enumerate(set(targets)):
    idx = targets == t
    pyplot.scatter(X_embedded[idx, 0], X_embedded[idx, 1], label=t)

pyplot.legend(bbox_to_anchor=(1, 1));
```

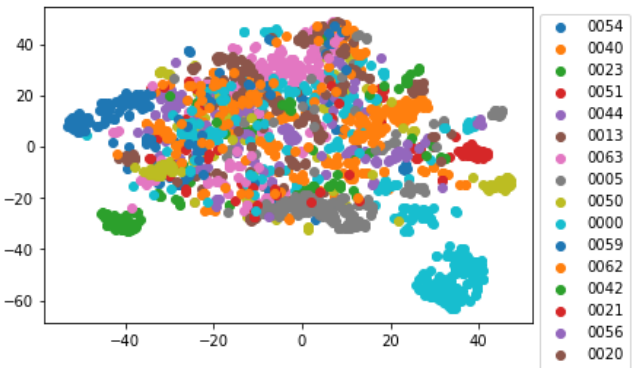


Figure 42: Identity cluster using FaceNet

e) Classification using KNN and SVM

```
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC

targets = np.array([m.name for m in metadata_curated])

encoder = LabelEncoder()
encoder.fit(targets)

# Numerical encoding of identities
y = encoder.transform(targets)

with open("data/train_idx_identification.pkl", "rb") as f:
    train_idx = pickle.load(f)

with open("data/test_idx_identification.pkl", "rb") as f:
    test_idx = pickle.load(f)

y_train = y[train_idx]
y_test = y[test_idx]

X_train = embedded[train_idx]
X_test = embedded[test_idx]

knn = KNeighborsClassifier(n_neighbors=1, metric='euclidean')
svc = LinearSVC()

knn.fit(X_train, y_train)
svc.fit(X_train, y_train)

acc_knn = accuracy_score(y_test, knn.predict(X_test))
acc_svc = accuracy_score(y_test, svc.predict(X_test))

print(f'KNN accuracy = {acc_knn}, SVM accuracy = {acc_svc}')
```

```
KNN accuracy = 0.8453038674033149, SVM accuracy = 0.850828729281768
```

Figure 43: Face identification using FaceNet

f) Classification using Joint Bayesian
Steps for face verification implementation using Joint Bayesian are same as Steps in section 3.f
Results of face verification using FaceNet

```
In [21]: excute_performance("result.pkl", -16.9, -16.6, 0.01)

          avg / total       0.87      0.87      0.87     12000

          threshold:  -16.619999999999955
                       precision    recall  f1-score   support

                False       0.83      0.93      0.88      6000
                 True       0.92      0.81      0.86      6000

          avg / total       0.87      0.87      0.87     12000

          threshold:  -16.609999999999953
                       precision    recall  f1-score   support

                False       0.83      0.93      0.88      6000
                 True       0.92      0.81      0.86      6000

          avg / total       0.87      0.87      0.87     12000
```

Figure 44: Face verification results FaceNet

6) Implementation of fine-tuned ResNet-50 model for face recognition

a) Import all required libraries along with model and load weights vgg_equi_face.h5[6]

```
import numpy as np
import os.path
import cv2
import sys
from src.common import *
from scipy.io import loadmat
from sklearn import metrics
import matplotlib.patches as patches
from sklearn.preprocessing import LabelEncoder
from sklearn.externals import joblib
from src.joint_bayesian import *
import keras_vggface
from matplotlib import pyplot
from PIL import Image
from align import AlignDlib
from numpy import expand_dims
from sklearn.decomposition import PCA
from keras_vggface.vggface import VGGFace
from sklearn.metrics import f1_score, accuracy_score
import tensorflow as tf
from tensorflow import keras as keras
from keras.models import Model
from keras.layers import Dense
from keras_vggface.utils import preprocess_input
```

```
# create a vggface2 model
model = VGGFace(model='resnet50')
```

Figure 45: Import libraries and model

[6] https://drive.google.com/file/d/1swdUPlQHaQ26ERoXYYlRJSEkw5k7yJG3/view?usp=sharing

b) Modify the model as per the architecture of ResNet-50 fine-tuned model

```
num_classes = 194
for layer in model.layers:
  layer.trainable = False

def new_model(bottom_model, num_classes):

  top_model = bottom_model.output
  top_model = Dense(1024, activation='relu')(top_model)
  top_model = Dense(1024, activation='relu')(top_model)
  top_model = Dense(512, activation='relu')(top_model)
  top_model = Dense(num_classes, activation='softmax')(top_model)
  return top_model

vgg_face=Model(inputs=model.layers[0].input,outputs=model.layers[-2].output)
face_model = new_model(vgg_face, num_classes)
vgg_face = Model(inputs = vgg_face.input, outputs = face_model)
vgg_face.summary()
```

Figure 46: Modify model architecture

c) Load weights and remove classification layer

```
In [6]: vgg_face.load_weights('weights/VGG_Equi_Face.h5')
        customized_vgg_face=Model(inputs=vgg_face.layers[0].input,outputs=vgg_face.layers[-2].output)
        customized_vgg_face.summary()
```

add_32 (Add)	(None, 7, 7, 2048)	0	conv5_3_1x1_increase/bn[0][0] activation_95[0][0]
activation_98 (Activation)	(None, 7, 7, 2048)	0	add_32[0][0]
avg_pool (AveragePooling2D)	(None, 1, 1, 2048)	0	activation_98[0][0]
flatten_2 (Flatten)	(None, 2048)	0	avg_pool[0][0]
dense_1 (Dense)	(None, 1024)	2098176	flatten_2[0][0]
dense_2 (Dense)	(None, 1024)	1049600	dense_1[0][0]
dense_3 (Dense)	(None, 512)	524800	dense_2[0][0]

```
Total params: 27,233,728
Trainable params: 3,672,576
Non-trainable params: 23,561,152
```

Figure 47: Load weights and remove classification layer

d) Step for Data pre-processing and metadata creation is same as ResNet-50 model (section 3.b)

e) Extract feature using fine-tuned ResNet-50 model

```
embedded = np.empty(512)
metadata_curated = []

for i, m in enumerate(metadata):
    img = load_image(m.image_path())
    img = align_image(img)
    if img is None:
        pass
    else:
        img = img.astype('float32')
        samples = expand_dims(img, axis=0)
        # prepare the face for the model, e.g. center pixels
        samples = preprocess_input(samples, version=2)
        metadata_curated.append(IdentityMetadata(m.base, m.name, m.file))
        embedded = np.vstack([embedded , customized_vgg_face.predict(samples)])
metadata_curated = np.array(metadata_curated)
embed = np.delete(embedded, 0, 0)
data_to_pkl(embed, "ResNet-50_Fine-Tuned_MOT360_fet.pkl")
```

```
Saving data to file(ResNet-50_Fine-Tuned_MOT360_fet.pkl).

True
```

```
feature_list = []
for i in embed:
    feature_list.append(i.flatten())
features_array = np.array(feature_list)
```

```
pca = PCA(n_components=0.95)
pca.fit(features_array)
pca_embed = pca.transform(embed)
```

Figure 48: Feature extraction and PCA

f) Analysis of extracted features : identity cluster

```
from sklearn.manifold import TSNE

targets = np.array([m.name for m in metadata_curated])
X_embedded = TSNE(n_components=2).fit_transform(pca_embed)

for i, t in enumerate(set(targets)):
    idx = targets == t
    pyplot.scatter(X_embedded[idx, 0], X_embedded[idx, 1], label=t)

pyplot.legend(bbox_to_anchor=(1, 1));
```

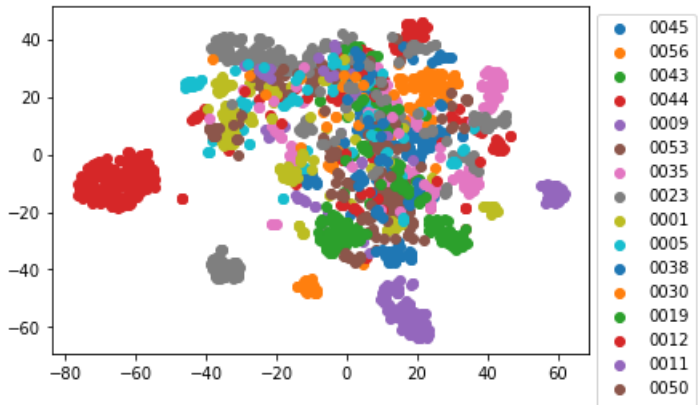


Figure 49: Identity cluster

g) Face identification implementation using fine-tuned ResNet-50

```
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC

targets = np.array([m.name for m in metadata_curated])

encoder = LabelEncoder()
encoder.fit(targets)

# Numerical encoding of identities
y = encoder.transform(targets)

with open("data/train_idx_identification.pkl", "rb") as f:
    train_idx = pickle.load(f)

with open("data/test_idx_identification.pkl", "rb") as f:
    test_idx = pickle.load(f)

y_train = y[train_idx]
y_test = y[test_idx]

X_train = embed[train_idx]
X_test = embed[test_idx]

knn = KNeighborsClassifier(n_neighbors=1, metric='euclidean')
svc = LinearSVC()

knn.fit(X_train, y_train)
svc.fit(X_train, y_train)

acc_knn = accuracy_score(y_test, knn.predict(X_test))
acc_svc = accuracy_score(y_test, svc.predict(X_test))

print(f'KNN accuracy = {acc_knn}, SVM accuracy = {acc_svc}')
```

```
KNN accuracy = 0.8922651933701657, SVM accuracy = 0.919889502762431
```

Figure 50: Face identification using fine-tuned ResNet-50

h) Classification using Joint Bayesian

Steps for classification using Joint Bayesian for face verification implementation are same as Steps in section 3.f

Results of face verification using fine-tuned ResNet-50

```
In [102]: excute_performance("result.pkl", -16.9, -16.6, 0.01)

avg / total       0.87      0.86      0.86     12000

threshold:  -16.619999999999955
              precision    recall  f1-score   support

       False       0.84      0.90      0.87      6000
        True       0.90      0.83      0.86      6000

avg / total       0.87      0.86      0.86     12000

threshold:  -16.609999999999953
              precision    recall  f1-score   support

       False       0.84      0.91      0.87      6000
        True       0.90      0.83      0.86      6000

avg / total       0.87      0.87      0.87     12000
```

Figure 51: Face verification results for fine-tuned ResNet-50 model