Reconnaissance d’émotion Documentation

Release 2.0

Qiuye DONG

Apr 02, 2017

Contents:

CHAPTER 1

Introduction

This document is written for introduce the project “Reconnaissance des émotions”.

The project contains the following four components.

1. Extract the image from the image database and classify it to form the original database.

   process_dataset.py

2. Image pre-processing, in order to improve the speed and accuracy of expression recognition. Brightness Normalization and Geometric Normalization

   adjust.py

   preProcessing.py

3. The recognition part, extracting the feature value of the human face from the picture.

   extractFeaturesCoordinates.py

   FaceLand.py

   setFeaturesData.py : Save features data to the text files.

   utilText.py : I/O functions.

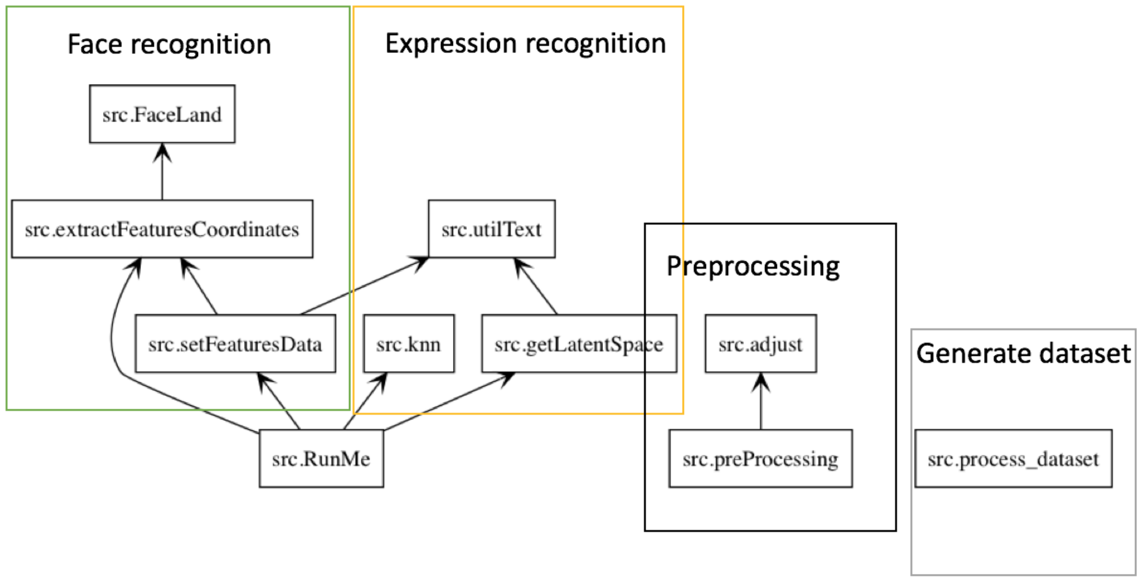
4. The expression recognition section, predict and use the kNN classification method to classify the result.

   getLatentSpace.py

   knn.py

   RunMe.py : Main function.

This is the package diagram of this project.


Face recognition
src.FaceLand
src.extractFeaturesCoordinates
src.setFeaturesData
Expression recognition
src.utilText
src.knn
src.getLatentSpace
src.RunMe
Preprocessing
src.adjust
src.preProcessing
Generate dataset
src.process_dataset

CHAPTER 2

Documentation for the Code

This section is used to explain in detail all the classes and functions in the project.

2.1 extractFeaturesCoordinates.py

This class is used to detect faces in a picture.

`extractFeaturesCoordinates.`**`extractFeatures`**(*basepath_train*, *show_images*)

This function detect landmarks for a dataset of faces.

Parameters

- **`basepath_train`** – (str) Training data storage path.
- **`show_images`** – (bool) Whether to display the results after detecting the face.

Returns It return a tuple (res, facedb) where res is to check is all was ok (False if there are more than one face on the image) and facedb contains the images, together with coordinates points of landmarks for each image

`extractFeaturesCoordinates.`**`getFeatures`**(*basepath_train*, *show_images*)

This function detect landmarks for a dataset of faces.

Parameters

- **`basepath_train`** – (str) Training data storage path.
- **`show_images`** – (bool) Whether to display the results after detecting the face.

Returns It return a tuple (res, facedb) where res is to check is all was ok (False if there are more than one face on the image) and facedb contains the images, together with coordinates points of landmarks for each image

2.2 FaceLand.py

This file contains the fonction to find frontal human faces in an image and estimate their pose. The pose takes the form of 68 landmarks. These are points on the face such as the corners of the mouth, on the eyes, and so forth.

Note: You can get the shape_predictor_68_face_landmarks.dat file from: http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2

2.3 getLatentSpace.py

This module is responsible for creating the latent space using traing and testing data.

`getLatentSpace.`**`genLatentSpace`**(*iters*, *latentDim*)

This function generate Latent Space (by Gaussian Process Latent Variable Model) using training data.

Parameters

- **`iters`** – Number of iterations to optimise the datas
- **`latentDim`** – Dimension of the latent space

Returns Mymodel (Gaussian Process Latent Variable Model): the model generated by user's data.

dataSamples (vector of numpyarray): the samples data in type vector.

labelSamples (vector of numpyarray): the classification label of the samples data.

`getLatentSpace.`**`genLatentSpaceTest`**(*dataAsListTest*, *labelsAsListTest*, *iters*, *latentDim*)

This function generate Latent Space (by Gaussian Process Latent Variable Model) using training data and testing data.

Parameters

- **`dataAsListTest`** – The samples data (data-type: map<class, list of dict[('points',array of couple),('dist',array of couple)]>
- **`labelsAsListTest`** – The classification label (list of int)
- **`iters`** – Number of iterations to optimise the datas
- **`latentDim`** – dimension of the latent space

Returns

Mymodel (Gaussian Process Latent Variable Model): the model generated by user's data.

dataSamples (vector of numpyarray): the samples data in type vector.

labelSamples (vector of numpyarray): the classification label of the samples data.

2.4 knn.py

kNN: k Nearest Neighbors

Input:

1. newInput: vector to compare to existing dataset (1xN)
2. dataSet: size m data set of known vectors (NxM)
3. labels: data set labels (1xM vector)
4. k: number of neighbors to use for comparison

Output:

1. the most popular class label

`classify.knn.`**`Emotion_dict`** **= {1: 'joie', 2: 'degout', 3: 'tristesse', 4: 'colere', 5: 'surprise'}**

Dictionary of emotions: In this project we use 5 base emotions.

`classify.knn.`**`Matrix_dict`** **= {'surprise': 0, 'tristesse': 0, 'colere': 0, 'degout': 0, 'joie': 0}**

Dictionary of result matrix: This dictionary is used for recording the results of each prediction.

`classify.knn.`**`createDataSet`**`(`*dataList*, *lableList*, *lenTest*`)`

Prepare data sets for predict

Parameters

- **`dataList`** – The coordinates of all expressions in latent space.
- **`lableList`** – The classification labels of all expressions.
- **`lenTest`** – the number of the test data

Returns

groupData: Array list of the training data's coordinates

groupTest: Array list of the testing data's coordinates

labels: list of classification labels

`classify.knn.`**`kNNClassify`**`(`*newInput*, *dataSet*, *labels*, *k*`)`

kNN method

Parameters

- **`newInput`** – Data being classified
- **`dataSet`** – Training data
- **`labels`** – The classification labels
- **`k`** – nNumber of neighbors to use for comparison

Returns

maxIndex: The closest expression index.

classCount: The matrix of the result.

`classify.knn.`**`predictImage`**`(`*dataList*, *lableList*, *lenTest*, *K*, *labelsAsListTest*`)`

Classify emotions using kNN then generate the predict emotion.

Parameters

- **`dataList`** – The coordinates of all expressions in latent space.
- **`lableList`** – The classification labels of all expressions.
- **`lenTest`** – the number of the test data
- **`K`** – Number of neighbors to use for comparison
- **`labelsAsListTest`** – the real classification labels of testing datas.

Returns Print the Matrix of results.

`classify.knn.`**`predictMatrix`**`(`*dataList*, *lableList*, *lenTest*, *K*, *labelsAsListTest*`)`

Classify emotions using kNN then generate the predict matrix.

Parameters

- **`dataList`** – The coordinates of all expressions in latent space.
- **`lableList`** – The classification labels of all expressions.
- **`lenTest`** – the number of the test data
- **`K`** – Number of neighbors to use for comparison
- **`labelsAsListTest`** – the real classification labels of testing datas.

Returns Print the Matrix of results.

2.5 process_dataset.py

This module is responsible for harvesting CK database for images of emotions. It gets a neutral face and a emotion face for each subject. Based on Paul van Gent’s code from blog post: http://www.paulvangent.com/2016/04/01/emotion-recognition-with-python-opencv-and-a-face-dataset/

2.6 RunMe.py

The main function of the project.

2.7 setFeaturesData.py

This module is responsible for saving features data to the text database.

`dataSet.setFeaturesData.`**`getListsFromImages`**`(`*basepath_train*`)`

This function performs the function of face detection. And the results will be saved to two list.

Parameters **`basepath_train`** – (str)Training data storage path.

Returns

dataAsList ({list}<type ‘int’>): A map contains all the detected feature values.

labelsAsList list(int): A list contains all the classification labels.

2.8 adjust.py

This module is responsible for adjusting the face in the image by the location of 2 eyes. Copyright (c) Philipp Wagner. All rights reserved.

`preProcess.adjust.`**`CropFace`**`(`*image, eye_left=(0, 0), eye_right=(0, 0), offset_pct=(0.2, 0.2), dest_sz=(70, 70)*`)`

Parameters

- **`image`** – Original image
- **`eye_left`** – Position of the left eye
- **`eye_right`** – Position of the right eye
- **`offset_pct`** – used for calculating offsets in original image
- **`dest_sz`** – size of cropped image

Returns Cropped image

`preProcess.adjust.`**`Distance`**`(`*p1, p2*`)`

This function calculate the distance between 2 points. :param p1: point 1 :param p2: point 2 :return: distance `preProcess.adjust.`**`ScaleRotateTranslate`**`(`*image*, *angle*, *center=None*, *new_center=None*, *scale=None*, *resample=3*`)`

This function adjusts the angle of the face. :param image: Image to be processed :param angle: The inclination angle of the face :param center: The center of the angle transformation :param new_center: New center :param scale: Scale factor :param resample: Image.BICUBIC :return: Transformed image

2.9 preProcessing.py

This module is responsible for preprocessing of images. The purpose is to improve the recognition rate of expression.

2.10 utilText.py

This module is used to implement IO operations as a tool.

It is called after the face detection,the purpose is to save the detected feature values in a document for later use. The operations for reading the detected feature values from the text are also implemented in this class.

Note:

Feature values and classification labels are stored in a different path.

- Feature values are stored in `FeaturesData/dataValues.txt`.
- classification labels are stored in `FeaturesData/labels.txt`.

`dataSet.utilText.`**`feature_values_path`** **= '/Users/Alex/Documents/PRD/IHM_EmotionDetection/ProjetSI_FinalVers**

str: The path of the text which stored feature values. Can be modified according to the requirements

`dataSet.utilText.`**`label_path`** **= '/Users/Alex/Documents/PRD/IHM_EmotionDetection/ProjetSI_FinalVersion/Ressourc**

str: The path of the text which stored labels of classification. Can be modified according to the requirements

`dataSet.utilText.`**`SaveFeatures`**`(`*dataAsList*`)`

This function store detected feature values into text file .

Parameters **`dataAsList`** – ({list}<type 'list'>) All the detected feature values.

`dataSet.utilText.`**`ReadFeatures`**`()`

This function read detected feature values from the text file .

Returns A map(int, list) contains all the detected feature values.

`dataSet.utilText.`**`SaveLabels`**`(`*dataLabelsAsList*`)`

This function store classification labels into text file.

:param dataLabelsAsList:({list}<type 'int'>) All the classification labels.

`dataSet.utilText.`**`ReadLabels`**`()`

This function read classification labels from the text file .

Returns A list(int) contains all the classification labels.

CHAPTER 3

Update Log

08/02/2017

Complete the first version of the document.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index