DePaul University

College of Computing and Digital Media

Facial Expression Recognition System

Master's Thesis Technical Report

Author: Ewa Piątkowska

Supervisor: Prof. Dr. Jacob Furst

Submission Date: 20th of July, 2010

Acknowledgments

I would like to express my gratitude to my supervisor Prof. Jacob Furst and Prof. Daniela Raicu for their valuable advice and support.

I would also like to thank Prof. T. Kanade, Prof. J.F. Cohn and Prof. Y. Tian for providing the Cohn-Kanade database and Dr.-Ing. F. Wallhoff for the access to FG-NET Facial Expressions and Emotion Database.

Abstract

This thesis describes the problem of facial expression recognition in the field of computer vision. Firstly, the psychological background of a problem is presented. Then, the idea of facial expression recognition system (FERS) is outlined and the requirements of such system are specified. The FER system consists of 3 stages: face detection, feature extraction and expression recognition. Methods proposed in literature are reviewed for each stage of a system. Finally, the design and implementation of my system are explained. The face detection algorithm used in the system is based on work by Viola and Jones [13]. The expressions are described by appearance features obtained from texture encoded with Local Binary Patterns [32]. The Support Vector Machine with RBF kernel function is used for classification. Databases that were used are: The Facial Expressions and Emotion Database [34], which contains spontaneous emotions and Cohn-Kanade Database [35] with posed emotions. The system was trained on two databases separately and achieves accuracy of 71% for spontaneous emotions recognition and 77% for posed actions recognition.

Table of Contents

Introduction

Face plays significant role in social communication. This is a 'window' to human personality, emotions and thoughts. According to the psychological research conducted by Mehrabian [1], nonverbal part is the most informative channel in social communication. Verbal part contributes about 7% of the message, vocal – 34% and facial expression about 55%. Due to that, face is a subject of study in many areas of science such as psychology, behavioral science, medicine and finally computer science.

In the field of computer science much effort is put to explore the ways of automation the process of face detection and segmentation. Several approaches addressing the problem of facial feature extraction have been proposed. The main issue is to provide appropriate face representation, which remains robust with respect to diversity of facial appearances.

The objective of this report is to outline the problem of facial expression recognition, that is a great challenge in the area of computer vision. Advantages of creating a fully automatic system for facial action analysis are constant motivation for exploring this field of science and will be mentioned in this thesis.

First Chapter is devoted to the psychological background of the facial expression recognition problem. The motivation of such study is outlined from the psychological aspect. Moreover, the techniques used by psychologists for facial action analysis are presented.

Second chapter shows the idea of facial expression recognition system, the way such system is composed and its main features and requirements. Furthermore, approaches proposed in the literature will be described briefly. Finally, the application areas will be mentioned to show that automatic facial action recognition is widely used.

In the third chapter, I will present the design and implementation of Facial Expression Recognition System. Techniques used at each stage of my system will be described and explained. Moreover, the performance of a system will be evaluated by testing the recognition accuracy on two training sets.

1 Psychological Background

In 1978, Ekman et al. [2] introduced the system for measuring facial expressions called FACS – Facial Action Coding System. FACS was developed by analysis of the relations between muscle(s) contraction and changes in the face appearance caused by them. Contractions of muscles responsible for the same action are marked as an Action Unit (AU).

The task of expression analysis with use of FACS is based on decomposing observed expression into the set of Action Units. There are 46 AUs that represent changes in facial expression and 12 AUs connected with eye gaze direction and head orientation. Action Units are highly descriptive in terms of facial movements, however, they do not provide any information about the message they represent. AUs are labeled with the description of the action (Fig.1).

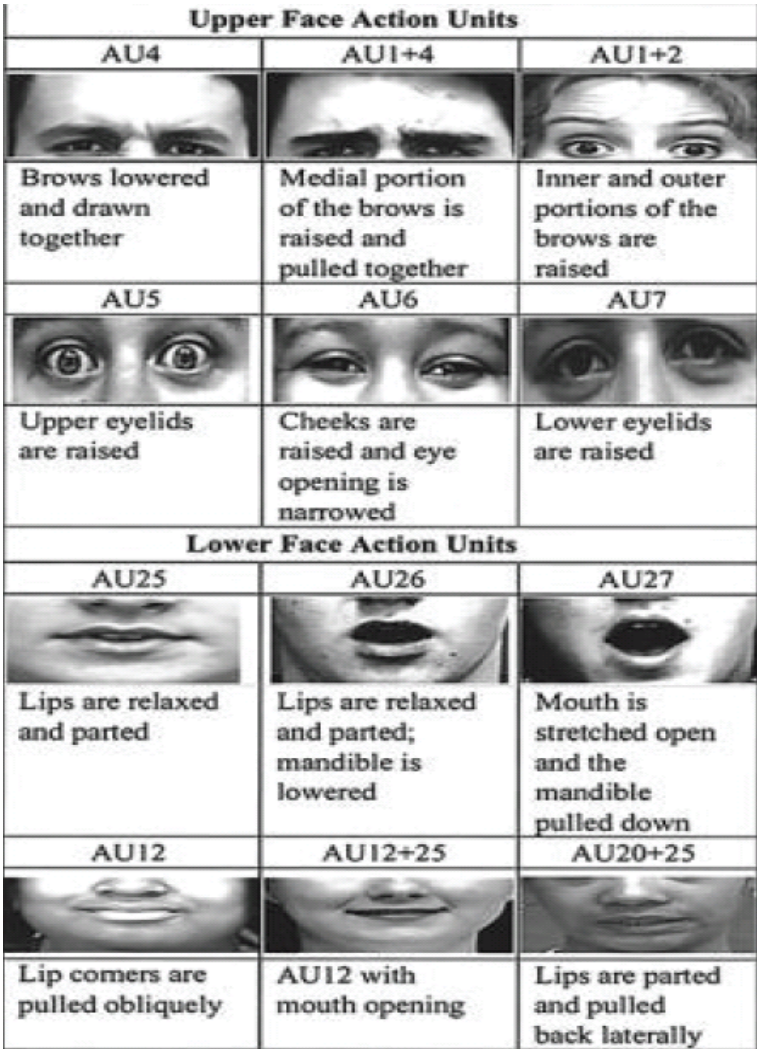

Fig. 1: Examples of Action Units [3]

Facial expression described by Action Units can be then analyzed on the semantic level in order to find the meaning of particular actions. According to the Ekman's theory [2], there are six basic emotion expressions that are universal for people of different nations and cultures. Those basic emotions are joy, sadness, anger, fear, disgust and surprise (Fig. 2).

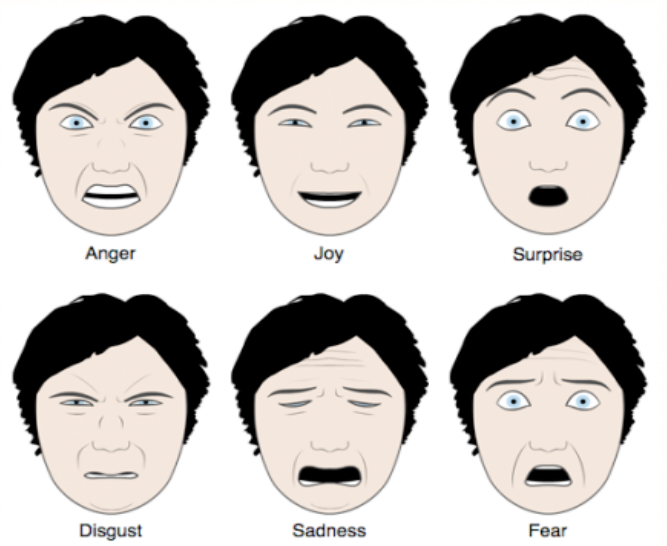

Fig. 2: Six universal emotions [4]

The Facial Action Coding System was developed to help psychologists with face behavior analysis. Facial image was studied to detect the Action Units occurrences and then AU combinations were translated into emotion categories. This procedure required much effort, not only because the analysis was done manually, but also because about 100 hours of training were needed to become a FACS coder. That is why, FACS was quickly automated and replaced by different types of computer software solutions.

2 Related work

The system that is designed for automatic analysis of facial actions is usually called Facial Expression Recognition System (FERS). The FER system is composed of 3 main elements: face detection, feature extraction and expression recognition. Different methods were proposed for each stage of the system, however, only the major ones will be mentioned in the report. More in-depth study and comparison of related work can be found in surveys done by Pantic and Rothkrantz [5] as well as by Zeng et al. [6].

Firstly, I would like to outline the basic idea of the FER system and explain the most important issues which should be taken under consideration in the process of system design and development. Then, each FER system stage will be described in details, namely: main task, typical problems and proposed methods. Furthermore, the recent advances in the area of facial expression analysis will be listed. Finally, some exemplary applications of FER systems will be mentioned to show that they are widely used in many fields of science as well as in everyday life.

2.1 System requirements

The goal of FERS is to imitate the human visual system in the most similar way. This is very challenging task in the area of computer vision because not only it requires efficient image/video analysis techniques but also well-suited feature vector used in machine learning process.

The first principle of FER system is that it should be effortless and efficient. That is connected with full automation, so that no additional manual effort is required. It is also preferred for such system to be real-time which is especially important in both: human-computer interaction and human-robot interaction applications.

Furthermore, the subject of study should be allowed to act spontaneously while data is being captured for analysis. System should be designed to avoid limitations on body and head movements which could also be an important source of information about displayed emotion. The constraints about facial hair, glasses or additional make-up should be reduced to minimum. Moreover, handling the occlusions problem seems to be a challenge for a system and it should be also taken into account.

Another important features that are desired in FER system are user and environment independence. The former means that, any user should be allowed to work with the system, despite of skin color, age, gender or nation.
The latter is connected with handling the complex background and variety in lightning conditions.
Additional benefit could be the view independence in FERS, which is possible in systems based on 3D vision.

2.2 Face detection

As it was mentioned before, FER system consists of 3 stages. In the first stage, system takes input image and performs some image processing techniques on it in order to find the face region. System can operate on static images, where this procedure is called face localization or videos where we are dealing with face tracking.

Major problems which can be encountered at this stage are different scales and orientations of face. They are usually caused by subject movements or changes in distance from camera. Significant body movements can also cause drastic changes in position of face in consecutive frames what makes tracking harder. What is more, complexity of background and variety of lightning conditions can be also quite confusing in tracking. For instance, when there is more than one face in the image, system should be able to distinguish which one is being tracked. Last but not least, occlusions which usually appear in spontaneous reactions need to be handled as well.

Problems mentioned above were a challenge to search for techniques which would solve them. Among the techniques for face detection, we can distinguish two groups: holistic where face is treated as a whole unit and analytic where co-occurrence of characteristic facial elements is studied.

<u>Holistic face models:</u>

- Huang and Huang [7] used Point Distribution Model (PDM) which represents mean geometry of human face. Firstly, Canny edge detector is applied to find two symmetrical vertical edges which estimate the face position and then PDM is fitted.
- Pantic and Rothkrantz [8] proposed system which process images of frontal and profile face view. Vertical and horizontal histogram analysis is used to find face boundaries. Then, face contour is obtained by thresholding the image with HSV color space values.

Analytic face models:

- Kobayashi and Hara [9] used image captured in monochrome mode to find face brightness distribution. Position of face is estimated by iris localization.
- Kimura and Yachida [10] technique processes input image with an integral projection algorithm to find position of eye and mouth corners by color and edge information. Face is represented with Potential Net model which is fitted by the position of eyes and mouth.

All of the above mentioned systems were designed to process facial images, however, they are not able to detect whether the face is present in the image. Systems which handle arbitrary images are listed below:

- Essa and Pentland [11] created the "face space" by performing Principal Component Analysis of eigenfaces from 128 face images. Face is detected in the image if its distance from the face space is acceptable.
- Rowley et al. [12] proposed neural network based face detection. Input image is scanned with a window and neural network decides if particular window contains a face or not.
- Viola and Jones [13] introduced very efficient algorithm for object detection with use of Haar-like features as object representation and Adaboost as machine learning method. This algorithm is widely used in face detection.

2.3 Feature extraction

After the face has been located in the image or video frame, it can be analyzed in terms of facial action occurrence. There are two types of features that are usually used to describe facial expression: geometric features and appearance features. Geometric features measure the displacements of certain parts of the face such as brows or mouth corners, while appearance features describe the change in face texture when particular action is performed. Apart from feature type, FER systems can be divided by the input which could be static images or image sequences.

The task of geometric feature measurement is usually connected with face region analysis, especially finding and tracking crucial points in the face region. Possible problems that arise in face decomposition task could be occlusions and occurrences of facial hair or glasses. Furthermore, defining the feature set is difficult, because features should be descriptive and possibly not correlated.

Feature extraction methods:

- Pantic and Rothkrantz [8] selected a set of facial points from frontal and profile face images. The expression is measured by a distance between position of those points in the initial image (neutral face) and peak image (affected face).
- Essa and Pentland [11] proposed temporal approach to the problem of facial expression analysis. They used the multiscale coarse-to-fine Kalman filtering.
  The facial motion is represented by spatio-temporal energy templates.
- Black and Yacoob [14] introduced local parametric models of image motion based on optical flow information. Models could describe horizontal and vertical translation, divergence and curl.
- Edwards et al. [15] used Active Appearance Model which is statistical model of shape and gray scale information. Relationships between AAM displacement and the image difference is analyzed for expression detection. Proposed system operates on static images.
- Cohn et al.[16] developed geometric feature based system in which the optical flow algorithm is performed only in 13x13 pixel regions surrounding facial landmarks.
- Zeng et al. [17] used data extracted by the 3D face tracker called Piecewise Bezier Volume Deformation Tracker [33]. The system was designed to recognize spontaneous emotions so three-dimensional tracking was beneficial.
- Littlewort et al. [18] proposed system which uses only appearance features to describe facial expressions. Facial texture is measured by Gabor waveletes.
- Shan et al. [19] investigated the Local Binary Pattern method for texture encoding in facial expression description. Two methods of feature extraction were proposed. In the first one, features are extracted from fixed set of patches and in the second method from most probable patches found by boosting.

2.4 Expression Recognition

The last part of the FER system is based on machine learning theory, precisely it is the classification task. The input to the classifier is a set of features which were retrieved from face region in the previous stage. The set of features is formed to describe the facial expression. Classification requires supervised training, so the training set should consist of labeled data.
Once the classifier is trained, it can recognize input images by assigning them a particular class label. The most commonly used facial expressions classification is done both in terms of Action Units, proposed in Facial Action Coding System and in terms of universal emotions: joy, sadness, anger, surprise, disgust and fear. There are a lot of different machine learning techniques for classification task, namely: K-Nearest Neighbors, Artificial Neural Networks, Support Vector Machines, Hidden Markov Models, Expert Systems with rule based classifier, Bayesian Networks or Boosting Techniques (Adaboost, Gentleboost).

Three principal issues in classification task are: choosing good feature set, efficient machine learning technique and diverse database for training. Feature set should be composed of features that are discriminative and characteristic for particular expression. Machine learning technique is chosen usually by the sort of a feature set. Finally, database used as a training set should be big enough and contain various data. Approaches described in the literature are presented by categories of classification output.

Action Units classification:

- Pantic and Rothkrantz [8] introduced the expert system with rule based classifier, which can recognize 31 action units with accuracy rate of 89%.
- Cohn et al. [16] performed recognition with use of discriminant functions. Proposed method can distinguish 8 AUs and 7 AUs combinations. Tests were performed on 504 image sequences of 100 subjects and the system obtained accuracy rate of 88%.

Emotions classification:

- Huang and Huang [7] detected motion by analysis of difference image between neutral and expression image. The minimum distance classifier is used for recognition of six basic emotions. Recognition result is 84.5%
- Kobayasi and Hara [9] used 234x50x6 neural back propagation network for recognition of 6 basic emotions. The achieved recognition accuracy is 85%.

- Zeng et al.[17] used Support Vector Data Description (SVDD) with Kernel Whitenning to avoid influence of nonhomogeneous data distributions in input space. The accuracy of a system is approximately 83%.
- Littlewort et al. [18] introduced method called AdaSVM where facial expression is represented by Gabor wavelet coefficients. Firstly, the Adaboost method is applied and the most probable features are chosen by the highest value of frequencies. Then, reduced expression representation is the input to SVM classifier. System obtains 97% accuracy of generalization to novel subjects.
- Pantic and Rothkranz [8] in their Expert System implemented also the rule based classification of emotions with use of previously recognized action units. For example, happiness is a combination of AU6, AU12, AU16, AU25. Blended emotions are allowed. The result can be: 75% of happiness if only AU6,AU12, AU16 occurred. Accuracy achieved by a system is 91%.

2.5 Recent advances

Apart from principal methods used in FER systems there were some advances made in the field of facial expression analysis recently. Facial expressions are recognized at higher semantic level. Expressions could be classified into categories such as confusion, boredom, agreement, frustration, pain etc. The example of such approach could be fatigue detection proposed by Ji et al. [20] or pain detection proposed by Littlewort et al. [21]. Additionally, more pressure is put on recognition of spontaneous emotions. Some systems are designed to divide emotions into posed or spontaneous categories to recognize if emotion was genuine or fake. Such functionality was proposed by Valstar et al. [22] for genuine smile detection. What is more, head motions or body gestures are also studied in order to describe human affective states, especially with use of three-dimensional tracking. For instance, Gunes et al. [23] examined the significance of body movements in affective states analysis. Some efforts were also done in context-dependent interpretation of facial expressions, among the others by Fasel et al. [24]. Another improvement in the area of feature extraction could be found in the work by Valstar et al. [25] in which expression is described by temporal dynamics parameters such as speed, intensity, duration and co - occurrence of facial muscle activations.

2.6 Applications

Huge amount of different information is encoded in facial movements. Observing someone's face we can learn about his/her:

- affective state, connected with emotions like fear, anger and joy and moods such as euphoria or irritation
- cognitive activity (brain activity), which can be perceived as concentration or boredom
- personality features like sociability, shyness or hostility
- truthfulness using analysis of micro-expressions to reveal concealed emotions
- psychological state giving information about some disorders helpful with diagnosis of depression, mania or schizophrenia.

Due to the variety of information visible on human face, facial expression analysis has applications in different fields of science and life.

Firstly, teachers uses facial expression analysis to adjust the difficulty of the exercise and learning pace on a base of feedback visible on students faces. Virtual tutor in e-learning proposed by Amelsvoort and Krahmer [26] provides student with suitable content and adjusts the complexity of courses or tasks by the information obtained from student's face.

Another application of FERS is in the field of business where the measurement of people's satisfaction or dissatisfaction is very important. Usage of this application can be found in many marketing techniques where information is gathered from customers by surveys. The great opportunity to conduct the surveys in the automatic way could be able by using customers' facial expressions as a level of their satisfaction or dissatisfaction [3]. Moreover, prototype of Computerized Sales Assistant, proposed by Shergill et al. [27] selects the suitable marketing and sales methods by the response deducted from customers' facial expressions.

Facial behavior is also studied in medicine not only for psychological disorder diagnosis but also to help people with some disabilities. Example of it could be the system proposed by Pioggial et al. [28], that helps autistic children to improve their social skills by learning how to recognize emotions. Facial expressions could be also used for surveillance purposes like in prototype developed by Hazelhoff et al. [29]. Suggested system automatically detects discomfort of newborn babies by recognition of 3 behavioral states: sleep, awake and cry.

Additionally, facial expression recognition is widely used in human robot and human computer interaction. Kazi et al. [30] proposed Intelligent Robotic Assistant for people with disabilities based on multimodal HCI. Another example of human computer interaction systems could be system developed by Zhan et al. [31] for automatic update of avatar in multiplayer online games.

3 Proposed system

The goal of this project was to design and implement the facial expression recognition system. On a basis of the extensive study of different approaches to the problem of face action representation, appropriate algorithms were selected for each stage of a system.

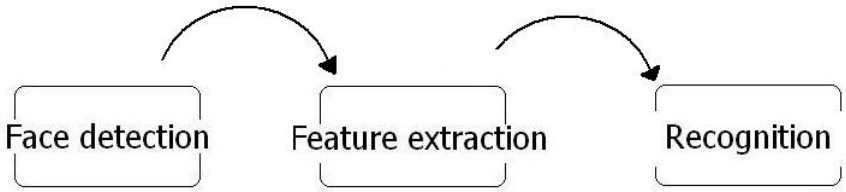

Fig. 3: The structure of FER system

The proposed system is built in traditional manner and consists of 3 stages: face detection and tracking, face expression representation and expression recognition (Fig. 3). System operates on both static images and image sequences. Static images are used in training and testing procedures but the interaction with a system is designed for video analysis.

This chapter includes the description of all three stages of a system. Algorithms used at each stage will be explained from theoretical aspect. Next, the implementation details will be mentioned and the system's behavior will be illustrated.

3.1 Face detection and tracking

First part of my system is module for face detection and landmark localization in the image. The algorithm for face detection is based on work by Viola and Jones [13]. In this approach image is represented by a set of Haar-like features. Possible types of features are two-, three- and four-rectangular features (Fig. 4).

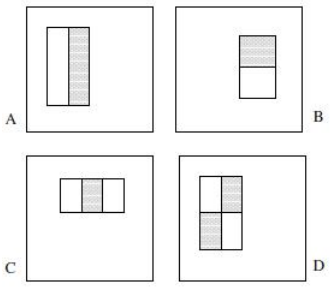

Fig. 4: Examples of Haar-like features [13]

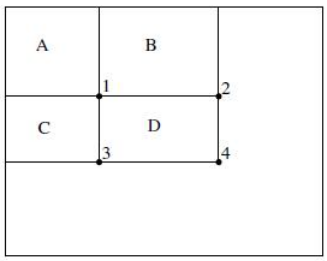

Fig. 6: Integral image concept [13]

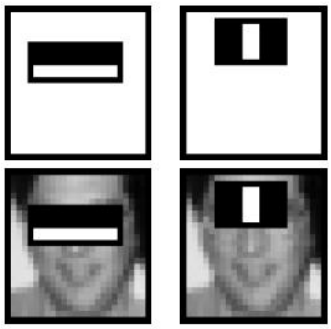

Fig. 5: Haar-like features detected on a face [13]

Feature value is calculated by subtracting sum of the pixels covered by white rectangle from sum of pixels under gray rectangle. Two rectangular features detect contrast between two vertically or horizontally adjacent regions. Three rectangular features detect contrasted region placed between two similar regions and four rectangular features detect similar regions placed diagonally (Fig. 5).

Input image is transformed into integral image in which each pixel is a sum of all pixels above and to the left.

$$ii(x, y) = \sum_{x' \le x, y' \le y} i(x', y') \quad where,$$

$$ii(x, y) - integral\ image, \quad i(x, y) - input\ image$$

This is computed in one pass, thus feature can be computed rapidly because the value of each rectangle requires only 4 pixel references (Fig. 6).

Having the representation of the image in rectangular features, the classifier needs to be trained to decide if the image contains searched object (face) or not. The number of features is much higher than the number of pixels in the original image. However, it was proven that even a small set of well-chosen features can build a strong classifier. That is why, the Adaboost algorithm was used for training. Each step selects the most discriminative feature which separates positive and negative examples in the best way.

The method is widely used in area of face detection. However, it can be trained to detect any object. What is more, this algorithm is quick and efficient and could be used in real-time applications. In proposed system, the algorithm is applied for face, eyes and mouth localization with use of already trained classifiers from OpenCV library.

The face detection procedure includes some steps which are consecutively performed on the input image. Procedure flow is illustrated by the output of each function called on input image.
Firstly, the classifier trained for face detection searches for a face in the image (Fig. 7). In case when face is not found in the image, further processing is omitted and system returns appropriate error message.

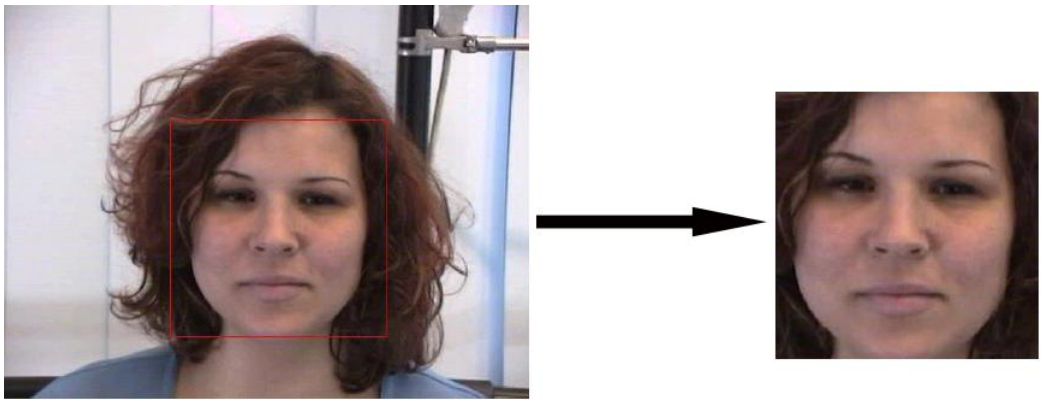

Fig. 7: Face detection procedure

If the face is located, the classifiers for eye detections are employed only on the upper part of the face. The left and right eyes are detected separately – in left and right upper face regions (Fig. 8). Finally, the mouth region is located with the fourth classifier which searches in the lower part of the face. The search area of facial elements detectors is narrowed in order to improve the time efficiency of the algorithm.

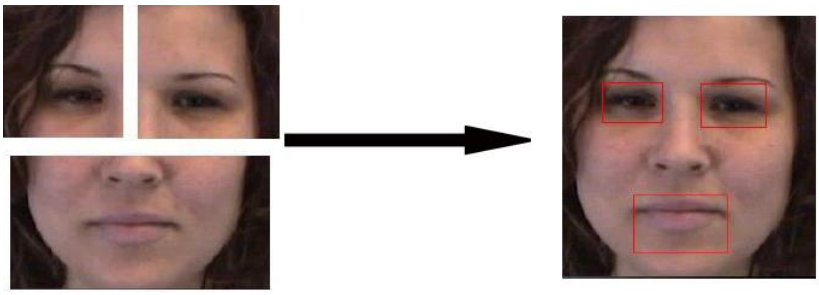

Fig. 8: Face elements localization

Having locations of the face and facial landmarks, the face representation can be formed. If there are more faces detected in the image, the algorithm takes the biggest one for further processing.

3.2 Feature Extraction

As mentioned in the previous chapter, two main approaches are used to describe the expression. Geometric or appearance features can be used either separately or in combination.
In geometric feature based systems the face is represented by a set of facial points which are tracked. Deformations between neutral state and current frame are parameters of facial action.
This approach requires reliable methods for points detection and tracking which are difficult to obtain. Appearance based methods measure the appearance changes which are mainly based on texture analysis. Although, Gabor filters are proved to be powerful in face expression analysis, they are time and memory consuming. Due to that fact, another method called Local Binary Patterns (LBP) gains more popularity in facial texture analysis.

Local Binary Patterns were introduced by Olaja et al. [32] as an effective texture descriptors. Input image is transformed into LBP representation by sliding window technique where value of each pixel in the neighborhood is thresholded with value of central pixel (Fig. 9). Central pixel is encoded with LBP code (binary or decimal) in corresponding LBP image pixel. Binary codes are so called 'micro-textons' that represent texture primitives such as curved edges, flat or convex areas.

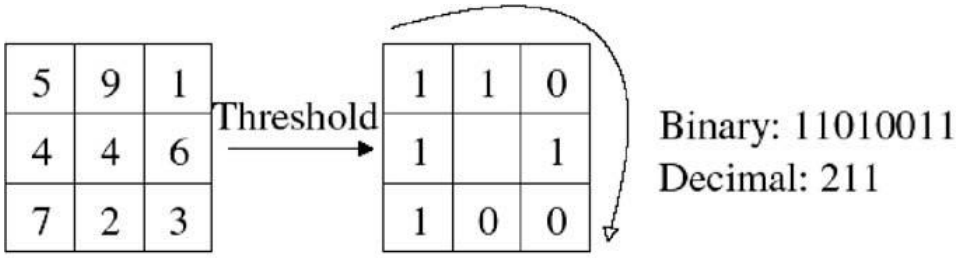

Fig. 9: LBP encoding [20]

Basic version of LBP uses 3x3 sliding window to code the texture. Recently, the operator has been extended to different sizes and shapes (circular neighborhood). The size of the neighborhood directly influences the range of code values. Having operator of size P and radius R, the range of possible codes are from 0 to 2^P. The image texture is described by a 2^P bin histogram of corresponding LBP image.

Encoding facial texture features can be done in holistic or analytic way. Holistic approach encodes whole face region with LBP features. The disadvantage of this approach is that spatial information about texture is lost. In the second method face region is divided into a grid of patches and each patch is transformed to LBP histogram separately.

The latter method encodes the spatial information about the face texture. However, many patches consist of data that is not affected by expression like hair or neck parts. That is why, in my system the LBP operator is applied on two regions that are highly involved in face activity. Those regions are forehead-eyes area and chin-mouth-cheeks area (Fig. 10). Regions are estimated with regard to face representation created in the first module of my system.

Fig. 10: Regions involved in expressions analysis

Before features can be extracted, particular face region need to be normalized. All regions are rescaled to the same size, namely: 90x48 for upper region and 72x48 for lower region. Next, regions are divided into grids of sizes: 4x4 in lower part and 5x4 in upper part of the face. (Fig. 11).

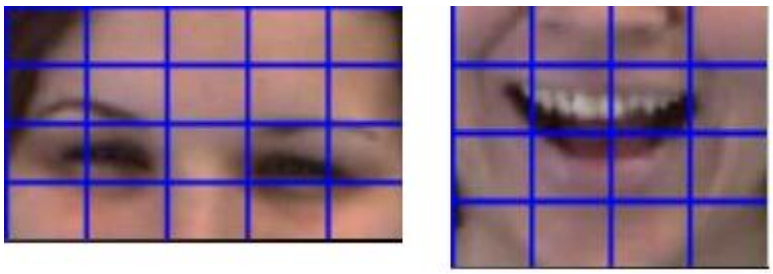

Fig. 11: Face regions grids

Each window from a grid is encoded with LBP histogram. Basic version of LBP operator was implemented in the proposed system so the amount of bins in the histogram is $2^8 = 256$. Feature vector that represents particular emotion consists of 36 histograms (Fig. 12). Thus, each expression is described by 9216 features.

Fig. 12: Visualization of feature set

3.3 Expression Recognition

The last stage of my system is devoted to facial expressions recognition. This task requires classifier training with a set of images with particular emotions displayed. For the purpose of training I obtained two facial expression databases.

First one is the FG-NET Facial Expression and Emotion Database [34] which consists of MPEG video files with spontaneous emotions recorded. Database contains examples gathered from 18 subjects (9 female and 9 male). Proposed system was trained with captured video frames in which the displayed emotion is very representative. The training set consists of 675 images of seven states neutral and emotional (surprise, fear, disgust, sadness, happiness and anger).

Second database is the Cohn-Kanade Facial Expression Database [35] and contains 486 image sequences displayed by 97 posers. The sequence displays the emotion from the start to the peak, however, only the last image of a sequence is used for training. What is more, the database is labeled with Action Units which is not applicable in my system. That is why, the AU labels needed to be translated into emotion categories, according to the rules provided with the database. The training and testing sets formed from Cohn-Kanade database contain 518 images divided into seven classes: neutral, surprise, fear, disgust, sadness, happiness and anger.

Having appropriate training sets, the procedure of classifier training could be performed.
In proposed system the Support Vector Machine with Radial Based Kernel Function is used as a classifier. The Support Vector Machine is an adaptive learning system which receives labeled training data and transforms it into higher dimensional feature space. Then separating hyperplane with respect to margin maximization is computed to determine the best separation between classes. The greatest advantage of SVM is that even with small set of training data it has good performance in generalization.

Developed FER system was trained and tested with each database separately. In FEED database, each person shows certain emotion three times and usually first two samples are used in training and the third one in testing. Similarly, 330 images captured from the third trial videos are used as a testing set in proposed system. In Cohn-Kanade database peak images from sequences are split into training set (518 images) and testing set (53 images).

System's performance was measured with accuracy rate, that is the proportion of properly classified images to all images in the test set. System trained with a FEED database can recognize emotions with accuracy rate of 71%. However, the system trained with Cohn-Kanade database is able to recognize emotions with 77% of accuracy (Table 3).

Additionally, the recognition results were presented by confusion matrix, which not only shows accuracy of each emotion recognition but also indicates the emotions with which the certain one is confused. Dealing with spontaneous expression recognition, the emotions are often confused, probably because the differences between them are quite subtle. The best recognition rate was obtained for sadness (91%), the worst one for anger (51%) which was usually confused with sadness (Table 2).

%	neutral	happiness	sadness	surprise	anger	fear	disgust
neutral	60	0	20	0	0	20	0
happiness	10	90	0	0	0	0	0
sadness	14	0	86	0	0	0	0
surprise	0	0	0	100	0	0	0
anger	0	20	20	0	60	0	0
fear	11	11	0	11	11	56	0
disgust	0	0	0	0	1	0	100

Table 1: Confusion matrix of testing with Cohn-Kanade Database

%	neutral	happiness	sadness	surprise	anger	fear	disgust
neutral	61	0	33	0	0	6	0
happiness	3	78	3	10	0	3	3
sadness	3	1	91	0	4	1	0
surprise	3	3	3	76	0	12	3
anger	0	2	36	0	51	4	7
fear	2	2	28	4	0	64	0
disgust	0	2	20	0	7	7	64

Table 2: Confusion matrix of testing with FG-NET FEED

System trained with Cohn-Kanade database performs much better and the best accuracy obtained for disgust and surprise is 100%. This could be caused by the fact that texture in both those emotions is distinct. That is why, in expressions which involve less changes in texture like neutral and fear have the worst recognition rates, namely 60% and 56% (Table 1). Furthermore, CK database consists of posed emotions and due to that, expressions are usually exaggerated and easier to differentiate.

What is more, the system was compared to other systems proposed in literature (Table 3). In order to make the comparison reliable, only the systems which use the same database and recognition type (7-class classification) were mentioned.

System	Accuracy rate(%)	Database
Shan et al. [19]	89	CK
Zhan et al.[31]	82	FEED
proposed FERS	77	CK
	70	FEED

Table 3: Accuracy rates of exemplar FERS

Shan et al. [19] evaluated the performance of LBP features as facial expression descriptors and they obtained the result of 89% by use of SVM with RBF Kernel trained with CK database. My system uses the similar technique but the result is lower (77%) what could be caused by difference in LBP code range. Shan et al. used the 59 bin histogram according to the theory that some of the patterns are uniform. Furthermore, proposed system has lower accuracy rate than system developed in work by Zhan et al.[31] in which the Gabor wavelets are used as texture descriptors. Although, the results achieved by proposed system are lower than in compared systems, they can be accepted with regard to the fact, that the best accuracy rate obtained for 7-class recognition is 89%.

Conclusion

The aim of this thesis was to explore the area of facial expression recognition. Beginning with the psychological motivation for facial behavior analysis, this field of science has been extensively studied in terms of application and automation. Manual face analysis used by psychologists was quickly replaced by suitable computer software. A wide variety of image processing techniques was developed to meet the facial expression recognition system requirements. However, there are still many challenges and problems to solve in such systems, especially in the area of their performance and applicability improvement.

Apart from theoretical background, this work provides the design and implementation of Facial Expression Recognition System. Proposed system was developed to process the video of facial behavior and recognize displayed actions in terms of six basic emotions. Major strengths of the system are full automation as well as user and environment independence. Even though the system cannot handle occlusions and significant head rotations, the head shifts are allowed. Additionally, the recognition results are quite promising with regard to the fact that only appearance features were used to encode the facial expression.

In the future work, I would like to focus on improving the recognition rate of my system. One of the possible solutions could be adding the motion information to the expression representation. The action could be described by geometrical features as well as appearance features. Finally, I would like to improve the time efficiency of my system in order to make it appropriate to use in different applications.

References

[1] A. Mehrabian, "Communication without Words", Psychology Today, Vol. 2, No. 4, p. 53-56, 1968.

[2] P. Ekman, T. Huang, T. Sejnowski, J. Hager, "Final Report To NSF of the Planning Workshop on Facial Expression Understanding", 1992. accessed on:"http://face-and-emotion.com/dataface/nsfrept/nsf_contents.html"

[3] Z. Kasiran, S. Yahya, "Facial Expression as an Implicit Customers Feedback and the Challenges", Computer Graphics, Imaging and Visualisation, p. 377-381, 2007.

[4] http://2wanderlust.files.wordpress.com/2009/03/picture-2.png

[5] M. Pantic, L. Rothkrantz, "Automatic Analysis of Facial Expressions: The State of the Art", IEEE Transactions On Pattern Analysis and Machine Intelligence, Vol. 22, No. 12, 2000.

[6] Z. Zeng, .M. Pantic, G. I. Roisman, T. S. Huang, „A Survey of Affect Recognition Methods: Audio,Visual, and Spontaneous Expressions", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 31, No. 1, 2009.

[7] C.L. Huang and Y.M. Huang, "Facial Expression Recognition Using Model-Based Feature Extraction and Action Parameters Classification," J. Visual Comm. and Image Representation, Vol. 8, No. 3, p. 278-290, 1997.

[8] M. Pantic and L. Rothkrantz, "Expert System for Automatic Analysis of Facial Expression", Image and Vision Computing J., Vol. 18, No. 11, p. 881-905, 2000.

[9] H. Kobayashi and F. Hara, "Facial Interaction between Animated 3D Face Robot and Human Beings," Proc. Int'l Conf. Systems, Man, Cybernetics, p. 3,732-3,737, 1997.

[10] S. Kimura and M. Yachida, "Facial Expression Recognition and Its Degree Estimation", Proc. Computer Vision and Pattern Recognition, p. 295-300, 1997.

[11] I. Essa and A. Pentland, "Coding, Analysis Interpretation,Recognition of Facial Expressions", IEEE Trans. Pattern Analysis and Machine Intelligence, Vol.19, No. 7, p. 757-763, July 1997.

[12] H. Rowley, S. Baluja, T. Kanade, "Neural Network-Based Face Detection", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 20, No. 1, p. 23 – 38,1998.

[13] P. Viola and M. J. Jones, "Robust real-time object detection", International Journal of Computer Vision, Vol. 57, No. 2, p.137–154, 2004.

[14] M.J. Black and Y. Yacoob, "Recognizing Facial Expressions in Image Sequences Using Local Parameterized Models of Image Motion," lnt'l J. Computer Vision, Vol. 25, No. 1, p. 23-48, 1997.

[15] G.J. Edwards, T.F. Cootes, and C.J. Taylor, "Face Recognition Using Active Appearance Models," Proc. European Conf. Computer Vision, Vol. 2, p. 581-695, 1998.

[16] J.F. Cohn, A.J. Zlochower, J.J. Lien, and T. Kanade, "Feature-Point Tracking by Optical Flow Discriminates Subtle Differences in Facial Expression," Proc. Int'l Conf. Automatic Face and Gesture Recognition, p. 396-401, 1998.

[17] Z. Zeng, Y. Fu, G. I. Roisman, Z. Wen, Y. Hu and T. S. Huang, "Spontaneous Emotional Facial Expression Detection", Journal of Multimedia, Vol. 1, No. 5, p. 1-8, 2006.

[18] G.C. Littlewort, M.S. Bartlett, J. Chenu, I. Fasel, T. Kanda, H. Ishiguro, J.R. Movellan,"Towards social robots: Automatic evaluation of human-robot interaction by face detection and expression classification", Advances in Neural Information Processing Systems, Vol 16, p. 1563-1570, 2004.

[19] C.Shan, S. Gong, P. McOwan, “Facial expression recognition based on Local Binary Patterns: A comprehensive study”, Image and Vision Computing, Vol. 27, p. 803-816, 2009.

[20] Q. Ji, P. Lan, and C. Looney, “A Probabilistic Framework for Modeling and Real-Time Monitoring Human Fatigue”, IEEE Systems, Man, and Cybernetics Part A, Vol. 36, No. 5, p. 862-875, 2006.

[21] G.C. Littlewort, M.S. Bartlett, and K. Lee, “Faces of Pain: Automated Measurement of Spontaneous Facial Expressions of Genuine and Posed Pain”, Proc. Ninth ACM Int’l Conf. Multimodal Interfaces, p. 15-21, 2007.

[22] M.F. Valstar, H. Gunes, and M. Pantic, “How to Distinguish Posed from Spontaneous Smiles Using Geometric Features”, Proc. Ninth ACM Int’l Conf. Multimodal Interfaces, p. 38-45, 2007.

[23] H. Gunes and M. Piccardi, “Affect Recognition from Face and Body: Early Fusion versus Late Fusion”, Proc. IEEE Int’l Conf.Systems, Man, and Cybernetics, p. 3437-3443, 2005.

[24] B. Fasel, F. Monay, and D. Gatica-Perez, “Latent Semantic Analysis of Facial Action Codes for Automatic Facial Expression Recognition,” Proc. Sixth ACM Int’l Workshop Multimedia Information Retrieval, p. 181-188, 2004.

[25] M. Valstar, M. Pantic, and I. Patras, “Motion History for FacialAction Detection from Face Video”, Proc. IEEE Int’l Conf. Systems,Man and Cybernetics, vol. 1, p. 635-640, 2004.

[26] M. van Amelsvoort, E. Krahmer, "Appraisal of Children's Facial Expressions while Performing Mathematics Problems", Proceedings of the 31st Annual Meeting of the Cognitive Science Society, p.1698-1703, 2009.

[27] G. Shergill, A. Sarrafzadeh, O. Diegel, A. Shekar, "Computerized Sales Assistants: The Application of Computer Technology to Measure Consumer Interest – a Conceptual Framework", Journal of Electronic Commerce Research, Vol. 9, No. 2, p.176-191, 2008.

[28] G. Pioggial, M.L. Sical, M. Ferrol, R. Jgliozzi, F. Muratori, A. Ahluwalia, D. De Rossi, "Human-Robot Interaction in Autism: FACE, an Android-based Social Therapy", 16th IEEE International Conference on Robot & Human Interactive Communication, 2007.

[29] L. Hazelhoff, J. Han, S. Bambang-Oetomo, P. de With, "Behavioral State Detection of Newborns Based on Facial Expression Analysis", Advanced Concepts for Intelligent Vision Systems, p. 698–709, 2009.

[30] Z. Kazi, S. Chen, M. Beitler, D. Chester, R. Foulds, "Multimodal HCI for Robot Control: Towards an Intelligent Robotic Assistant for People with Disabilities", AAAI Technical Report p.46-53, 1996.

[31] C. Zhan, W. Li, P. Ogunbona,F. Safaei, A Real-Time Facial Expression Recognition System for Online Games, International Journal of Computer Games Technology, Volume 2008.

[32] T. Ojala, M. Pietikainen, T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture with local binary patterns", IEEE Transactions on Pattern Analysis and Machine Intelligence Vol. 7, No. 7, p. 971-987, 2002.

[33] H. Tao and T.S. Huang, "Explanation-Based Facial Motion Tracking Using a Piecewise Bezier Volume Deformation Mode", Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition, Vol. 1, p. 611-617, 1999.

[34] F. Wallhoff, "Facial Expressions and Emotion Database", Technische Universität München, 2006, http://www.mmk.ei.tum.de/~waf/fgnet/feedtum.html.

[35] T. Kanade, J. F. Cohn, Y. Tian, "Comprehensive database for facial expression analysis", Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition, Grenoble, France, p.46-53, 2000.

Appendix 1

modules.hpp

```
/**
* author: Ewa Piatkowska
* header file for 3 modules of FER system
* Face detection
* Expression description
* Training & Recognition
*/

#ifndef modules_hpp
#define modules_hpp

#include "helpers.hpp"
#include "cv.h"
#include "highgui.h"
#include "ml.h"
using namespace cv;

class Landmark
{
public:
	Landmark();
	Landmark(CvRect box);

	CvRect bbox;
	bool isEmpty();
	int getX();
	int getY();
	CvRect getRect();
};
class Eye :public Landmark
{
public:
	Eye(){}
	Eye(CvRect box){}
	~Eye(){}
};
class Eyebrow :public Landmark
{
public:
	CvPoint left;
	CvPoint center;
	CvPoint right;

	Eyebrow(){}
	Eyebrow(CvRect box){}
	~Eyebrow(){}
};
class Mouth :public Landmark
{
public:
	CvPoint left;
	CvPoint center;
```

```
        CvPoint upperCenter;
        CvPoint lowerCenter;
        CvPoint right;

        Mouth(){}
        Mouth(CvRect box){}
        ~Mouth(){}
};
class Face :public Landmark
{
public:
        CvRect bbox;
        CvRect upperface;
        CvRect lowerface;
        Eye lefteye;
        Eye righteye;
        Mouth mouth;
        Eyebrow lefteyebrow;
        Eyebrow righteyebrow;

/**methods**/
        Face(){}
        Face(CvRect box){}
        ~Face(){}

        void drawBox(IplImage* image, CvRect box);
        void drawPoints(IplImage *image);
        void drawElements(IplImage *image);
};

/********detector***********/
class FaceDetection
{
public:
        Face face;
private:
        CvMemStorage *buffer;
        CvHaarClassifierCascade *faceCascade, *reyeCascade, *leyeCascade, *mouthCascade;
        CvPoint currentROIlocation;
        IplImage *image;

public:
        FaceDetection(IplImage *image);
        ~FaceDetection();
        Face getFace();
        IplImage* getImage();
        void calculatePoints();
        bool detectElements();
        void setRegions();
private:
        bool detectFace();
        void detectEyes();
        CvRect setBrow(CvRect box);
        void detectMouth();
        void setEyebrows();
        void setAbsoluteCoordinates(CvRect &r);
        void setAbsoluteCoordinates(CvPoint &p);
        void setCurrentROIlocation(int x, int y);
        CvSeq* getMax(CvSeq * contours, double boxarea);
};
```

```
/*********extraction**********/
class FeatureExtraction
{
public:
        FeatureExtraction();
        FeatureExtraction(IplImage* upper, IplImage* lower);
        ~FeatureExtraction();

        IplImage *upper;
        IplImage *lower;
        int indx;
        float feature_vector[36*256];
public:
        void normalize(IplImage* upper, IplImage* lower);
        void calculateLBP();
        void setLBPGrid(IplImage *img, int width, int height);
};
/****Multiclass Training****/
class MultiTrain
{
public:
        MultiTrain();
        ~MultiTrain();

        CvSVM SVM;
        CvMat *trainData;
        CvMat *labels;
        CvTermCriteria criteria;
        CvSVMParams params;
        double ACC;

        void createDataSet(string inputdir, string outputdir);
        void loadDataSet(string filename);
        void trainModel(string outputdir);
        void loadModel(string filename);
        int  getPrediction(IplImage *image);
        void testModel(string filename);
        void createConfusionMatrix(string filename);
        void calculateTrainDataCount(int tab[]);
        void prepareSets(int class_counts[], int counts[], int part, int parts, CvMat *traindata, CvMat *trainlabels,
CvMat *testdata, CvMat *testlabels);
        void performCrossValidation(int parts);

private:
        void processData(string path, int i);
};

/**cvSeq comparison function**/
static int comp_func(const void* _a, const void* _b, void* userdata);
static int comp_func_x(const void* _a, const void* _b, void* userdata);

#endif
```

modules.cpp

```
/**
* author: Ewa Piatkowska
*/

#include "helpers.hpp"
#include "modules.hpp"
#include "lbp.hpp"

/****landmark****/
Landmark::Landmark()
{
	this->bbox = cvRect(0,0,0,0);
}
Landmark::Landmark(CvRect box)
{
	this->bbox = box;
}
bool Landmark::isEmpty()
{
	if(this->bbox.height == 0 || this->bbox.width == 0)
		return true;
	return false;
}
int Landmark::getX()
{
	return this->bbox.x;
}
int Landmark::getY()
{
	return this->bbox.y;
}
CvRect Landmark::getRect()
{
	return this->bbox;
}
/******Face******/
void Face::drawBox(IplImage* image, CvRect box)
{
	cvRectangle(image, cvPoint(box.x, box.y), cvPoint(box.x+box.width, box.y+box.height), CV_RGB(255,0,0),
1, 8, 0);
}
void Face::drawElements(IplImage *image)
{
	this->drawBox(image, this->bbox);
	this->drawBox(image, this->lefteye.bbox);
	this->drawBox(image, this->righteye.bbox);
	this->drawBox(image, this->mouth.bbox);
}
/***********Face Detection******************/
FaceDetection::FaceDetection(IplImage *image)
{
	this->buffer = cvCreateMemStorage(0);
	char *face = "../haarcascades/haarcascade_frontalface_default.xml";
	char *eye_left= "../haarcascades/haarcascade_mcs_lefteye.xml";
	char *eye_right = "../haarcascades/haarcascade_mcs_righteye.xml";
	char *mouth = "../haarcascades/haarcascade_mcs_mouth.xml";
	this->faceCascade = ( CvHaarClassifierCascade*  )cvLoad( face, 0, 0, 0);
```

```
	this->leyeCascade = ( CvHaarClassifierCascade*  )cvLoad( eye_left, 0, 0, 0);
	this->reyeCascade = ( CvHaarClassifierCascade*  )cvLoad( eye_right, 0, 0, 0);
	this->mouthCascade = ( CvHaarClassifierCascade*  )cvLoad( mouth, 0, 0, 0);
	this->currentROIlocation = cvPoint(0,0);
	this->image = image;
	this->face = Face();
}
FaceDetection::~FaceDetection()
{
	cvReleaseHaarClassifierCascade( &faceCascade);
	cvReleaseHaarClassifierCascade( &leyeCascade);
	cvReleaseHaarClassifierCascade( &reyeCascade);
	cvReleaseHaarClassifierCascade( &mouthCascade);
	cvReleaseMemStorage( &buffer);
}
Face FaceDetection::getFace()
{
	return this->face;
}
IplImage* FaceDetection::getImage()
{
	return this->image;
}
bool FaceDetection::detectElements()
{
	if(! this->detectFace()) return false;
	this->detectEyes();
	this->detectMouth();
	this->setEyebrows();
	this->setCurrentROIlocation(0,0);
	return true;
}
bool FaceDetection::detectFace()
{
	CvSeq *faces = cvHaarDetectObjects(this->image, faceCascade, buffer, 1.1, 3, 0, cvSize(30,30));
	if(!faces->total) return false;
	else
	{
		/**get the biggest detected face**/
		cvSeqSort(faces, comp_func, 0);
		CvRect *r = (CvRect*) cvGetSeqElem(faces, 0);
		this->face.bbox = *r;
		cvClearMemStorage(this->buffer);
	}
	return true;
}
void FaceDetection::detectEyes()
{
	/*left eye*/
	cvSetImageROI(this->image, cvRect(this->face.bbox.x, this->face.bbox.y, this->face.bbox.width/2, (this-
>face.bbox.height*2/3)));
	this->setCurrentROIlocation(this->face.bbox.x, this->face.bbox.y);

	CvSeq *eyes = cvHaarDetectObjects(this->image, this->leyeCascade, this->buffer, 1.1, 3,0, cvSize(5,5));
	cvSeqSort(eyes, comp_func, 0);
	if( eyes->total != 0)
	{
		CvRect *left = (CvRect*) cvGetSeqElem( eyes, 0);
		this->setAbsoluteCoordinates(*left);
		this->face.lefteye.bbox = *left;
```

```
		}
		cvClearMemStorage(this->buffer);
		cvResetImageROI(this->image);


		/*right eye*/
		cvSetImageROI(this->image, cvRect(this->face.bbox.x+(this->face.bbox.width/2), this->face.bbox.y, this->face.bbox.width/2, (this->face.bbox.height*2/3)));
		this->setCurrentROIlocation(this->face.bbox.x+(this->face.bbox.width/2), this->face.bbox.y);

		CvSeq *reyes = cvHaarDetectObjects(this->image, this->reyeCascade, this->buffer, 1.1, 3,0, cvSize(5,5));
		cvSeqSort(reyes, comp_func, 0);
		if( reyes->total != 0)
		{
			CvRect *right = (CvRect*) cvGetSeqElem( reyes, 0);
			this->setAbsoluteCoordinates(*right);
			this->face.righteye.bbox = *right;

		}
		cvClearMemStorage(this->buffer);
		cvResetImageROI(this->image);
}
void FaceDetection::detectMouth()
{
		cvSetImageROI(image, cvRect(this->face.bbox.x, this->face.bbox.y+(this->face.bbox.height/2), this->face.bbox.width, (this->face.bbox.height/2)));
		setCurrentROIlocation(this->face.bbox.x, this->face.bbox.y+(this->face.bbox.height/2));

		CvSeq *mouth = cvHaarDetectObjects(image, mouthCascade, buffer, 1.1, 3,0, cvSize(1,1));
		if(mouth->total)
		{
			cvSeqSort(mouth, comp_func, 0);
			CvRect *r = (CvRect*) cvGetSeqElem( mouth, 0);
			this->setAbsoluteCoordinates(*r);
			this->face.mouth.bbox = *r;
		}
		cvClearMemStorage(this->buffer);
		cvResetImageROI(this->image);
}
void FaceDetection::setEyebrows()
{
		this->face.lefteyebrow.bbox = this->setBrow(this->face.lefteye.bbox);
		this->face.righteyebrow.bbox = this->setBrow(this->face.righteye.bbox);
}
CvRect FaceDetection::setBrow(CvRect box)
{
		int x = box.x - box.width/3;
		int y = box.y - box.height*3/2;
		int width = box.width*5/3;
		int height = box.height*2;
		return cvRect(x, y, width, height);
}
void FaceDetection::setRegions()
{
		//upper
		int x = this->face.lefteyebrow.bbox.x;
		int y = this->face.lefteyebrow.bbox.y;
		int width = this->face.lefteyebrow.bbox.width + this->face.righteyebrow.bbox.width;
		int height = this->face.lefteyebrow.bbox.height + this->face.lefteye.bbox.height;
```

```
		this->face.upperface = cvRect(x, y, width, height);
		//lower
		x = this->face.lefteye.bbox.x;
		width = (this->face.righteye.bbox.x+this->face.righteye.bbox.width) - this->face.lefteye.bbox.x;
		y = this->face.mouth.bbox.y - this->face.mouth.bbox.height/2;
		height = 2*this->face.mouth.bbox.height;

		this->face.lowerface = cvRect(x,y,width,height);
}
void FaceDetection::setCurrentROIlocation(int x, int y)
{
		currentROIlocation.x = x;
		currentROIlocation.y = y;
}
void FaceDetection::setAbsoluteCoordinates(CvRect &r)
{
		r.x += currentROIlocation.x;
		r.y += currentROIlocation.y;
}
void FaceDetection::setAbsoluteCoordinates(CvPoint &p)
{
		p.x += currentROIlocation.x;
		p.y += currentROIlocation.y;
}

/***********feature extraction ***************/
FeatureExtraction::FeatureExtraction()
{
		this->upper = 0;
		this->lower = 0;
		this->indx = 0;
}
FeatureExtraction::FeatureExtraction(IplImage* upper, IplImage* lower)
{
		this->normalize(upper, lower);
}
FeatureExtraction::~FeatureExtraction()
{
		if(this->upper) cvReleaseImage(&this->upper);
		if(this->lower) cvReleaseImage(&this->lower);
}

void FeatureExtraction::normalize(IplImage* upper, IplImage* lower)
{
		this->upper = cvCreateImage(cvSize(90,48), upper->depth, upper->nChannels);
		this->lower = cvCreateImage(cvSize(72,48), lower->depth, lower->nChannels);

		cvResize(upper, this->upper);
		cvResize(lower, this->lower);
}
void FeatureExtraction::calculateLBP()
{
		this->indx = 0;
		this->setLBPGrid(this->lower, 18, 12);
		this->setLBPGrid(this->upper, 18, 12);
}
void FeatureExtraction::setLBPGrid(IplImage *img, int width, int height)
{
		for(int i=0; i< (img->width/width); i++)
```

```
		for (int j=0; j< (img->height/height); j++)
		{
			LBP lbp;
			cvSetImageROI(img, cvRect(i*width,j*height, width, height));
			lbp.createLBP(img);
			lbp.histogram();
			lbp.fillFeatureSet(this->feature_vector, this->indx);
			this->indx+=256;
			//cvRectangle(img, cvPoint(0,0), cvPoint(width, height), cvScalar(255,0,0));
			cvResetImageROI(img);

		}
}

/******MulticlassTraining****/
MultiTrain::MultiTrain()
{
	this->trainData = 0;
	this->labels = 0;
	this->params = CvSVMParams();
	this->params.term_crit.epsilon = 1.0000000116860974e-007;
	this->params.term_crit.type = CV_TERMCRIT_EPS;
	this->params.svm_type = CvSVM::C_SVC;
	this->params.kernel_type = CvSVM::RBF;
	this->params.gamma = 3.0000000000000001e-006;
	this->params.C = 20;
}
MultiTrain::~MultiTrain()
{
	if(this->trainData)		cvReleaseMat(&this->trainData);
	if(this->labels)	cvReleaseMat(&this->labels);
}
void MultiTrain::createDataSet(string inputdir, string outputdir)
{
	/*
	0 neutral
	1 happiness
	2 sadness
	3 surprise
	4 anger
	5 fear
	6 disgust
	*/
	vector<string> neutral = vector<string>();
	vector<string> happy = vector<string>();
	vector<string> sad = vector<string>();
	vector<string> surprise = vector<string>();
	vector<string> angry = vector<string>();
	vector<string> fear = vector<string>();
	vector<string> disgust = vector<string>();

	listFiles(inputdir, "*neutr*", neutral);
	listFiles(inputdir, "*happy*", happy);
	listFiles(inputdir, "*sad*", sad);
	listFiles(inputdir, "*surpr*", surprise);
	listFiles(inputdir, "*ang*", angry);
	listFiles(inputdir, "*fear*", fear);
	listFiles(inputdir, "*disg*", disgust);

	int count = (int)(neutral.size()+happy.size()+sad.size()+surprise.size()+angry.size()+fear.size()+disgust.size());
```

```
cout<<count<<endl;
this->trainData = cvCreateMat(count, 36*256, CV_32FC1);
this->labels = cvCreateMat(count, 1, CV_32SC1);
cvZero(this->trainData);
cvZero(this->labels);
int j=0;

for(int i=0; i< (int)neutral.size(); i++)
{
        cout<<"processing image # "<<i<<endl;
        this->processData(inputdir+"/"+neutral[i], i);
        CV_MAT_ELEM(*this->labels, int, i,0) = 0;
}
j += (int) neutral.size();

for(int i=0; i< (int)happy.size(); i++)
{
        cout<<"processing image # "<<i+j<<endl;
        this->processData(inputdir+"/"+happy[i], i+j);
        CV_MAT_ELEM(*this->labels, int, i+j,0) = 1;
}
j += (int) happy.size();
for(int i=0; i< (int)sad.size(); i++)
{
        cout<<"processing image # "<<i+j<<endl;
        this->processData(inputdir+"/"+sad[i], i+j);
        CV_MAT_ELEM(*this->labels, int, i+j,0) = 2;
}
j += (int) sad.size();
for(int i=0; i< (int)surprise.size(); i++)
{
        cout<<"processing image # "<<i+j<<endl;
        this->processData(inputdir+"/"+surprise[i], i+j);
        CV_MAT_ELEM(*this->labels, int, i+j,0) = 3;
}
j += (int) surprise.size();
for(int i=0; i< (int)angry.size(); i++)
{
        cout<<"processing image # "<<i+j<<endl;
        this->processData(inputdir+"/"+angry[i], i+j);
        CV_MAT_ELEM(*this->labels, int, i+j,0) = 4;
}
j += (int) angry.size();
for(int i=0; i< (int)fear.size(); i++)
{
        cout<<"processing image # "<<i+j<<endl;
        this->processData(inputdir+"/"+fear[i], i+j);
        CV_MAT_ELEM(*this->labels, int, i+j,0) = 5;
}
j += (int) fear.size();
for(int i=0; i< (int)disgust.size(); i++)
{
        cout<<"processing image # "<<i+j<<endl;
        this->processData(inputdir+"/"+disgust[i], i+j);
        CV_MAT_ELEM(*this->labels, int, i+j,0)= 6;
}

CvFileStorage *file = cvOpenFileStorage((outputdir+"/emotrainset.xml").c_str(), 0, CV_STORAGE_WRITE);
cvWrite(file, "dataset", this->trainData);
cvWrite(file, "labels", this->labels);
```

```
	cvReleaseFileStorage(&file);
}
void MultiTrain::loadDataSet(string filename)
{
	CvFileStorage* file = cvOpenFileStorage(filename.c_str(), 0, CV_STORAGE_READ);
	this->trainData = (CvMat*)cvRead(file, cvGetFileNodeByName(file,0, "dataset"));
	this->labels = (CvMat*) cvRead(file, cvGetFileNodeByName(file,0, "labels"));
	cvReleaseFileStorage(&file);
}
void MultiTrain::trainModel(string outputdir)
{
	cout<<"Training the SVM classifier....."<<endl;
	SVM.train(this->trainData, this->labels, 0,0,this->params);
	SVM.save((outputdir+"/emo_svm_model.xml").c_str());
	cout<<"SVM model saved to file: "<<"emo_svm_model.xml"<<endl;
}


void MultiTrain::loadModel(string filename)
{
	this->SVM.load(filename.c_str());
}
void MultiTrain::calculateTrainDataCount(int tab[])
{
	for(int i=0; i<this->labels->rows; i++)
	{
		tab[CV_MAT_ELEM(*this->labels, int, i,0)]++;
	}
}
void MultiTrain::prepareSets(int class_counts[], int counts[], int part, int parts, CvMat *traindata, CvMat *trainlabels,
CvMat *testdata, CvMat *testlabels)
{
	int class_integral[7] = {0};
	for(int i=1; i<7; i++)
	{
		class_integral[i] = calcSum(class_counts,0,i-1);
	}

	int test_iter, train_iter;
	test_iter = train_iter = 0;
	int type = -1;
	for(int i=0; i<this->trainData->rows; i++)
	{
		if(i < (class_integral[0]+class_counts[0]))
			type = 0;
		else if(i < (class_integral[1]+class_counts[1]))
			type = 1;
		else if(i < (class_integral[2]+class_counts[2]))
			type = 2;
		else if(i < (class_integral[3]+class_counts[3]))
			type = 3;
		else if(i < (class_integral[4]+class_counts[4]))
			type = 4;
		else if(i < (class_integral[5]+class_counts[5]))
			type = 5;
		else if(i < (class_integral[6]+class_counts[6]))
			type = 6;

		if(type>=0)
		{
```

```
                        CvMat *r = cvCreateMat(1, 36*256, CV_32FC1);
                        cvGetRow(this->trainData, r, i);

                        if((i >=(part*counts[type]+class_integral[type])) && (i<((part+1)*counts[type]
+class_integral[type])))
                        {
                                for(int j= 0; j<r->cols; j++)
                                {
                                        CV_MAT_ELEM(*testdata, float, test_iter, j) = r->data.fl[j];
                                        CV_MAT_ELEM(*testlabels, int, test_iter, 0) = type;
                                }
                                test_iter++;
                        }
                        else
                        {
                                for(int j= 0; j<r->cols; j++)
                                {
                                        CV_MAT_ELEM(*traindata, float, train_iter, j) = r->data.fl[j];
                                        CV_MAT_ELEM(*trainlabels, int, train_iter, 0) = type;
                                }
                                train_iter++;
                        }
                }
        }
}

void MultiTrain::performCrossValidation(int parts)
{
        vector<double> test_results;
        int class_counts[7] = {0};
        int counts [7] = {0};

        if( parts !=0)
        {
                calculateTrainDataCount(class_counts);
                for(int i=0; i<7; i++)
                {
                        counts[i] = (class_counts[i]/parts);
                }
        }
        if(parts == 0 || parts == 1) cout<<"Cross validation cannot be performed for such input values"<<endl;
        else if(calcSum(counts, 7)<7) cout<<"The database is too small for performing the "<<parts<<"-fold cross
validation."<<endl;
        else
        {
                //MAIN LOOP
                for(int p=0; p<parts; p++)
                {
                        //CREATE SETS
                        CvMat *traindata = 0;
                        CvMat *trainlabels = 0;
                        CvMat *testdata = 0;
                        CvMat *testlabels = 0;

                        testdata = cvCreateMat(calcSum(counts, 7), 36*256, CV_32FC1);
                        testlabels = cvCreateMat(calcSum(counts, 7),1, CV_32SC1);

                        traindata = cvCreateMat(this->trainData->rows - calcSum(counts, 7), 36*256, CV_32FC1);
                        trainlabels = cvCreateMat(this->trainData->rows - calcSum(counts,7), 1, CV_32SC1);
```

```
                        //PREPARE SETS
                        this->prepareSets(class_counts, counts, p, parts, traindata, trainlabels, testdata, testlabels);

                        //PERFORM TRAINING
                        cout<<"Training the SVM classifier...part#"<<p<<endl;
                        SVM.train(traindata, trainlabels, 0,0, this->params);

                        //PERFORM TESTING
                        int TP = 0;
                        for(int i=0; i<(int)traindata->rows; i++)
                        {
                                CvMat *row = cvCreateMat(1, 36*256, CV_32FC1);
                                cvGetRow(traindata, row, i);
                                int res = (int)this->SVM.predict(row);
                                if(res == trainlabels->data.i[i])
                                        TP++;
                        }
                        double accuracy = (double)TP/(double)traindata->rows;
                        cout<<"accuracy for part#"<<p<<" : "<<accuracy<<endl;
                        //RELEASE SETS
                        cvReleaseMat(&traindata);
                        cvReleaseMat(&trainlabels);
                        cvReleaseMat(&testdata);
                        cvReleaseMat(&testlabels);
                        this->SVM.clear();
                        cout<<"----------------------------------------"<<endl;
                }
        }
}
int MultiTrain::getPrediction(IplImage *image)
{
        int res = 0;
        FaceDetection fd(image);

        if( !fd.detectElements()) return -3;

        fd.setRegions();
        Face f = fd.getFace();
        cvSetImageROI(image, f.upperface);
        IplImage *up = cvCreateImage(cvGetSize(image), image->depth, image->nChannels);
        cvCopy(image, up, 0);
        cvResetImageROI(image);
        cvSetImageROI(image, f.lowerface);
        IplImage *lo = cvCreateImage(cvGetSize(image), image->depth, image->nChannels);
        cvCopy(image, lo, 0);
        cvResetImageROI(image);
        FeatureExtraction fe(up, lo);
        fe.calculateLBP();

        CvMat *mat = cvCreateMat(1, 36*256, CV_32FC1);
        //cvInitMatHeader(mat, 1, 36*256, CV_32FC1, fe.feature_vector);
        for(int i=0; i<mat->cols; i++)
                mat->data.fl[i] = fe.feature_vector[i];


        res = (int) this->SVM.predict(mat);
        cvReleaseMat(&mat);
        return res;
}
void MultiTrain::testModel(string filename)
```

```
{
	this->loadDataSet(filename);

	int TP = 0; //true prediction counter

	for(int i=0; i<(int)this->trainData->rows; i++)
	{
		CvMat *row = cvCreateMat(1, 36*256, CV_32FC1);
		cvGetRow(this->trainData, row, i);
		int res = (int)this->SVM.predict(row);
		if(res == this->labels->data.i[i])
			TP++;
	}
	cout<<TP<<endl;
	this->ACC = (double)TP/(double)this->trainData->rows;
	cout<<"accuracy:"<<this->ACC<<endl;
}
void MultiTrain::createConfusionMatrix(string filename)
{
	int confusionMatrix[7][7]= {0};
	int overall[7] = {0};
	int TP = 0;


	this->loadDataSet(filename);

	for(int i=0; i<(int)this->trainData->rows; i++)
	{
		CvMat *row = cvCreateMat(1, 36*256, CV_32FC1);
		cvGetRow(this->trainData, row, i);
		int res = (int)this->SVM.predict(row);
		confusionMatrix[this->labels->data.i[i]][res]++;
		if(res == this->labels->data.i[i])
			TP++;
	}

	cout<<"-------------------------------------"<<endl;
	cout<<"----------confusion matrix-----------"<<endl;
	for(int i=0; i<7; i++)
	{
		overall[i] = calcSum(confusionMatrix[i], 7);
		for(int j=0; j<7; j++)
		{
			double perc = (double)confusionMatrix[i][j]*100/(double)overall[i];
			cout<<perc<<"\t";
		}
		cout<<endl;
	}
	cout<<"-------------------------------------"<<endl;
	this->ACC = (double)TP/(double)this->trainData->rows;
	cout<<"accuracy:"<<this->ACC<<endl;
}
void MultiTrain::processData(string path, int i)
{
	//(inputdir+"/"+filenames[i]).c_str()
	IplImage *image = cvLoadImage(path.c_str(), 1);
	FaceDetection detector(image);
	if( detector.detectElements())
	{
		detector.setRegions();
```

```
                Face f = detector.getFace();

                cvSetImageROI(image, f.upperface);
                IplImage *up = cvCreateImage(cvGetSize(image), image->depth, image->nChannels);
                cvCopy(image, up, 0);
                cvResetImageROI(image);
                cvSetImageROI(image, f.lowerface);
                IplImage *lo = cvCreateImage(cvGetSize(image), image->depth, image->nChannels);
                cvCopy(image, lo, 0);
                cvResetImageROI(image);

                FeatureExtraction fe(up, lo);
                fe.calculateLBP();

                for(int j=0; j<36*256; j++)
                {
                        this->trainData->data.fl[i*36*256+j] = fe.feature_vector[j];
                }
                cvReleaseImage(&up);
                cvReleaseImage(&lo);
        }
        cvReleaseImage(&image);
}
/**cvSeq comparison function**/
static int comp_func(const void* _a, const void* _b, void* userdata)
{
        CvRect * a = (CvRect*)_a;
        CvRect * b = (CvRect*)_b;
        int areaA = a->width * a->height;
        int areaB = b->width * b->height;
        if(areaA < areaB) return 1;
        else return -1;
}

static int comp_func_x(const void* _a, const void* _b, void* userdata)
{
        CvRect * a = (CvRect*)_a;
        CvRect * b = (CvRect*)_b;
        if(a->x > b->x) return 1;
        else return -1;
}
```

lbp.hpp

```
/**
* author: Ewa Piatkowska
* class for LBP encoding
*/
#include "modules.hpp"
#include "cxcore.h"


class LBP
{
public:
        LBP();
public:
        ~LBP();
```

```
        IplImage* image;
        IplImage* LBPimage;
        CvHistogram* hist;

public:
        /* calculate LBP features */
        void createLBP(IplImage *patch);
        /* create histogram of LBP features */
        void histogram();
        /* copy histogram to feature set */
        void LBP::fillFeatureSet(float *set, int start_indx);
};
```

lbp.cpp

```
/**
* author: Ewa Piatkowska
*/
#include "lbp.hpp"

LBP::LBP()
{
        image =0;
        LBPimage =0;
        hist =0;
}
LBP::~LBP()
{
        if(image)       cvReleaseImage(&image);
        if(LBPimage) cvReleaseImage(&LBPimage);
}
void LBP::createLBP(IplImage* patch)
{
        IplImage* temp_image = cvCreateImage(cvGetSize(patch), patch->depth, patch->nChannels);
        cvCopy(patch, temp_image);
        image = cvCreateImage(cvSize(temp_image->width, temp_image->height), 8, 1);


        if (temp_image->nChannels == 3)
        {
                cvCvtColor(temp_image, image, CV_BGR2GRAY);
        }
        LBPimage = cvCreateImage(cvSize(image->width, image->height), 8, 1);

        int center=0;
        int center_lbp=0;
        for (int row=1; row<image->height-1; row++)
        {
                for (int col=1; col<image->width-1; col++)
                {
                        center = cvGetReal2D(image, row, col);
                        center_lbp = 0;
                        if (center >= cvGetReal2D(image, row-1, col-1))
                        {
                                center_lbp += 1;
                        }
                        if (center >= cvGetReal2D(image, row-1, col))
```

```
                        {
                                center_lbp += 2;
                        }
                        if (center >= cvGetReal2D(image, row-1, col+1))
                        {
                                center_lbp += 4;
                        }
                        if (center >= cvGetReal2D(image, row, col-1))
                        {
                                center_lbp += 8;
                        }
                        if (center >= cvGetReal2D(image, row, col+1))
                        {
                                center_lbp += 16;
                        }
                        if (center >= cvGetReal2D(image, row+1, col-1))
                        {
                                center_lbp += 32;
                        }
                        if (center >= cvGetReal2D(image, row+1, col))
                        {
                                center_lbp += 64;
                        }
                        if (center >= cvGetReal2D(image, row+1, col+1))
                        {
                                center_lbp += 128;
                        }
                        cvSetReal2D(LBPimage, row, col, center_lbp);
                }
        }
        cvReleaseImage(&temp_image);
}



void LBP::histogram()
{
        int bins = 256;
        int hsize[] = {bins};
        float range[] = {0,256};
        float * ranges[] = {range};
        float min_value =0, max_value = 0;
        IplImage * planes[] = {this->LBPimage};

        this->hist = cvCreateHist(1, hsize, CV_HIST_ARRAY, ranges, 1);
        cvCalcHist(planes, this->hist, 0,0);
}
void LBP::fillFeatureSet(float *set, int start_indx)
{
        for(int i=0; i<256; i++)
        {
                set[i+start_indx] = cvQueryHistValue_1D(hist, i);
        }
}
```

helpers.hpp

```
/**
* author: Ewa Piatkowska
* set of different functions
*/

#ifndef helpers_hpp
#define helpers_hpp

#include <vector>
#include <string>
#include <sstream>
#include <fstream>
#include <math.h>
#include <iostream>
using namespace std;

/* function for listing files that match the pattern from directory*/
void listFiles(string directory, string pattern, vector<string> &files);
/* function for concatenating strings with integers */
string createSname(string path, string fname, string f, int indx, string ext);
/* function for concatenating strings - creating temporary file names */
string createFname(string path, string fname, string ext);
/* function for translating emotion codes */
string showResult(int code);
/* function for suming the array values*/
int calcSum(int *tab, int n);
/* function for suming the array values from indx1 to indx2*/
int calcSum(int *tab, int idx1, int idx2);

#endif
```

helpers.cpp

```
/**
* author: Ewa Piatkowska
*/
#include "helpers.hpp"
#include "modules.hpp"

void listFiles (string directory, string pattern, vector<string> &files)
{
	string command = "dir "+directory+"\\"+pattern+" /B > temp.txt";
	string d;
	system(command.c_str());
	ifstream in;
	in.open("temp.txt", ifstream::in);

	if( in.is_open())
	{
		while(true)
		{
			if(!(in>>d)) break;
			files.push_back(d);
		}
		in.close();
	}
```

```
}
string createSname(string path, string fname, string f, int indx, string ext)
{
	//path +filename+ frame+ idnx + ext
	string t;
	stringstream s;
	s<< indx;
	s>> t;
	string result = path+fname+f+t+ext;
	return result;
}
string createFname(string path, string fname, string ext)
{
	string result= path+fname+ext;
	return result;
}
string showResult(int code)
{
	string result="";

	switch(code)
	{
	case 0:
		result = "neutral";
		break;
	case 1:
		result = "happy";
		break;
	case 2:
		result = "sad";
		break;
	case 3:
		result = "surprised";
		break;
	case 4:
		result = "angry";
		break;
	case 5:
		result = "fear";
		break;
	case 6:
		result = "disgusted";
		break;
	}
	return result;
}
int calcSum(int *tab, int n)
{
	int sum =0;
	for(int i =0; i<n; i++)
		sum+= tab[i];

	return sum;
}
int calcSum(int *tab, int idx1, int idx2)
{
	int sum=0;
	for(int i=idx1; i<=idx2; i++)
		sum+= tab[i];
```

```
	return sum;
}
```

tasks.hpp

```
/**
* author Ewa Piatkowska
* functions for performing particular tasks
*/

#ifndef tasks_hpp
#define tasks_hpp

#include "modules.hpp"

/** show image **/
void show(IplImage *im);
/** sample emotion recognition **/
void sample(string filename);
/** perform face detection **/
void processDetection(string inputdir, string outputdir, string pattern);
/** capture snaphots (frames) from a video file **/
void captureSnapshots(string inputdir, string outputdir, int rate, string pattern);


/*****DEMO****/
class Demo
{
public:
	Demo(string filename, string type);
	~Demo();

	vector<int> predictions;
	string videofile;
	string type;

	void processVideo();
	void displayPredictions(bool save);
	void getStatistics();
};
/************/
#endif
```

tasks.cpp

```
/**
* author: Ewa Piatkowska
*/

#include "tasks.hpp"

/********************DEMO**********************************/
Demo::Demo(string filename, string type)
{
	this->videofile = filename;
	this->predictions = vector<int>();
	this->type = type;
}
```

```
Demo::~Demo(){}

void Demo::processVideo()
{
	CvCapture * capture = cvCaptureFromAVI(this->videofile.c_str());
	cout<<"Loading classifier...."<<endl;
	MultiTrain mt;
	string path = "..\\datasets\\"+type+"\\";
	mt.loadModel((path+"emo_svm_model.xml"));
	int fps = ( int )cvGetCaptureProperty( capture, CV_CAP_PROP_FPS );

	if( !capture)
	{
		cout<<"problems with avi file"<<endl;
	}
	else
	{
		cout<<"Processing video...."<<endl;
		IplImage *frame=0;
		int k = 0;
		while(1)
		{
			frame = cvQueryFrame(capture);
			if(!frame)  break;

			if( k%10 == 0)
			{
				int res = (int)mt.getPrediction(frame);
				cout<<"Prediction for frame #"<<k<<" => "<<res<<endl;
				this->predictions.push_back(res);
			}
			k++;
		}
		cvReleaseImage(&frame);
	}
	cvReleaseCapture(&capture);
}
void Demo::displayPredictions(bool save)
{
	CvCapture * capture = cvCaptureFromAVI(this->videofile.c_str());
	CvVideoWriter *writer = 0;
	int isColor = 1;
	int fps = (int) cvGetCaptureProperty(capture, CV_CAP_PROP_FPS);
	int frameWidth = (int) cvGetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH);
	int frameHeight = (int) cvGetCaptureProperty(capture, CV_CAP_PROP_FRAME_HEIGHT);

	if(save)
	{
		writer = cvCreateVideoWriter("demo.avi",-1, fps, cvSize(frameWidth, frameHeight), isColor);
	}

	cvNamedWindow( "preview", 1 );
	CvFont font;
	cvInitFont(&font,0, 1.1f, 1.1f,0,2,8);
	string result ="";
	if( !capture)
	{
		cout<<"problems with avi file"<<endl;
	}
	else
```

```
	{
		IplImage *frame=0;
		int k = 0, l=0;
		while(1)
		{
			frame = cvQueryFrame(capture);
			if(!frame) break;
			if( k%10 == 0)
			{
				result = showResult(this->predictions[l++]);
				cout<<"Prediction for frame #"<<k<<" => "<<result<<endl;
			}
			cvPutText(frame, result.c_str(), cvPoint(50,50), &font, cvScalar(255,0,0));
			cvShowImage("preview",frame);
			if(save && writer)
			{
				cvWriteFrame(writer, frame);
			}
			cvWaitKey(1000/fps);
			k++;
		}
		cvReleaseImage(&frame);
	}
	cvReleaseCapture(&capture);
	cvDestroyWindow("preview");
	if(writer) cvReleaseVideoWriter(&writer);
}
void Demo::getStatistics()
{
	int n, h, sd, su, a, f,d;
	n=h=sd=su=a=f=d=0;
	for(int i=0; i< (int)this->predictions.size(); i++)
	{
		switch(this->predictions[i])
		{
		case 0:
			n++; break;
		case 1:
			h++; break;
		case 2:
			sd++; break;
		case 3:
			su++; break;

		case 4:
			a++; break;
		case 5:
			f++; break;
		case 6:
			d++; break;
		}
	}
	cout<<"---------------------------------"<<endl;
	cout<<"Emotions recognized:"<<endl;
	cout<<"Neutral: "	<< (double)n/(double)this->predictions.size()	<<endl;
	cout<<"Happy: "		<< (double)h/(double)this->predictions.size()	<<endl;
	cout<<"Sad: "		<< (double)sd/(double)this->predictions.size()	<<endl;
	cout<<"Suprised: "	<< (double)su/(double)this->predictions.size()	<<endl;
	cout<<"Angry: "		<< (double)a/(double)this->predictions.size()	<<endl;
	cout<<"Fear: "		<< (double)f/(double)this->predictions.size()	<<endl;
```

```
	cout<<"Disgusted: "	<< (double)d/(double)this->predictions.size()	<<endl;
	cout<<"---------------------------------"<<endl;
}
/*****************************************************/
void show(IplImage *im)
{
	cvNamedWindow("preview");
	cvShowImage("preview", im);
	cvWaitKey(0);
	cvDestroyWindow("preview");
}
/*****************************************************/
void sample(string filename, string type)
{
	CvFont font;
	cvInitFont(&font,0, 1.0f, 1.0f,0,2,8);

	IplImage *img = cvLoadImage(filename.c_str(), 1);
	MultiTrain mt;
	cout<<"Loading the classifier...."<<endl;
	mt.loadModel("..\\datasets\\"+type+"\\emo_svm_model.xml");
	int res = mt.getPrediction(img);
	string result = showResult(res);
	cvPutText(img, result.c_str(), cvPoint(20,20), &font, cvScalar(255,255,0));
	cout<<result<<endl;
	show(img);
	cvReleaseImage(&img);
}
/*****************************************************/
void processDetection(string inputdir, string outputdir, string pattern="*.jpg")
{
	vector<string> filenames = vector<string>();
	listFiles(inputdir, pattern, filenames);
	try
	{
		IplImage *image=0;
		for(int i=0; i<(int)filenames.size(); i++)
		{
			image = cvLoadImage((inputdir+"/"+filenames[i]).c_str(), 1);
			FaceDetection detector(image);
			if( ! detector.detectElements()) continue;

			detector.detectElements();
			detector.setRegions();

			Face f = detector.getFace();

			cvRectangle(image, cvPoint(f.upperface.x, f.upperface.y), cvPoint(f.upperface.x+
f.upperface.width, f.upperface.y+f.upperface.height), cvScalar(0,255,255));
			cvRectangle(image, cvPoint(f.lowerface.x, f.lowerface.y),
cvPoint(f.lowerface.x+f.lowerface.width, f.lowerface.y+f.lowerface.height), cvScalar(0,0,255));

			cvSaveImage((outputdir+"/"+filenames[i]).c_str(), image);
			cout<<"image #"<<i<<(outputdir+"/"+filenames[i]).c_str()<<" is being saved..."<<endl;
		}
		cvReleaseImage(&image);
	}
	catch(cv::Exception &e)
	{
		cout<<e.what()<<endl;
```

```
        }
}
/*********************************************************************/
void captureSnapshots(string inputdir, string outputdir, int rate, string pattern="*")
{
        CvCapture *capture = 0;
        cvNamedWindow( "preview", 1);
        vector<string> filenames = vector<string>();
        listFiles(inputdir, pattern, filenames);

        for(int i=0; i< (int) filenames.size(); i++)
        {
                capture = cvCaptureFromAVI((inputdir+"/"+filenames[i]).c_str());
                int fps = ( int )cvGetCaptureProperty( capture, CV_CAP_PROP_FPS );
                if( !capture)
                {
                        cout<<"problems with avi file"<<endl;
                        continue;
                }
                else
                {
                        IplImage *frame=0;
                        int k=1;
                        while(1)
                        {
                                frame = cvQueryFrame(capture);
                                if(!frame)  break;

                                if(k%rate == 1)
                                {
                                        //if(k>30)
                                        {
                                                string fname = filenames[i].substr(0, (filenames[i].length()-4));
                                                fname = createSname(outputdir+"/", fname, "_frame", k, ".jpg");
                                                cvSaveImage(fname.c_str(), frame);
                                                cout<<"file: "<<fname<<" is saving..."<<endl;
                                        }
                                }
                                k++;
                                cvShowImage("preview",frame);
                                cvWaitKey(1000/fps);
                        }
                        cvReleaseImage(&frame);
                }
        }
        cvReleaseCapture(&capture);
        cvDestroyWindow("preview");
}
```

main.cpp

```
#include "modules.hpp"
#include "lbp.hpp"
#include "tasks.hpp"

void playDemo(string type)
{
	//string videofile = "../sample_videos/disgs_0004_3.mpg";
	string videofile = "../sample_videos/happy_0014_1.mpg";
	//string videofile = "../sample_videos/sadns_0005_1.mpg";
	Demo demo(videofile, type);
	demo.processVideo();
	demo.displayPredictions(false);
	demo.getStatistics();
}

int main()
{
	try
	{
		playDemo("FEED");
	}
	catch( Exception &e)
	{
		cout<<e.what()<<endl;
	}
	system("pause");
	return 0;
}
```