

Detection of Human Face and Facial Features and Recognition of Emotion on the Detected Human Face

Titli Sarkar

CLID# txs7980, txs7980@louisiana.edu

Ippili Aashish

CLID# axi1334, axi1334@louisiana.edu

Baregal Abhijeeth

CLID# axb6412, axb6412@louisiana.edu

Supervised By:

Dr. Henry Chu

A Project Submitted in Partial Fulfillment of the Requirements for the Course
CSCE-508: Image Processing

Project URL: <https://github.com/TitliSarkar/CSCE-508>



The Center for Advanced Computer Studies
University of Louisiana at Lafayette, USA
December 2016

Summary

Automatic Face Detection and then Emotion Detection on it is a useful problem in image processing which can be found useful to Intelligence bureau, Robotics, Apps developers etc. This project is based on facial feature extraction using the knowledge of the face geometry representing the relationship between the motion of features and change of expressions which enables us to detect human emotion from real life image. From human face structure, we divide in four regions such as right eye, left eye, nose and mouth areas from the face image, firstly come the detection of face and then detection of facial features regions. We crop the facial skin region and connect the largest region to detect the skin surface of the human face. Then we detect different facial features on the detected human face eg. Right and Left Eyes, Nose and Mouth. After facial features are detected, the expression on the face is tried to be recognized and classified. Classifier from Deep Learning concept is used to first train the system with human faces with different emotions. Then, emotion on the particular input image is tried to be recognized using the trained system.

I. Introduction

As the number of social media applications and image-based applications increase day by day, Facial Feature Detection system is becoming a popular feature in ‘apps’ and even on websites for the different purpose. Human face localization and detection are often the first step in applications such as video surveillance, human computer interface, and face recognition and image database management. Furthermore, we can detect facial expressions, which not only express our emotions, but also to provide important communicative cues during social interaction.

This project is based on human’s facial feature extraction from a given image and then categorizing the face as per different kinds of emotions. From the human face structure, we divide the region into four sub-regions such as right eye, left eye, nose, mouth. Then we detect human emotions on that face.

II. Problem Statement

The Human’s facial feature extraction from an input image and detecting emotions on it. This involves the following two steps:

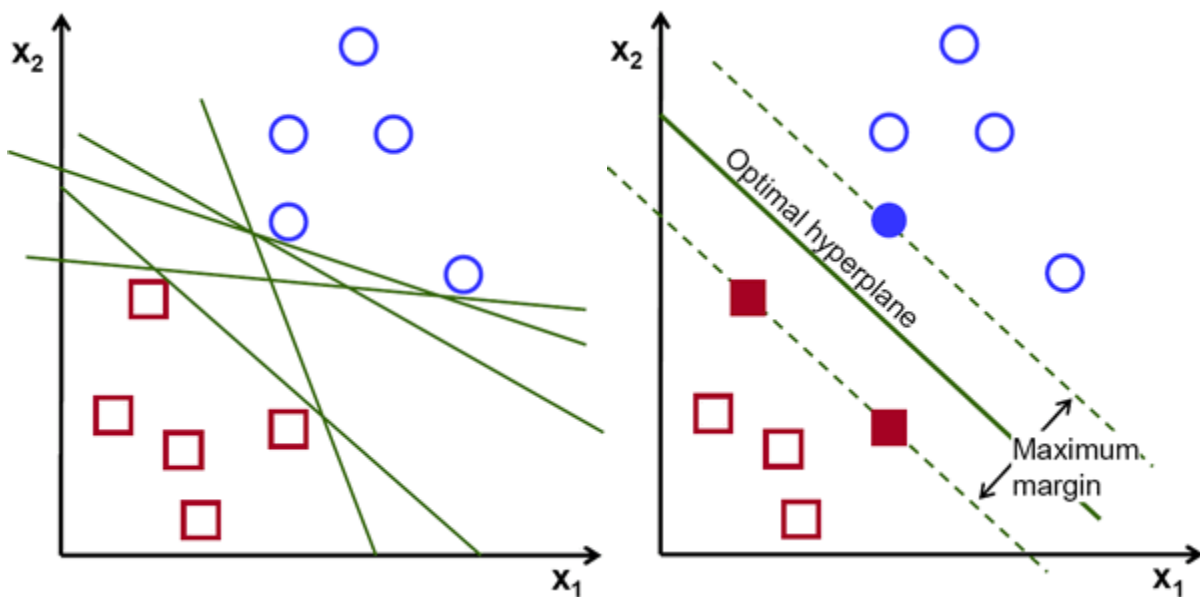
- i. **Face and Facial Features Recognition:** An input image will be extracted for finding the four facial regions, i.e. ‘Right Eye’, ‘Left Eye’, ‘Nose’, ‘Mouth’.
- ii. **Emotion Detection:** Emotion of the image is recognized. Expressions to be detected are:
 - Neutral
 - Happy
 - Sad
 - Fear

III. Methodology

In *deep learning*, the *classifier* is a method of comparing the image to the set of categories and finalizing the category based on observations and on *training dataset* of data containing observations whose category membership is known. At first, we need to train the classifier with some known data set (i.e. training dataset) and *label* them in groups. When the new image is uploaded, the classifier predicts in which class or group the new image will fall based on the knowledge of trained dataset, i.e. the classifier separates classes in feature space.

The *First part* of the project is done using **Cascade Classifier** for detecting an object in a captured image. Cascading Classifiers are trained with several hundred "positive" sample views of a particular object and arbitrary "negative" images of the same size. After the classifier is trained it can be applied to a region of an image and detect the object in question. To search for the object in the entire frame, the search window can be moved across the image and check every location for the classifier.

The *Second part* of the project has used **Support Vector Machine**. A Support Vector Machine (**SVM**) is a discriminating classifier formally defined by a separating hyperplane that analyzes data and recognizes patterns used for classification. In other words, given labeled training data (*supervised learning*), the algorithm outputs an optimal hyperplane which categorizes new examples.



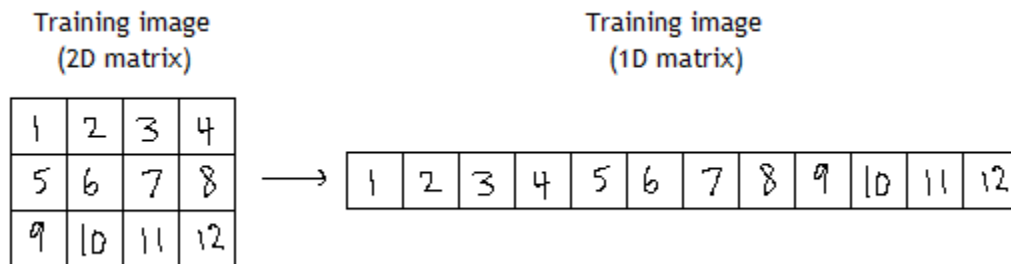
IV. Algorithm

1. ***Acquisition of input image:*** Read the Input Human Face Image. If the Input Image is colored (RGB), then convert it to the grayscale image.
2. ***Detection of face and facial features:*** Using *Cascade Classifier* (a *Harr* like classifier), identify Human Face region and regions of Right Eye, Left Eye, Nose, Mouth. *DetectMultiScale()* function does this work. A grayscale image, one vector attributes representing a facial feature and some other functional attributes are passed as arguments.
3. ***Identifying facial regions:*** Rectangular boundaries are drawn around the face and facial features regions.
4. ***Classification of Expression on detected human face:*** Using *SVM*, detect expressions on recognizing human face.
 - a. Get all images to be used for *TrainingData* and the input image on which emotions should be detected.
 - b. *Preprocessing*: Resize all those images to the same size.
 - c. Form *TrainingDataSet* with those images.
 - d. Set up *Labels* corresponding to each of the images in the *TrainingDataSet*.
 - e. Set up the parameters of *SVM*.
 - f. *Train* the *SVM* with *TrainingDataSet* and corresponding *Labels*. Each label should correspond to an image with a specific type of emotion.
 - g. *Predict* the emotion on the face of Input Image by matching with the result of trained dataset.
 - h. Pop up the specific emotion based on the result of predicting function.

V. Implementation

1. The *SVM* module available in the *OpenCV* documentation was for classifying points only. In this project this module is used to implement it for 2D color images. This method can work on different type of images (*JPEFG*, *PNG*, *GIF* etc.).
2. In implementation of the algorithm, user has freedom to choose the input image for testing.

3. If the input image is not grayscale, the code first converts it to grayscale before processing.
4. All images (the input image and the training images) need to be equal sized for processing. So, after taking input, all the images need to be resized to a fixed size.
5. To train the SVM on a set of images, first, the training matrix for the SVM need to be constructed. This matrix is specified as follows: each row of the matrix corresponds to one image, and each element in that row corresponds to one feature of the class -- in this case, the color of the pixel at a certain point. Since the images are 2D, they should be converted to a 1D matrix. The length of each row will be the area of the images. Below is an example of how this mapping would work for one row.



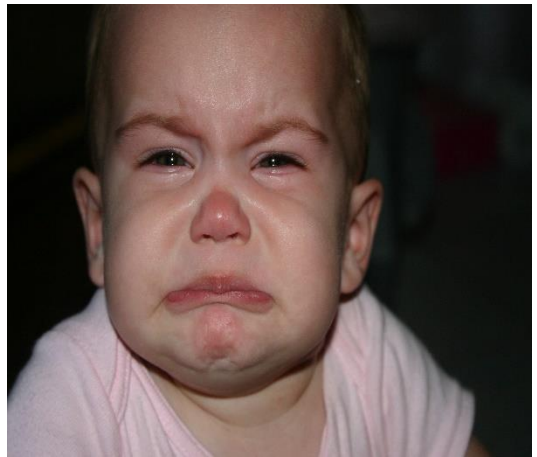
6. The each element of the label matrix need to correctly correspond each row of the training matrix. Later, this data will be matched with the result of svmPredict(Input Image) to determine the correct emotion on Input Image.

V. Results

Training Dataset:



Happy Image



Sad Image

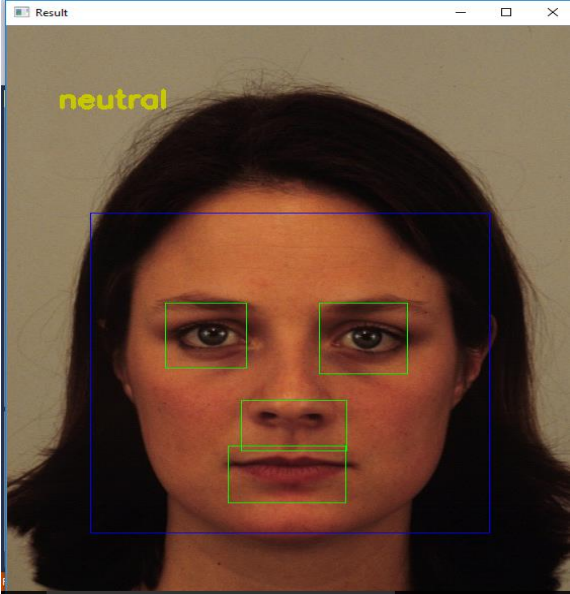




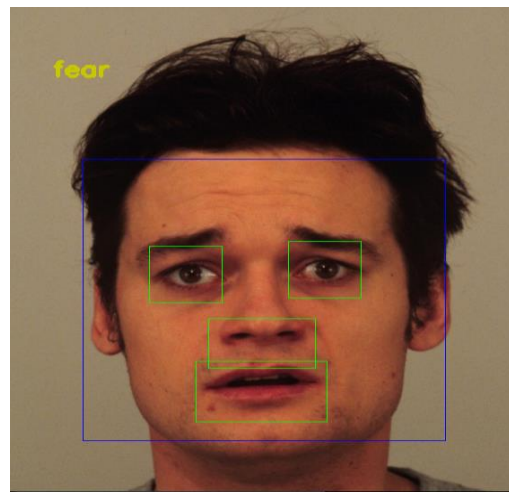
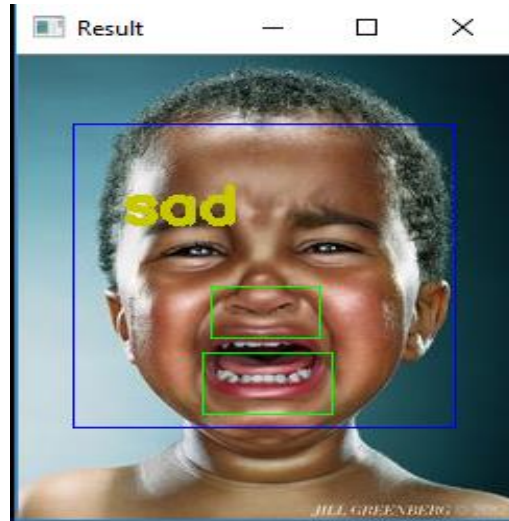
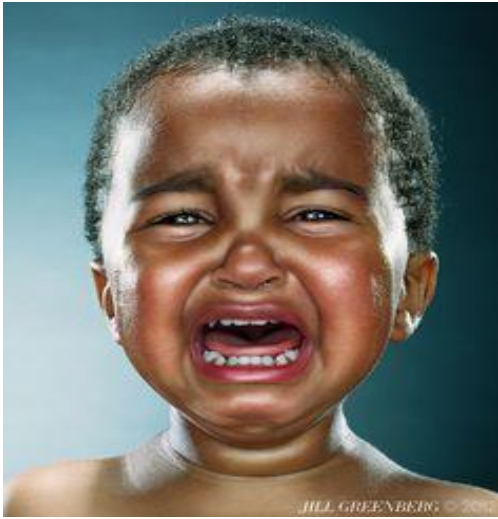
Fear Image



Neutral Image

Test Results:

Input image	Output image
	
	





VI. Performance Analysis

Accuracy: In *first* part, for detection of the face, the skin region need to be clearly separable from the color of rest of the regions of the image. The facial feature regions also need to be very prominent in the input image and the image need to be aligned front to be get detected accurately. If not, for example, if the input image is not perfectly aligned and if the facial skin region is almost same color as rest of the regions in the image the face region does not get detected properly by this algorithm. The input image need to be near similar to training image.

In *second* part, one training image is used for each type of emotions in the training set. Some emotions are very similar facial features to each other, so it may happen that in some cases the classifier cannot correctly recognize the emotion on the face. More training images for every type of emotion will give more accurate result.

The **overall** recognition rate of the proposed algorithm is **79%**.

Performance measure of Part(1): Facial Feature Detection:

No. of input images	No. of images whose facial features are identified accurately	Result (%)
12	9	75%

Performance measure of Part (2): Expression Classification:

Total no. of tested images	No. of training sets	Total no. of images on training set	No. of images whose expression are identified accurately	Result (%)
12	1	4	10	83%

VII. Specifications

Dataset: We have used downloaded images from Google as both input data and training data.

Operating System: Windows 10 amd64

Platform: Visual Studio 2013 Professional

Tools: OpenCV 2.4

Programming Language: OpenCV C++

VII. Conclusion

SVM, in general works as a non-probabilistic *binary linear classifier*. But in OpenCV, SVM can be used as a *multi-classifier* (no. Of class ≥ 2) to classify more than two expressions, which advantage we have utilized. Due to lack of a huge set of the training set of images in expression classification, the proposed method cannot accurately identify any facial expression. If the training database can be loaded with more different images, then the method will work more accurately. Future work will be done to classify more different types of expressions in the human face and identify each emotion properly. Some measure of recognizing faces and facial features with on input images with different alignment is also in the scope of future work. We also want to classify more emotions.

IX. References

1. http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html
2. <http://stackoverflow.com/questions/14694810/using-opencv-and-svm-with-images>
3. http://docs.opencv.org/2.4/doc/tutorials/objdetect/cascade_classifier/cascade_classifier.html

X. Appendix

source.cpp

```
/* This code detects face and facial features on an input image
and recognises emotion on it. Input is taken by users choice
from a list of images */
#include "opencv2/objdetect.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/ml/ml.hpp>
#include <opencv/cv.h>
#include <iostream>
#include <cstdio>
#include <vector>
#include <algorithm>
#include <Windows.h>
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <dirent.h>
using namespace std;
using namespace cv;

// Functions for facial feature detection

static void detectFaces(Mat&, vector<Rect_<int> >&, string);
static void detectEyes(Mat&, vector<Rect_<int> >&, string);
static void detectNose(Mat&, vector<Rect_<int> >&, string);
static void detectMouth(Mat&, vector<Rect_<int> >&, string);
static void detectFacialFeatures(Mat&, const vector<Rect_<int> >,
string, string, string);
static float setsvm(String, String, String, String, String);

string input_image_path;
string face_cascade_path, eye_cascade_path, nose_cascade_path,
mouth_cascade_path;

int main(int argc, char** argv)
{
    cv::CommandLineParser parser(argc, argv,
        "{eyes|}|{nose|}|{mouth|}|");

    input_image_path = "C://irs/AF02HAS.jpg";
```

```

    face_cascade_path =
"C://irs/haarcascade_frontalface_alt.xml";
    eye_cascade_path = "C://irs/haarcascade_eye.xml";
    nose_cascade_path = "C://irs/haarcascade_mcs_nose.xml";
    mouth_cascade_path = "C://irs/haarcascade_mcs_mouth.xml";

    String pImg =
"F:\\CODES\\OPENCVCodes\\DatasetSVM\\disgust1.jpg";

    if (input_image_path.empty() || face_cascade_path.empty())
    {
        cout << "IMAGE or FACE_CASCADE are not specified";
        return 1;
    }
    // Load image and cascade classifier files
    Mat image;
    image = imread(input_image_path);

    // Detect faces and facial features
    vector<Rect_<int> > faces;
    wchar_t szFileName[MAX_PATH] = { 0 };
    OPENFILENAMEW ofn;
    ZeroMemory(&ofn, sizeof(ofn));
    ofn.lStructSize = sizeof(OPENFILENAME);
    ofn.nMaxFile = MAX_PATH;
    ofn.lpstrFile = szFileName;

    GetSaveFileNameW(&ofn);

    wstring ws(szFileName);
    string str(ws.begin(), ws.end());

    cv::Mat frame = imread(str);
    detectFaces(frame, faces, face_cascade_path);
    detectFacialFeaures(frame, faces, eye_cascade_path,
nose_cascade_path, mouth_cascade_path);
    DIR *dir;
    struct dirent *ent;
    std::vector<std::string> collection1;
    if ((dir = opendir("c:\\irs\\KDEF")) != NULL) {
        /* print all the files and directories within
directory */
        while ((ent = readdir(dir)) != NULL) {
            collection1.push_back(ent->d_name);
        }
        closedir(dir);
    }

```

```

    }
    else {
        /* could not open directory */
        perror("");
        return EXIT_FAILURE;
    }
    String mood;
    int NESCount = 0;
    int HASCount = 0;
    int SASCount = 0;
    int AFSCount = 0;
    int DISCount = 0;

    //dirent end

        //cout << collection1[i] << endl;
    String timg1 = "C:\\\\irs\\IMDataset\\girl.jpg";
    String timg2 = "C:\\\\irs\\IMDataset\\smiling2.jpg";
    String timg3 = "C:\\\\irs\\IMDataset\\sad.jpg";
    String timg4 = "C:\\\\irs\\IMDataset\\AF01AFS.JPG";

    float res = setsvm(timg1, timg2, timg3, timg4, str);
    if (res == -1)
        mood = "neutral";
    else if (res == 1)
        mood = "smiling";
    else if (res == 2)
        mood = "sad";
    else if (res == 3)
        mood = "fear";
    putText(frame, mood, Point(50, 100), FONT_HERSHEY_SIMPLEX,
1, Scalar(0, 200, 200), 4);
    imshow("Result", frame);

    waitKey(0);
    return 0;
}

static void detectFaces(Mat& img, vector<Rect_<int> >& faces,
string cascade_path)
{
    CascadeClassifier face_cascade;
    face_cascade.load(cascade_path);

    face_cascade.detectMultiScale(img, faces, 1.15, 3, 0 |
CASCADE_SCALE_IMAGE, Size(30, 30));

```

```

        return;
    }

    static void detectFacialFeaures(Mat& img, const
    vector<Rect_<int> > faces, string eye_cascade,
        string nose_cascade, string mouth_cascade)
    {
        for (unsigned int i = 0; i < faces.size(); ++i)
        {
            // Mark the bounding box enclosing the face
            Rect face = faces[i];
            rectangle(img, Point(face.x, face.y), Point(face.x +
            face.width, face.y + face.height),
                Scalar(255, 0, 0), 1, 4);

            // Eyes, nose and mouth will be detected inside the
            face (region of interest)
            Mat ROI = img(Rect(face.x, face.y, face.width,
            face.height));

            // Check if all features (eyes, nose and mouth) are
            being detected
            bool is_full_detection = false;
            if ((!eye_cascade.empty()) && (!nose_cascade.empty())
            && (!mouth_cascade.empty()))
                is_full_detection = true;

            // Detect eyes if classifier provided by the user
            if (!eye_cascade.empty())
            {
                vector<Rect_<int> > eyes;
                detectEyes(ROI, eyes, eye_cascade);

                // Mark points corresponding to the centre of the
                eyes
                for (unsigned int j = 0; j < eyes.size(); ++j)
                {
                    Rect e = eyes[j];
                    //circle(ROI, Point(e.x + e.width / 2, e.y +
                    e.height / 2), 3, Scalar(0, 255, 0), -1, 8);
                    rectangle(ROI, Point(e.x, e.y), Point(e.x +
                    e.width, e.y + e.height),
                        Scalar(0, 255, 0), 1, 4);
                }
            }

            // Detect nose if classifier provided by the user

```



```

double nose_center_height = 0.0;
if (!nose_cascade.empty())
{
    vector<Rect_<int> > nose;
    detectNose(ROI, nose, nose_cascade);

    // Mark points corresponding to the centre (tip)
of the nose
    for (unsigned int j = 0; j < nose.size(); ++j)
    {
        Rect n = nose[j];
        //circle(ROI, Point(n.x + n.width / 2, n.y +
n.height / 2), 3, Scalar(0, 255, 0), -1, 8);
        nose_center_height = (n.y + n.height / 2);
    }
}

// Detect mouth if classifier provided by the user
double mouth_center_height = 0.0;
if (!mouth_cascade.empty())
{
    vector<Rect_<int> > mouth;
    detectMouth(ROI, mouth, mouth_cascade);

    for (unsigned int j = 0; j < mouth.size(); ++j)
    {
        Rect m = mouth[j];
        mouth_center_height = (m.y + m.height / 2);

        // The mouth should lie below the nose
        if ((is_full_detection) &&
(mouth_center_height > nose_center_height))
        {
            rectangle(ROI, Point(m.x, m.y),
Point(m.x + m.width, m.y + m.height), Scalar(0, 255, 0), 1, 4);
        }
        else if ((is_full_detection) &&
(mouth_center_height <= nose_center_height))
            continue;
        else
            rectangle(ROI, Point(m.x, m.y),
Point(m.x + m.width, m.y + m.height), Scalar(0, 255, 0), 1, 4);
    }
}
}

```

```

        return;
    }

static void detectEyes(Mat& img, vector<Rect_<int> >& eyes,
string cascade_path)
{
    CascadeClassifier eyes_cascade;
    eyes_cascade.load(cascade_path);

    eyes_cascade.detectMultiScale(img, eyes, 1.20, 5, 0 |
CASCADE_SCALE_IMAGE, Size(30, 30));
    return;
}

static void detectNose(Mat& img, vector<Rect_<int> >& nose,
string cascade_path)
{
    CascadeClassifier nose_cascade;
    nose_cascade.load(cascade_path);

    nose_cascade.detectMultiScale(img, nose, 1.20, 5, 0 |
CASCADE_SCALE_IMAGE, Size(30, 30));
    return;
}

static void detectMouth(Mat& img, vector<Rect_<int> >& mouth,
string cascade_path)
{
    CascadeClassifier mouth_cascade;
    mouth_cascade.load(cascade_path);

    mouth_cascade.detectMultiScale(img, mouth, 1.20, 5, 0 |
CASCADE_SCALE_IMAGE, Size(30, 30));
    return;
}

// another working code

float setsvm(String tImage1, String tImage2, String tImage3,
String tImage4, String pImage){
    // Data for visual representation
    //dirent start

    int num_files = 4;
    int h = 128, w = 128;
    Size size(h, w);
    /*Setup for 'Neutral Face'*/
    // Set up training data

```

```

Mat image[4];
image[0] = imread(tImage1, 0);
image[1] = imread(tImage2, 0);
image[2] = imread(tImage3, 0);
image[3] = imread(tImage4, 0);

resize(image[0], image[0], Size(w, h));
resize(image[1], image[1], Size(w, h));
resize(image[2], image[2], Size(w, h));
resize(image[3], image[3], Size(w, h));

Mat trainingDataMat(4, h*w, CV_32FC1); //Training sample
from input images
int ii = 0;
for (int i = 0; i < num_files; i++){
    Mat temp = image[i];
    ii = 0;
    for (int j = 0; j < temp.rows; j++){
        for (int k = 0; k < temp.cols; k++){
            trainingDataMat.at<float>(i, ii++) =
temp.at<uchar>(j, k);
        }
    }
}
//Set up labels
Mat labels(num_files, 1, CV_32FC1);
labels.at<float>(0, 0) = -1.0; //neutral
labels.at<float>(1, 0) = 1.0; //smiling
labels.at<float>(2, 0) = 2.0; //sad
labels.at<float>(3, 0) = 3.0; //fear

CvSVMParams params;
params.svm_type = CvSVM::C_SVC;
params.kernel_type = CvSVM::LINEAR;
params.gamma = 3;

params.term_crit = cvTermCriteria(CV_TERMCRIT_ITER, 100,
1e-6);

// Train the SVM
CvSVM svm;
svm.train(trainingDataMat, labels, Mat(), Mat(), params);
svm.save("svm.xml"); // saving
svm.load("svm.xml"); // loading

//Taking input image
Mat test_img = imread(pImage, 0);

```

```
resize(test_img, test_img, Size(w, h));
test_img = test_img.reshape(0, 1);
//imshow("shit_image", test_img);
test_img.convertTo(test_img, CV_32FC1);

//Predicting emotion
float res = svm.predict(test_img);
return res;
} //end of function setsvm()
```