521285S Affective Computing (Laboratory Exercises)

Exercise – I Emotion Recognition from facial expression

Objective

The objective of the exercise is to build the facial expression recognition system. The system includes face preprocessing, feature extraction and classification. In the exercise, you will learn how to preprocess the facial expression image, extract the feature from an image or a video, and classify the video into one category.

Specifically, the region of interest (i.e., facial image) is extracted using face tracking, face registration and face crop functions. Basic spatiotemporal features (i.e., LBP-TOP features) are extracted using LBP-TOP. To produce emotion recognition case, Support Vector Machine (SVM) classifiers are trained. 50 videos from 5 participants are used to train the emotion recognition, use spatiotemporal features. The rest of the data (50 videos) is used to evaluate the performances of the trained recognition systems.

Database

The original facial expression data is a sub-set of eNTERFACE (acted facial expression), from ten actors acting **happy** and **sadness** behaviors. The used dataset in the exercise includes 100 facial expression samples.

Help

The data and toolbox files used in this exercise can be found in the Affective Computing course webpage (see the Noppa system).

Use the following website to help the usage of MATLAB functions.
http://se.mathworks.com/help/matlab/
Or type 'doc xxx' (e.g., doc plot) in the command window.

In the exercise, you should know that some basic image processing functions (plot, hist, imshow) and two classifier functions (svmtrain, svmclassify) before the programming. If you have questions, send your question to me (xiaohua.huang@ee.oulu.fi)

Requirement

Download the code for Exercise 1 ('Lab1_code_update.zip'), which includes three tasks ('Task1', Task2' and 'Task3'). We provide the main code for each task. You need to code them. We will check the results of each task. If you do it at home, you can save your results on images and then show it in the course.

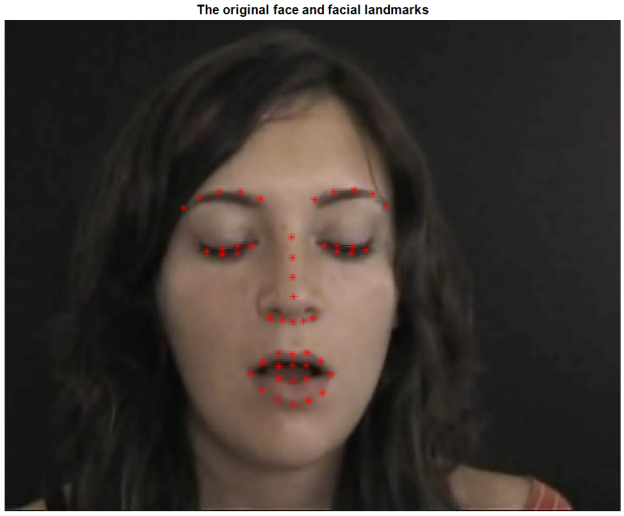
Task 1. Face preprocessing

You will use 'FaceTrack' and 'FaceCrop' to obtain the facial expression image:

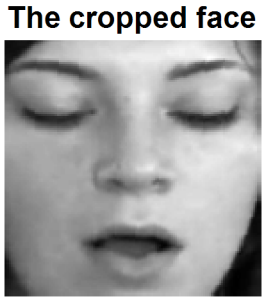
- Load 'example_img' variable from 'Task1_data'
- Use 'FaceTrack.m' to locate the 49 facial landmarks (variable: fpt). 'FaceTrack.m' is in the 'Chehra_v0.1_MatlabFit' file. In 'FaceTrack.m', there are three input argument, where if the last input argument is 1, it will show the image and landmark, otherwise, not.
- Register facial image (variable: 'example_img') into a model
    - Apply cp2tform on facial landmarks (variables: 'fpt' and 'model') to obtain the transform matrix (variable: tform). Usage: tform = cp2tform(xxx, xxx, 'lwm', 49), where xxx means the input landmarks.
    - Use 'imtransform.m' to register the facial image (variable: example_img) based on the transform matrix (variable: tform), and then obtain the registered facial image (variable: reg_img).
    - Use 'FaceCrop.m' to crop the facial image from the background (variable: my_norm_face).
- Use imshow function to see if you have got the cropped facial image.
- Use 'hist(double(my_norm_face(:)), 255)' to get the gray histogram in [0 255]. You can evaluate if your method is correct comparing with the gray histogram of our variable 'norm_face'. We are not sure the results will be the same due to matlab version.

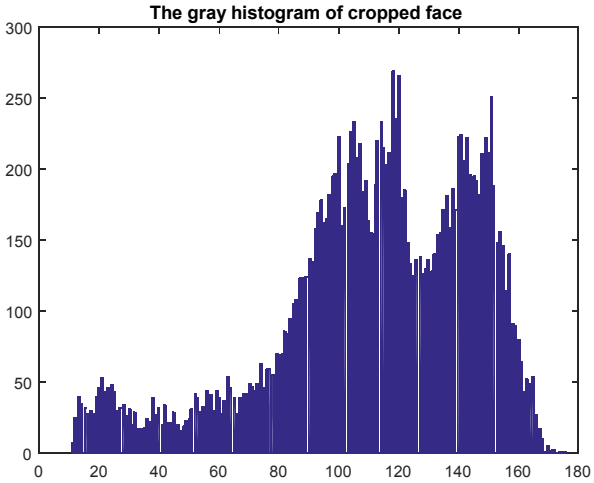
Show your results:

- Plot facial landmarks on the image.



- Plot your cropped image and the corresponding gray histogram.





Task 2. Feature extraction

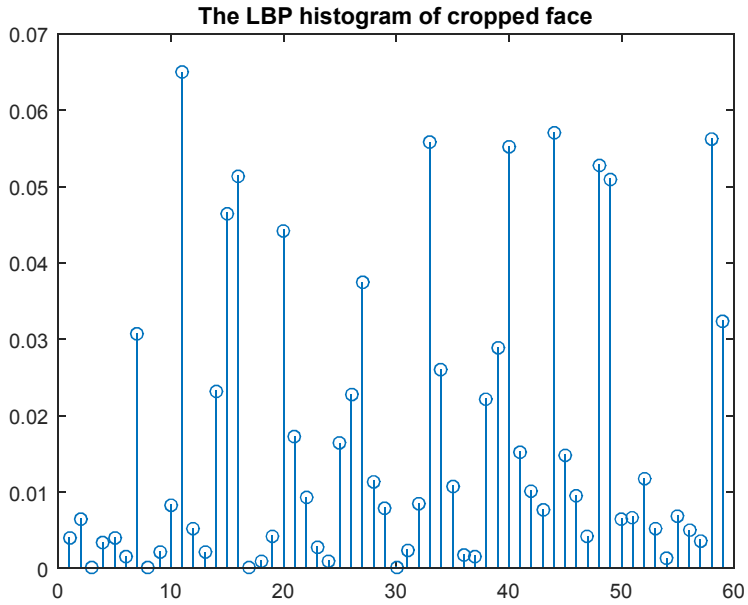
You will use lbp.m and getmapping.m to obtain one texture feature descriptor for face. The algorithm is named as Local Binary Pattern. So far many researcher use it in face recognition, facial expression recognition and texture classification. For the LBP function, we give one example:

```
%       I=imread('rice.png');
%       mapping=getmapping(8,'u2');
%       H1=LBP(I,1,8,mapping,'nh'); %LBP histogram in (8,1)
neighborhood, nh means we will get a normalized histogram
%                                   %using uniform patterns
%       subplot(2,1,1),stem(H1);
%       H1=LBP(I,1,8,mapping,'i'); %LBP histogram in (8,1)
neighborhood, I means we will get a LBP image
```

Extract the LBP feature in (8, 1) neighborhood.

- Set P = 8 and R = 1
- Load ‘norm_face’ variable from ‘task2_data’.
- Use getmapping to generate the uniform pattern (Eg. mapping = getmapping(P, ‘u2’))
- Use lbp.m function to obtain the LBP image

- Use lbp.m function to obtain the LBP normalized histogram.



Show and check the results:

- Check your LBP image and histograms.

Task 3. Feature Classification

Use the 'svmtrain' function to train Support Vector Machine (SVM) classifiers. The 'training_data' and 'testing_data' matrices contain the calculated LBP-TOP features for the training and testing sets, respectively. The block size for LBP-TOP used for training and testing data are 2x2x1. The 'training_class' group vector contains the class of samples: 1 = happy, 2 = sadness, corresponding to the rows of the training data matrices.

You need to do:

- Construct an SVM using the 'training_data' and linear kernel
- Use the 'svmclassify' function (and your trained SVM structures) to classify the 'training_data' and the 'testing_data' matrices. Then, calculate average classification performances for both training and testing data. The correct class labels corresponding with the rows of the training and testing data matrices are in the variables 'training_class' and 'testing_class', respectively.

- Calculate the average classification performances for the training data (‘training_data’) and the testing data (‘testing_data’) using the corresponding trained linear SVM.
- Calculate confusion matrices for the training and testing data for both classifiers. For example, C = confusionmat(group,grouphat)

Show and check the results:

- The average classification performances for the training data (‘training_data’) and the testing data (‘testing_data’) using the corresponding trained linear SVM.
- Calculate the confusion matrix of the linear kernel SVM using the ‘training_data’
- Calculate the confusion matrix of the linear kernel SVM using the ‘testing_data’