UNIVERSITATEA POLITEHNICA BUCUREŞTI
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT DE DIPLOMĂ

Expressio

Coordonator ştiinţific:
Conf.dr.ing. Rughiniş Răzvan
Ş.L. dr. ing. George Milescu
Ş.L. dr. ing. Vlad Posea

Absolvent:
Mihail Dunaev

BUCUREŞTI

2014

POLITECHNICA UNIVERSITY OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE DEPARTMENT

DIPLOMA PROJECT

Expressio

Thesis supervisor:
Conf.dr.ing. Rughiniş Răzvan
Ş.L. dr. ing. George Milescu
Ş.L. dr. ing. Vlad Posea

Thesis author:
Mihail Dunaev

BUCHAREST

2014

Table of Contents

Automatic emotion recognition can considerably improve the human-computer interaction (HCI). Hence it was an active research topic of great interest since 90s up to this date. Emotion recognition algorithms are based on voice and face analysis. I designed a system that can identify the user's main emotion from the six basic ones, using the video stream from a camera. I used the Intel Perceptual Computing SDK to capture the video frames as well as face detection and machine learning algorithms from the OpenCV library. Emotions are classified using the Fisherface model[1].

Recunoaşterea automată a emoţiilor poate îmbunătăţii considerabil interacţiunea om-calculator. Din această cauză, a fost un subiect activ şi de mare interes din anii '90 până în prezent. Modalităţile de identificare a emoţiilor se bazează în principal pe analiza vocii şi a feţei dintr-o captură 2D (în format RGB). Am creat astfel un sistem care, folosind o cameră video, identifică faţa utilizatorului şi determină emoţia predominantă a acestuia, din cele şase emoţii de bază. Am folosit SDK-ul Intel Perceptual Computing pentru a captura imagini de la cameră precum şi algoritmi de detecţie a feţei şi învăţare automată din cadrul OpenCV. Pentru clasificare am folosit modelul Fisherface[1].

1. Introduction

We can say that the beginning of facial expression analysis started back in 1872 when Charles Darwin published his book "*The Expression of the Emotions in Man and Animals*"[2]. He grouped various kinds of facial expression into similar categories, like low spirits, high spirits etc. This was followed by a long period of lack of research interest in the field until 1970 when psychologist Paul Ekman and his colleagues bring major contributions to facial expression analysis and emotion recognition. They talk about six so-called "basic" or "prototypic" emotions : happiness, sadness, anger, surprise, disgust and fear, as well as modalities to identify them. Their system is called FACS : Facial Action Coding System, and their work greatly influenced the development of modern day emotion recognition algorithms. Later on, more emotion classifications were made by Robert Plutchik in 1980 who identified 8 primary bipolar emotions and 8 advanced emotions (also called the "*Plutchik's wheel of emotions*") and in 2000 Parrott came up with 136 emotional states (primary, secondary and tertiary). However, since 1990, computer scientists have tried to create automatic systems that detect only the original six emotions described by Ekman.

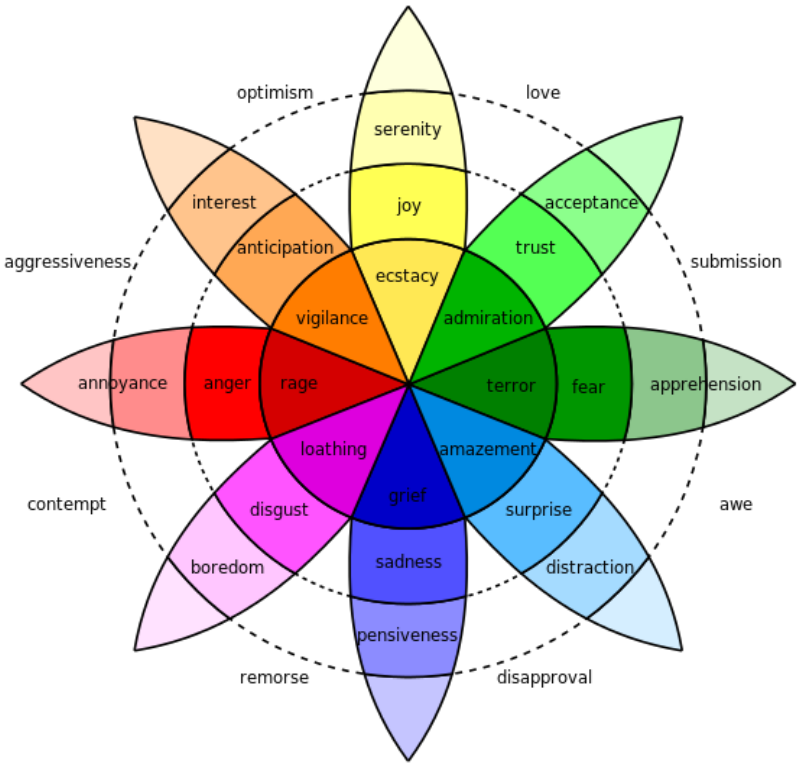


Fig. 1. Plutchik's wheel of emotions

The study of automatic emotion recognition is called Affective Computing. Even though the first attempt at an automatic system that detects the human emotions dates back to 1987[3], Affective Computing really took off in the mid 1990s. In fact, the term first appeared in a paper by Rosalind Picard in 1995[4]. She said that “computers that will interact naturally and intelligently with humans need the ability to at least recognize and express affect”. Few years later she co-founded *Affectiva,* a company that offers software solutions for recognizing human emotions based on facial cues. The company helps brands improve their advertising and marketing messages and is now quite successful. Current emotion recognition algorithms follow a specific pattern : they use machine learning to process different modalities like facial expression detection, speech recognition, natural language processing and outputs a label (i.e. ‘happy’). If we apply such an algorithm on some image containing a face in it, we need certain parameters to be used in the training process. The most widely used parameters are the Action Units (AU) described in FACS. I will address FACS later on.

1.1 Motivation

Affective Computing can substantially improve the human-computer interaction experience. Up until now machines could not identify user emotion and give them feedback taking this into account, resulting in a large drawback. Imagine that you just arrived home from work and you are stressed but you have a system installed that is capable of identifying what you feel and change the environment accordingly. For example it can change the lights from your room (to a warmer green color) or even the music to something relaxing.

Another example of this sort is AutoEmotive[5]. They try to build an emotion sensitive car, that is capable of changing the interior lights and music when you are stressed or when you are tired. It can also change outside lights intensity or even notify other drivers about your emotional state. This can potentially save human lives.

There are many other examples where Affective Computing can be useful. Learning Companion is a system that can track your alertness level while you attend a course and send feedback to the teacher or even change the environment so that you can become more interested. Affectiva, the company Rosalind Picard co-founded, used their product to help brands improve their advertising and marketing messages, as well as people on the autism spectrum.

2. Emotion Detection from Facial Expressions

2.1 Introduction

Emotion detection can be done by analyzing multiple types of data : images or video streams (facial expression detection), audio streams (acoustic properties of speech) and even text (using natural language processing). In my project I used the first method to build a real-time application that can predict user's emotions from one of the six basic ones.

Emotion detection from facial expression might seem easy for a human to do. For example, anyone can tell that this person is happy :

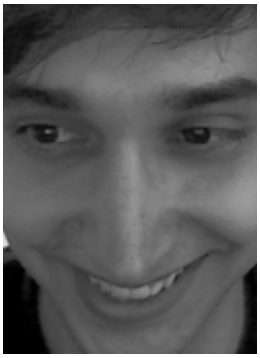


Fig. 2. A happy face

For a machine however, a portion of the above image would look like this :

254	143	203	176	109	229	177	220	192	9	229	142	138	64	0	63	28	8	88	82
27	68	231	75	141	107	149	210	13	239	141	35	68	242	110	208	244	0	33	88
54	42	17	215	230	254	47	41	98	180	55	253	235	47	122	208	76	110	152	100
9	186	192	71	104	193	88	171	37	233	18	147	174	1	143	211	176	188	192	68
179	20	238	192	190	132	41	248	22	134	83	133	110	254	176	238	188	234	51	204
232	25	0	183	174	129	81	30	110	189	0	173	197	183	153	43	22	87	68	118
235	35	151	185	129	81	239	170	195	94	38	21	67	101	58	37	196	149	52	154
155	242	54	0	104	109	189	47	130	254	225	156	31	181	121	15	126	35	252	205
223	114	79	129	147	0	201	68	89	107	58	44	253	84	38	1	62	5	231	218
55	188	237	188	80	101	131	241	68	133	124	151	111	28	190	4	240	78	117	145
152	155	229	78	90	217	219	105	116	77	38	49	2	9	214	181	205	116	135	33
182	94	176	199	20	149	57	223	232	113	32	45	177	15	31	179	100	119	208	81
224	118	124	172	75	29	69	180	187	195	41	44	8	170	158	101	131	31	28	112
238	83	38	7	83	69	173	183	98	237	67	227	18	218	248	237	75	192	201	146
88	195	224	207	140	22	31	118	234	34	182	116	23	47	68	242	189	152	116	248
140	37	101	230	246	145	122	64	27	58	229	1	225	143	91	100	98	90	40	195
251	4	178	139	121	95	97	174	249	182	77	115	223	186	182	82	65	252	83	196
179	180	223	230	87	182	148	78	176	19	17	4	184	176	183	102	83	81	132	206
173	137	185	242	181	181	214	49	74	238	197	37	98	102	15	217	148	8	102	188
85	9	17	222	18	210	70	21	78	241	184	216	93	93	208	102	153	212	119	47

Fig. 3. Grayscale image as seen by a machine

Color images are even more complicated since there are 3 values per pixel (red, green & blue for example). So we ask ourselves the question : how can machines detect emotions from an image like this, or how can it detect anything in fact ? The answer is machine learning. This is the standard way to do it, at least for now.

First off, what is machine learning ? Putting it simply, machine learning is that subfield of artificial intelligence that tries to make a machine (like a computer) learn on its own. A particular case of this is *supervised learning* where the machine learns from manually indexed events. Let us see an example : let's say we want to create a program that can predict whether a tumor is malignant or benign based on its size. First we give the program some examples to train on :

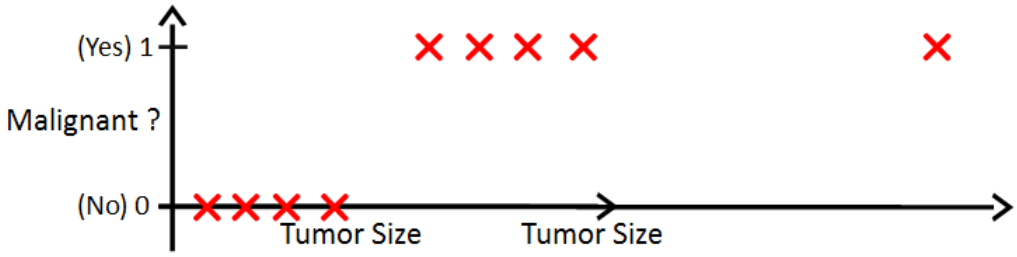


Fig. 4. Basic machine learning training example

The program will then try to generate a function that will look like this (one algorithm that can generate such a function is the gradient descent algorithm) :

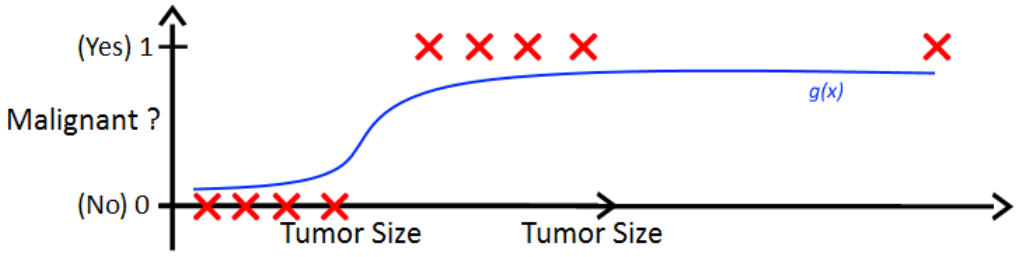


Fig. 5. Machine learning generated function

G(x) will take as input the size of the tumor (labeled as *x*) and will output a value between 0 and 1 (on the *y* axis) interpreted as the probability of a tumor being malignant or not (1 meaning the tumor is certainly *100%* malignant). Generating *g(x)* is the learning part. The more training data you provide to the algorithm, the more accurate it will be. But what happens if the input size has more than 1 dimension (let's say that in the tumor example we want to take into account also the age of the tumor) ? It's very simple, we can picture everything in 2D. We will see that *g(x)* this time is a 2D space (for example a line in the *x*-*y* plane) that separates the input data, like in Fig. 6. In general, for *n* input variables (x_1, x_2 ... x_n; also called features) *g(x)* will be an *n*-dimensional space*.

*in fact it's a bit more complicated than this; *g(x)* function is actually 3-dimensional, just like in our previous example with just one feature it was 2-D (it takes as input all features *x*, and outputs a *y* between 0 and 1; the idea behind that line is that *g(x)* should be 0.5 for every pair (x_1,x_2) that is on the line and it should go to 1 when it moves in the plane with positive training sets, and to 0 if it goes in the opposite plane

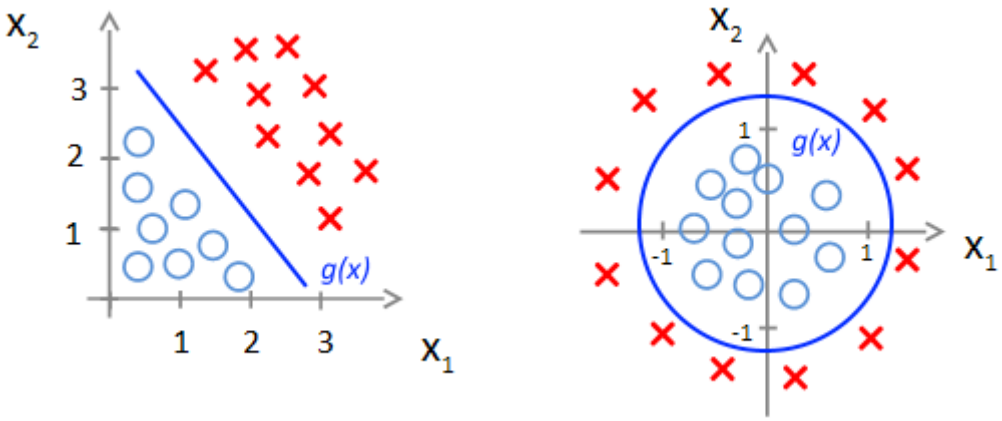


Fig. 6. Generated function when we have 2 features

Now that we know all this, how can we use machine learning for digital images ? Let's say we want to label images as happy *(y = 0)* or sad *(y = 1)*, what do we take as features ? The default approach is to take each pixel's value (or intensity) as a different feature. This means that for a 64x64 image we will have 4096 dimensions! This is really a lot and if the picture is also in RGB we will have 3 times more features.

Working with so many features is difficult and most machine learning algorithms will perform poorly (slow speed, bad classification precision etc) if applied directly on the unprocessed input data. That is why we try to decrease the number of features first. We could do this manually or use an algorithm that will do that for us in an automatic fashion. Most popular algorithms that do that are Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA). Both will reduce the dimensions from *n* to *k*, where $k < n$ while preserving important features from the training data. PCA will try to reduce the dimensions while preserving the features that give highest variation (in our case it will get rid of the pixels that are almost identical for all images in the training sets because they don't tell us anything) while LDA will try to keep the features that give the maximum variation between classes (it will get rid of the pixels that don't help us in making a difference between classes).

To best see how PCA and LDA work, let's look at this example : let's say we want to reduce the input space from 2D to 1D (for the sake of argument) in the next scenario (elements from one class are marked with red, while those from the other class with green) : input data is represented in Fig. 7. PCA will compute that the best 1D space to project the data on is the red line (Fig. 8), preserving the maximum variation among input data (doesn't take class into account), while LDA will compute the blue line as the best 1D subspace. You can see that LDA makes a really good distinction between classes. In practice LDA will perform better than PCA.

PCA consists of the following steps :

- mean normalization and feature scaling
- computing covariance matrix $C = 1/m * X * X^T$ where m is how many training sets we have and *X* is the input matrix (all x-s grouped together in a matrix)

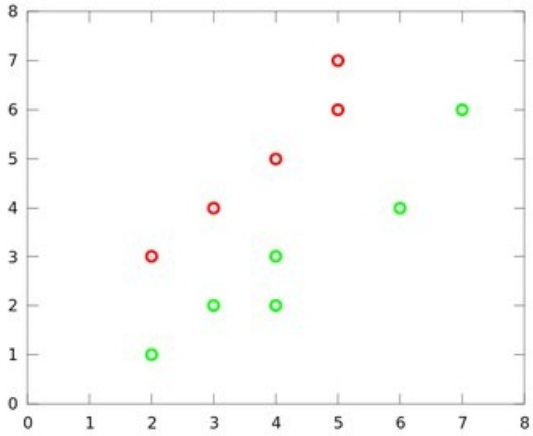


Fig. 7. Some input data set

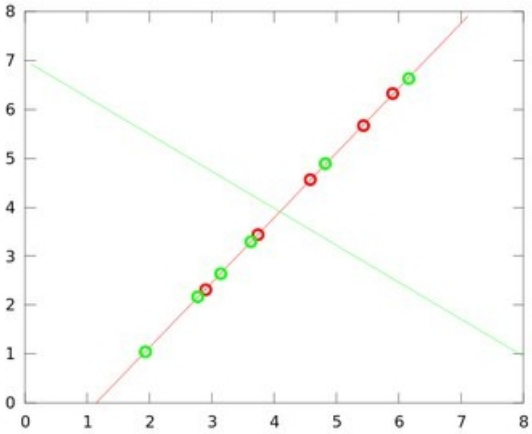


Fig. 8. Projecting with PCA

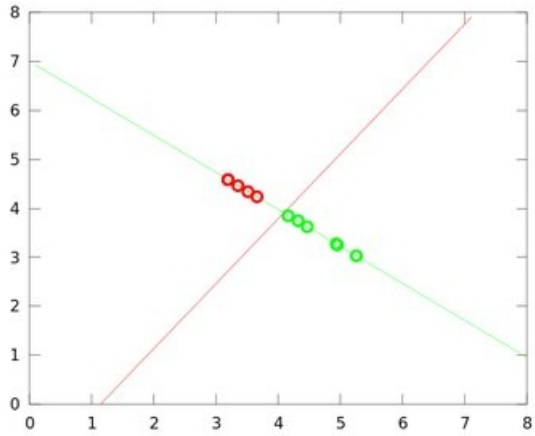


Fig. 9. Projecting with LDA

- singular value decomposition of matrix *C : [U,S,V] = svd(C)*
- iterating with k from 1 to n, and finding minimum value such that

$$\frac{\sum_{i=1}^{k} s_{ii}}{\sum_{i=1}^{n} s_{ii}} \geq 0.99$$

(this is basically how much variation of the original input we want to keep)

- projecting everything in the *k*-dimensional subspace $Z = U^T_k * X$

Finally we can use the new *Z* features (that are *k*-dimensional) to train our program, and project new *X* features before predicting the class. Also, mean normalization and feature scaling are not really part of PCA, but it is a good idea to do them before. You basically subtract the mean from every feature and then divide by the standard deviation so all features should be in same range.

This is just a brief introduction to PCA, if you want to find out more I recommend taking Andrew Ng's machine learning online course at coursera[x], for free. PCA was first introduced in 1901 by mathematician Karl Pearson[x] and first used in computer vision for face recognition by Matthew Turk and Alex Pentland in 1991, with a new name : Eigenface[x]. Eigenface is basically PCA followed by a nearest neighbor algorithm which gave good results at the time. The name of Eigenface comes from the fact that we use the eigenvectors from the *U* matrix (the row vectors) that we obtain after the singular value decomposition of matrix *C*, to compute the new input.

LDA is a bit more complicated than PCA. Basically we want to find the *k*-dimensional subspace that maximizes $|\mu_1' - \mu_2'|$, where μ_1', μ_2' are the means of the features from the 2 classes in the new space. If we want to reduce dimensionality from 2D to 1D then the algorithm will be :

- compute the two means μ_1 and μ_2 from the initial training sets
- compute the scatters s_1 and s_2, where $s_i = \sum(x - \mu_i)(x - \mu_i)^T$
- compute the within-class scatter $s_w = s_1 + s_2$
- the equation of the line we are looking for will be given by the vector $w = s_w^{-1}(\mu_1 - \mu_2)$

Now we can project the input on w and use our learning algorithm from there. LDA was first invented by statistician Ronald Fisher in 1963, and just like PCA it was introduced into computer science with a new name : Fisherface (after its creator), in 1997. Again, it is common practice to use k-Nearest Neighbor after LDA as the learning algorithm. Fisherface was proven to perform better at face recognition than Eigenface[x]. I wanted to mention Fisherface because it is also the algorithm I used for training and predicting emotions. I will address it later on in this paper.

After dimensionality reduction with PCA or LDA, we just need to take care of the learning process, generating *g(x)* function. There are many popular algorithms capable of doing this. Most used ones are logistic regression, neural networks and support vector machines (svm). Nearest neighbor is the easiest way of doing it, however it doesn't perform as good in some situations. In

fact, nearest neighbor doesn't even generate a *g(x)* function, it just checks the class of the closest neighbor (from the training data sets) and predicts that the new element (image) is from the same class. K-Nearest neighbor will look at k closest neighbors and will predict that the new element is from the same class with the majority of his neighbors.

Depending on what we want our program to predict, we choose the right input data. For example, for a simple face detection program, we will use two classes : images that contain just a face (labeled with *y = 1*) and images that don't contain a face (*y = 0*). Using the generated function g(x) we then can compute for new images the value which will tell us the probability that the new image is a face. If we want just to identify a face in a bigger picture, we can use a sliding window approach (start with a small frame, re-size it and move it around the image to look for a face). I will address *Viola-Jones* algorithm later on in this paper. Viola-Jones is a fast object detection algorithm that can be used in real-time applications. It is the current state of the art in face detection. Regarding emotion detection from facial expression, one very simple strategy would be to start with face images that express certain emotions. For each emotion identified, label each picture (let's say from *y = 1* to *y = 6* if we want to use the 6 emotions scheme), use LDA to reduce dimensions and train your program with the desired machine learning algorithm. You can then use it to predict emotions in new face pictures. However, modern emotion detection algorithms don't use the whole face in the training process, but only portions of it (the eyes or mouth sections, or training separately with the upper and lower regions of the face). I will talk about this in the next section.

2.2 State of the Art

The state-of-the-art algorithms in Affective Computing use the Facial Action Coding System or FACS. This system was developed by Paul Ekman and his colleagues in 1970. It uses facial parametrization to identify emotions. In other words, emotions influence what muscles are contracted on the face. If you identify what muscles groups are responsible for what emotion, you can easily tell what the user is feeling. An Action Unit is the abstraction of a contracting muscle. Ekman defined 46 such AUs. For example, when the inner portion of the brow is raised you say it’s AU1, when the corners of the lips are pulled down it’s AU15. Ekman divided the AUs into two categories : upper face (eyes, brows etc) and lower face (lips, mouth etc). You can see some examples of upper face AUs in fig. 1 and of lower face in fig. 2. If all the muscles are relaxed, it is said that the face is in neutral state. One last classification of AUs is that if they influence each other (muscles close together) they are non-additive (you can’t identify both on same face) and if they don’t, they are additive. With that in mind he mapped the six basic emotions to lists of AUs that uniquely idetify each, as we can see in table 1. All that is left for the programmer to do to correctly recognize emotion is to identify which AU is present on a face. Easy as it sounds, this is a difficult thing to do and is still an active research topic.

Other facial parametrizations worth mentioning : FAST, EMFACS, MAX, EMG, AFFEX, Mondic Phases, FACSAID and FAPs which is the MPEG-4 standard for emotion classification.

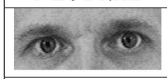
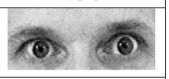
NEUTRAL	AU 1	AU 2	AU 4	AU 5
				
Eyes, brow, and cheek are relaxed.	Inner portion of the brows is raised.	Outer portion of the brows is raised.	Brows lowered and drawn together	Upper eyelids are raised.
AU 6	AU 7	AU 1+2	AU 1+4	AU 4+5
				
Cheeks are raised.	Lower eyelids are raised.	Inner and outer portions of the brows are raised.	Medial portion of the brows is raised and pulled together.	Brows lowered and drawn together and upper eyelids are raised.
AU 1+2+4	AU 1+2+5	AU 1+6	AU 6+7	AU 1+2+5+6+7
				
Brows are pulled together and upward.	Brows and upper eyelids are raised.	Inner portion of brows and cheeks are raised.	Lower eyelids cheeks are raised.	Brows, eyelids, and cheeks are raised.

Fig. 10. Upper face AUs

Table 1

Six basic emotions and the corresponding AUs

Emotion	Action Units
Happiness	AU6 + AU12
Sadness	AU1 + AU4 + AU15
Surprise	AU1 + AU2 + AU5B + AU26
Fear	AU1 + AU2 + AU4 + AU5 + AU7 + AU20 + AU26
Anger	AU4 + AU5 + AU7 + AU23
Disgust	AU9 + AU15 + AU16

NEUTRAL	AU 9	AU 10	AU 12	AU 20
				
Lips relaxed and closed.	The infraorbital triangle and center of the upper lip are pulled upwards. Nasal root wrinkling is present.	The infraorbital triangle is pushed upwards. Upper lip is raised. Causes angular bend in shape of upper lip. Nasal root wrinkle is absent.	Lip corners are pulled obliquely.	The lips and the lower portion of the nasolabial furrow are pulled pulled back laterally. The mouth is elongated.
AU15	AU 17	AU 25	AU 26	AU 27
				
The corners of the lips are pulled down.	The chin boss is pushed upwards.	Lips are relaxed and parted.	Lips are relaxed and parted; mandible is lowered.	Mouth stretched open and the mandible pulled downwards.

Fig. 11. Lower face AUs

So, how can we identify AUs from someone's facial expression ? Just like I explained in the previous section, the way to go is with a machine learning algorithm. We first have to train our program with labeled images with what AUs we want to predict. The number of classes is equal to the number of AUs per face region (upper or lower). We apply a dimensionality reduction algorithm, like LDA and then our desired learning algorithm. After the training process is done, we can start predicting new images : identifying the face, separating the upper and lower face, and predicting the present AUs. If there are more non-additive AUs visible, we should detect all of them. The emotion predicted will be the result of AUs detected on the face. One problem we might encounter is obtaining the AU database. Collecting face pictures and manually indexing them with AUs can be tedious work. However, there is one large database available for free (research purposes only) that has face images already indexed with AUs : Cohn-Kanade AU-Coded Expression Database[x]. Among results in the field, I would like to mention Y. Tian, T. Kanade, J.F. Cohn[x] who managed to identify 16 AUs with an accuracy of 93.2% using a neural networks learning algorithm (and their own database) and G. Donato[x] who succeeded at recognizing 8 AUs with an accuracy of 95.5%, using Independent Component Analysis instead of LDA, and Gabor wavelet representation[x].

3. Expressio

3.1 General

I have built a system that can identify the user's predominant emotion (from the 6 basic ones), analyzing frames received from a video camera. To achieve this I used several technologies and devices: Creative Senz3D Camera, Intel Perceptual Computing SDK, OpenCV, Windows API and Microsoft Visual Studio 2012. Once the emotion is identified, it is displayed on the screen in a graphical user interface (GUI). My program is called *Expressio* and it has 3 windows : a main window, a train window and the emorec window. The main window can be seen in Fig. 12.

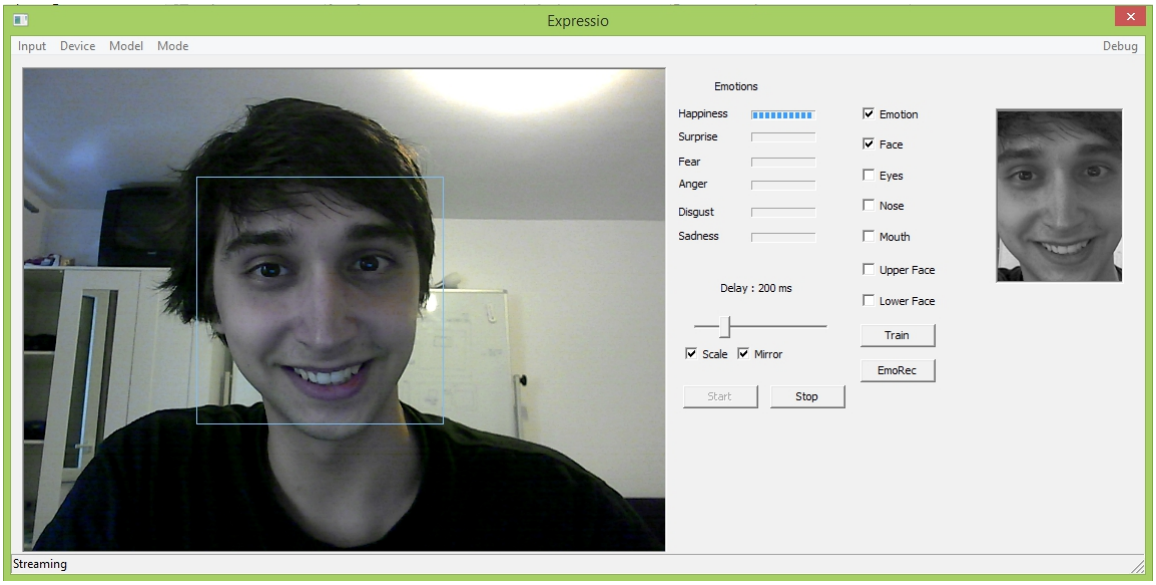


Fig. 12. Expressio Main Window

You can notice the big frame captured from the camera and the emotions section (the emotion predicted will have it's progress bar filled with blue up to *100%*). Expressio can either detect emotion with every frame, or it can wait for a minimum delay to pass (configurable with the *Delay*[*] slider), up to 1 second. The *Scale* checkbox will make the displayed frame from the camera change when you re-size the whole window while *Mirror* mirrors the image across the *y* axis.

As you can see it can also perform face detection, as well as eyes, nose and mouth detection and even gender detection. *Upper Face* and *Lower Face* are yet to be implemented. *Train* will open up the train window and *EmoRec* the emotion recording window. The small image on the right (the grayscale one) is the actual photo that will be used for emotion prediction. Expressio should work with any video camera available on the computer. You can change the camera (from installed ones) from the *Device*[**] menu and the libraries used for image capturing from *Input*. You can always stop the main loop by clicking on *Stop* and start it again later.

The program was previously trained with *Expressio Database*, but since the database is relatively small, I recommend you to use it like this : click on *Start* button and wait for the application to initialize. Click on *Train* afterwards and a window like the one in Fig. 13 should pop up. Select your gender and the emotion you want to simulate and click on *Take Snapshot*. I recommend you using at least 5 pictures for each emotion. After you are done taking pictures, click on *Done*. The window should freeze and the status should change to "*Training …"* . The training might take a while since it trains itself over all the pictures in the database (again). Clicking on *Cancel* will still save the pictures (they are saved as soon as you take them) but the algorithm won't start the training process now, so you can delete them manually from disk before the next training session.

Fig. 13. Train Window

[*] in the meantime this has changed to *Emotion Sample Rate* or *EMR*

[**] however you might experience crashes if you use *Intel PXC SDK* with a different camera than Creative Senz3D

After the training is done, the program will save the new training data to *models/emo.yml* (it overwrites it, so be careful!) and will take you back to the main window. From now on, emotions will be predicted based on the new training model just generated (with your pictures as well). Give it a try and simulate some emotions to see how it works.

The internal directories are structured in a simple fashion :

src/	- this is where the source files are located
models/	- this is where the models used for prediction are stored; I will talk more about this later
dbase/expressio/	- where the expressio database resides; you can add your own scaled pictures here
dbase/dynamic/	- this is where the images go after you "take snapshot" in "train" window; don't play with this directory, chances are that the stuff you put there will be replaced
bin/	- where the binary files are located

3.2 Architecture

In this section I will talk about my system's architecture. That is, a general overview of my program, how the training & predicting processes look like. I will go into more details in the *Implementation* section.

I had to train my program before anything. You can see how I did it (broadly) in Fig. 14. Basically the program will load every picture from the *Expressio Database* (127x175 PNG files each with a face corresponding to one of the six basic ones). These pictures are already labeled (those that represent happiness are located at *dbase/expressio/male/happiness/* or *.../female/happiness* etc) so I just had to apply the Fisherface training algorithm to generate the *models/emo.yml* file which will be loaded at start by the main program and used to predict emotions for new images.

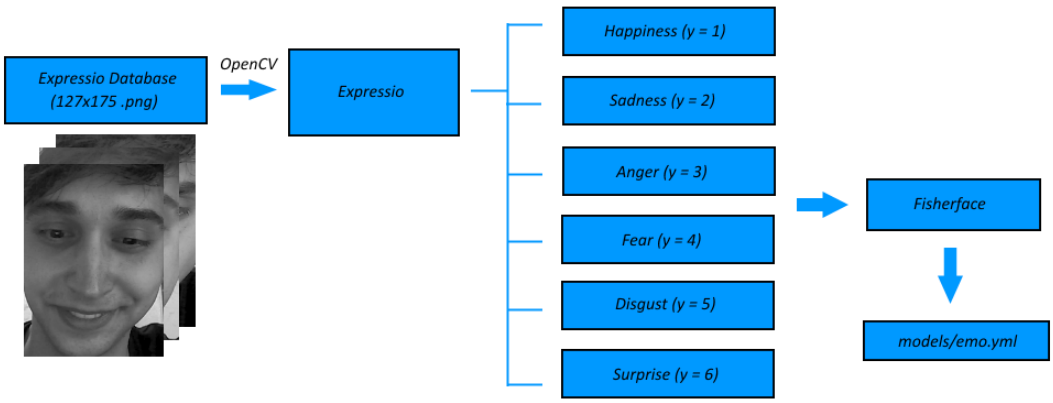


Fig. 14. Training Program Architecture

Once the main program is running, there are 2 threads that work simultaneously : one that deals with events for the currently displayed window and another one that takes frames from the camera buffer (using intel pxc sdk/opencv code), process it and displays the result on the screen in a window, using the Windows API. You can see that in Fig. 15.

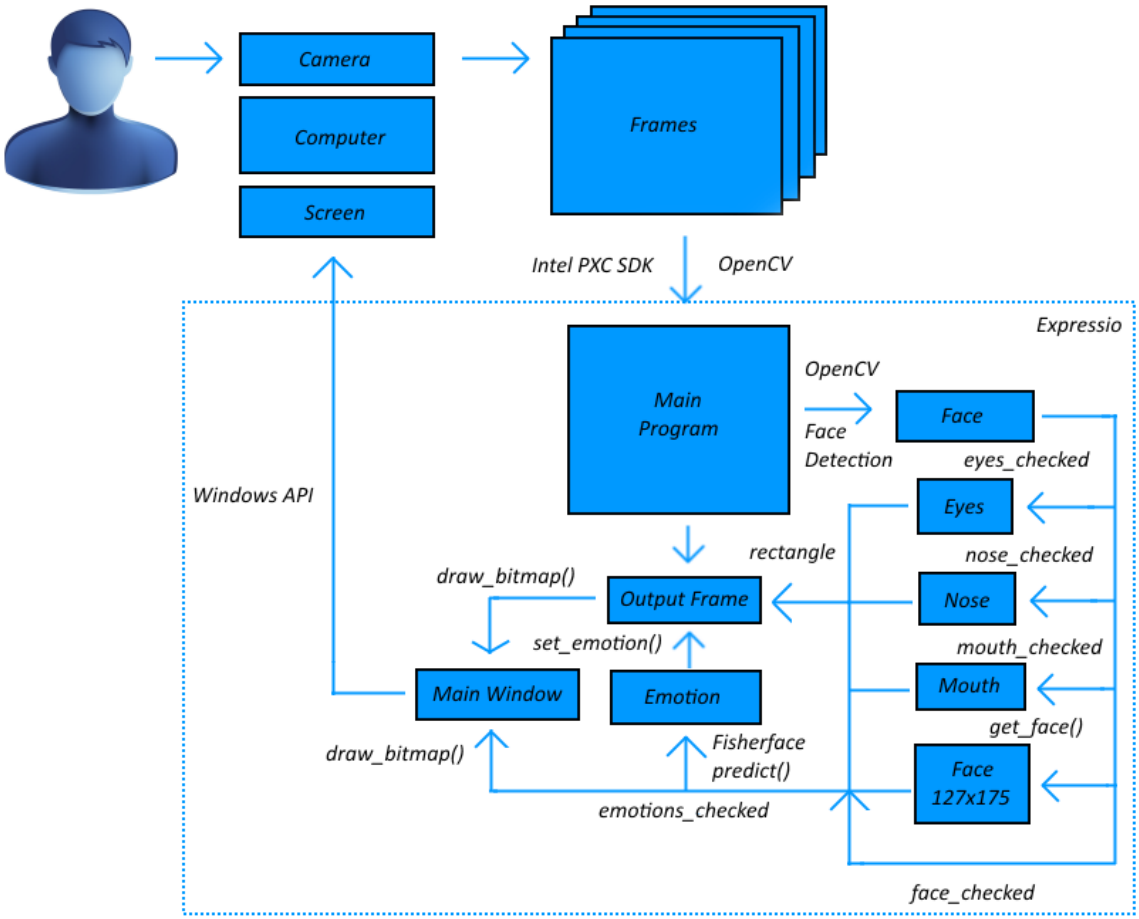


Fig. 15. Main Program Architecture

The *Train* window* is a combination of the two. It displays camera information to the screen like the main program/window, but once you click *Done* it will start training the Fisherface model again (it can't just be updated, the LDA process is too complex). *Take Snapshot* will just save the image to the right directory (it's the same as labeling the image), taking into account what you checked. As you can see, even if you click *Cancel*, the pictures you take will be saved on disk, so you have to delete them manually but it won't train the model again, so *models/emo.yml* is safe. Full architecture is displayed in Fig. 16.

*not to be mistaken with the *Training Program :* this is a program with no graphical interface that I used to initially train my algorithm

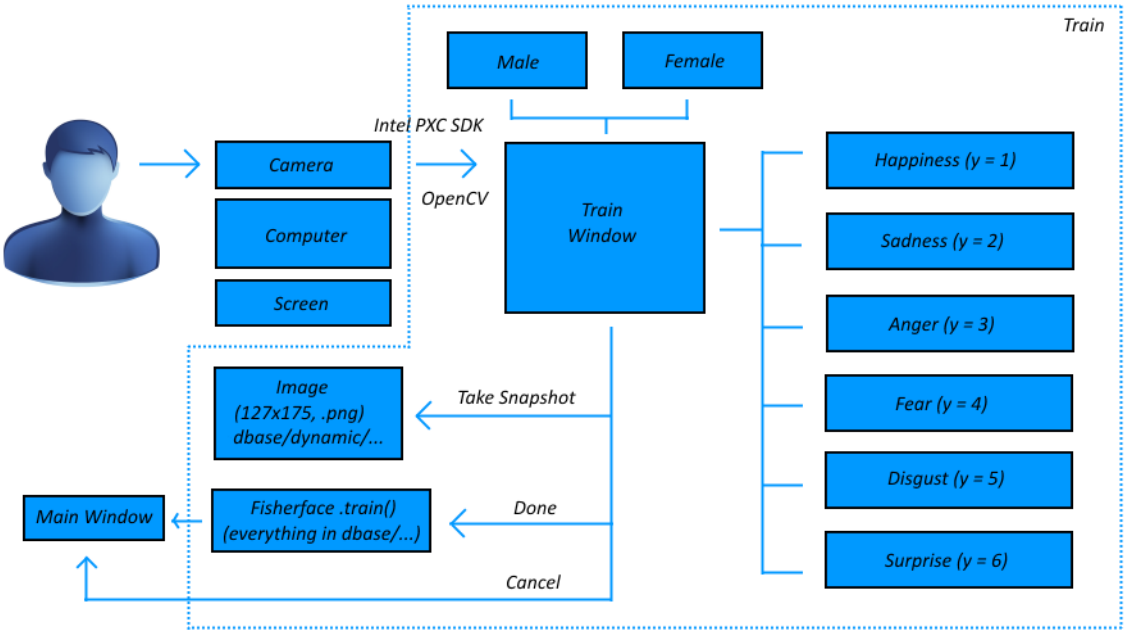


Fig. 16. Train Program Architecture

3.3 Software and Hardware Technologies

In this section I will talk about the software and hardware technologies I used. I will start with the Creative Senz3D camera and continue with the libraries I used for various tasks like capturing frames from the camera, computer vision and machine learning algorithms : Intel Perceptual Computing SDK and OpenCV. I will also talk about Windows API because I used it for the graphical user interface, and end with Microsoft Visual Studio 2012, the IDE that allowed me to combine all of the above.

Creative Senz3D. First off, this is a video camera, and a good one too, capable of streaming video at a resolution of 720p, 30 frames per second. What is really interesting about the camera is that it can also detect depth for object within a 1 meter range, giving you the ability to use 3D image analysis in your programs and create systems that can do 3D gesture detection or 3D face analysis. The initial motivation behind the camera was to expand the emotion detection algorithms to use the depth as well. However, such algorithms are too complex and something to look at in the near future. This device follows the Intel idea (the camera is designed to be used with Intel PXC SDK) of multimodal devices - capable of taking care of as many input streams as possible. With only this "camera" you can capture video streams, depth streams and even audio streams. It comes equipped with a dual-array microphone with noise-reduction and echo-cancellation that ensures clear voice pickup. Full specifications are available on Creative website [x].

Fig. 17. Creative Senz3D

Intel Perceptual Computing (PXC) Software Development Kit (SDK) is a collection of algorithms that will allow you to get data from external devices (those that support video, audio and depth data streaming for example) and process it in various ways. It uses these abstract units called *modules*. Each module deals with something different, for example one module will have code for audio or video capture, another one will have algorithms for face detection, object tracking etc. When I started working on my project, current SDK version was R7. In the meantime R8 was released and finally PXC turned into RealSense[x]. I upgraded to R8 but didn't use any of the new features, so what I will describe next might not be available for latest releases. Going back to modules, a module is responsible for one or several *devices*. Each device has, in turn, one or more *streams* that it can support. You can see Fig. 19 for a better picture. R7 had 9 modules implemented. You can see their names in Tabel 2.

Fig. 18. Intel SDK

In order to get frames from the Creative Senz3D camera, you would have to see what module 0 (DepthSense 325 Audio/Video Capture) has to offer (classes, methods), and use that to achieve your goal. Every program that uses the Intel PXC SDK must create a session first. If you don't

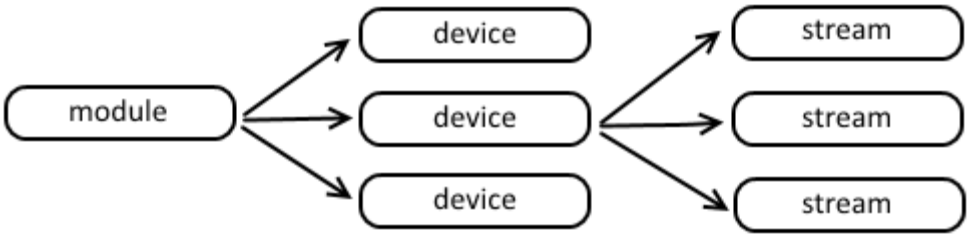


Fig. 19. Intel SDK Organization

Table 2

Modules Available in R7

Module	Name
0	DepthSense 325 Audio/Video Capture
1	Face Analysis
2	Hand/Finger Tracking and Gesture Recognition
3	Audio Capture (Media Foundation)
4	Video Capture (Media Foundation)
5	Markerless Object Tracking (Total Immersion*)
6	Segmentation 3DX
7	Voice Synthesis (Nuance*)
8	Voice Recognition (Nuance*)

create it, one will be created for you by default. A session is needed to maintain the SDK context. Regarding the code, a session is simply a *PXCSession* object. To find out information about a module, you can use the *QueryImpl* method, and to actually use it you would have to call the *CreateImpl* method and cast it to the appropriate class. All these are called from the session object. Similarly, *QueryDevice* and *CreateDevice* will allow device control while *QueryStream* and *CreateStream* will allow for stream configuration. You can see a sample code with what I just described in Fig. 20.

To change the device properties, you first have to query the device for available properties using *QueryProfile* and then, if you are satisfied with the new settings, apply them with *SetProfile*. Here is an example of changing the video settings for a camera. Let's assume that *dev* is a device with video streaming capabilities and that stream is the right number for the video stream. You can see how the code would look like in Fig. 21.

There are even easier ways to use the Intel PXC SDK (at least for device data capturing). *UtilPipeline* and *UtilCapture* are 2 classes that will allow you gather information from devices and configure them while hiding the *module* - *device* - *stream* architecture. I will talk more about them in the *Implementation* section. In my application I used the *UtilCapture* interface. You can see the overall Intel PXC SDK programming interface hierarchy in Fig. 22.

```
int profile = 0;
while (true) {

    vstream->QueryProfile(profile,&pinfo);

    /* we are looking for a profile with 640x480 settings */
    if (pinfo.imageInfo.width != 640 && pinfo.imageInfo.height != 480) {
        profile++;
        continue;
    }

    /* this is the right profile; just set it */
    vstream->SetProfile(&pinfo);

    /* now we can start streaming images from it; sp is a synchronization object */
    PXCSmartPtr<PXCImage> frame;
    PXCSmartSP sp;
    vstream->ReadStreamAsync(&frame,&sp);
    sp->Synchronize();

    /* data is ready in "frame"; do what you want with it */
}

/* we didn't find any profile for a 640x480 video streaming */
return -1;
```

Fig. 21. How to get Image Frames with Intel SDK

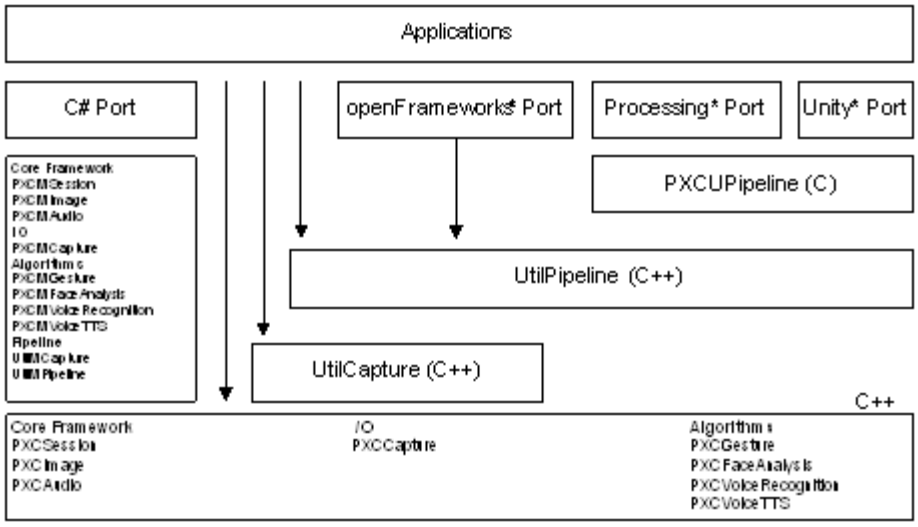


Fig. 22. Intel SDK Programming Interface Hierarchy

Open Source Computer Vision Library (OpenCV) - is an open source software library that comes with computer vision algorithms mostly, but it also has some machine learning algorithms. It can be used to get data from external devices as well. I managed to use it to obtain frames from Creative Senz3D camera. From my point of view it works faster than Intel PXC SDK R7 (especially

when it comes to normal laptop cameras). OpenCV purpose is to provide a common infrastructure to computer vision programs and it contains more than 2500 optimized algorithms (classic ones as well as state-of-the-art) that can be used for face detection and recognition, object identification, tracking moving objects, finding similar images from an image database etc. Many companies use OpenCV for tasks like inspecting labels on products, checking runways for debris, swimming pool drowning accidents detection, intrusions detection in surveillance videos any many others. The library has C++, C, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. It was initially developed at Intel Russia and it's free for use under the open source BSD license. It can take advantage of MMX and SSE instructions when available. I used version 2.4.9 in my project. You can find more about it at [x].

Fig. 23. OpenCV Logo

For the graphical user interface I used old **Win32 API** calls. This was the way to do it back when Windows 98 was around. My program starts with an *WinMain* function and I use procedure calls like *CreateWindow* to design the GUI. The technology might be old but it allowed me to do exactly what I wanted. More details are available on windows msdn website[x] and I recommend the book *Programming Windows API* (5th Edition) which was popular at the time.

Fig. 24. Microsoft Visual Studio Logo

Microsoft Visual Studio 2012 is an integrated development environment (IDE) from Microsoft. It can be used to create and develop complex projects on Windows. I used it to write the project code and to integrate other libraries (OpenCV and Intel SDK). It uses solutions - a set of code files and other resources that are used to build an application. A solution is made of one or multiple projects which are, in turn, made from source files and other resources. To include an external library I used *Property Manager*. Here you can set external library code to use, command line arguments and other useful things (for a shared library you basically need 3 directories : *include/*, *lib/* and *bin/* that can also be found in PATH). The nice thing about Property Manage is that you can save only particular changes into *".props"* files. If you need the same settings again later in another project, you can just use the props file you previously saved. For Intel SDK, I used *VS2010-12.Integration.MD.props* that came with the package and for OpenCV I created my own *opencv249.props* file.

3.4 Implementation

I will talk about emotion detection first, since it's the main part of my project, and then go on with object detection to explain face, eyes, mouth and nose detection. Finally I will talk about the graphical user interface, the flow of my program and end up with more details about my project.

Before going into emotion prediction, I want to say how the training process happens and explain the decisions I made. Like I said before, for emotion detection I used the Fisherface model/algorithm. My initial idea was to apply LDA on face images, followed by a simple machine learning algorithm, like logistic regression for instance. I wanted to first try it on the whole face and then move on to smaller sections of the face, like upper and lower parts (to try and predict AUs) for better results. In the time I had I just experimented with the whole face area, so AU detection is a thing for the future. One of the reasons I picked Fisherface is that it performs LDA as the first step. Secondly, in OpenCV documentation it states that Fisherface does a good job at classifying gender and it should do similar for emotion classification, despite the fact that it uses Nearest Neighbor as training algorithm. Fisherface is already implemented in OpenCV, you can see how to use it in Fig. 25.

```
cv::Ptr<cv::FaceRecognizer> emo_model;
cv::vector<cv::Mat> images;
cv::vector<int> labels;

/* create the model */
emo_model = cv::createFisherFaceRecognizer ();

/* read images */
read_images ("dbase/expressio/male/happiness/",L"dbase\\expressio\\male\\happiness\\*", HAPPINESS, images, label
read_images ("dbase/expressio/male/sadness/",L"dbase\\expressio\\male\\sadness\\*", SADNESS, images, labels);
...

/* train it */
emo_model->train (images,labels);

/* optionally save it to file */
emo_model->save ("models/emo.yml");
```

Fig. 25. Creating and Training a Fisherface Model in OpenCV

Once I decided to use Fisherface, my next task was to find an image database. I needed facial expression images that would express one of the 6 basic emotions. After searching on the internet, I came across *The Japanese Female Facial Expression Database* or *JAFFE*[x] which was exactly what I wanted : approximately 30 pictures for each emotion (a total of 183) from 10 female subjects. The images are in grayscale format with a size of 256x256 pixels. Each female subject has a unique identifier : KA, KL, KM, KR, MK, NA, NM, TM, UY, YM and contributed with 2-4 photos per emotion. In Table 3 I listed the number of pictures for each emotion.

Table 3

JAFFE Database

Emotion	Images
Happiness	31
Sadness	31
Anger	30
Fear	32

Disgust	29
Surprise	30
Total	183

In Fig. 26 you can see example of images from the database. To load the images into my program I used OpenCV.

Fig. 26. Examples from JAFFE

Taking into account that small number of subjects and pictures, I used the following method to test how *"good"* Fisherface would perform for emotion detection. I used 9 subjects for training and tested the prediction on the 10th one. For each subject I computed a *average recall :* first I computed the recall* for each emotion then I took the mean over all 6 emotions. Considering that the number of images is almost the same for each emotion, I think this is a good way of testing the algorithm, especially because it will be used in this way after I finish it (predicting emotions for new subjects, rather than same old ones). You can see the results in Fig. 27.

Person	Happiness	Sadness	Anger	Fear	Disgust	Surprise	Average
KA	0/4	0/3	1/3	2/4	2/3	3/3	41.66 %
KL	3/3	3/3	1/3	0/3	0/4	3/3	55.55 %
KM	3/4	0/4	1/3	0/3	0/2	3/3	34.72 %
KR	0/2	0/3	0/3	0/3	0/3	3/3	16.66 %
MK	3/3	0/3	3/3	3/3	3/3	2/3	77.77 %
NA	3/3	2/3	0/3	3/3	1/3	0/3	50.00 %
NM	2/3	1/3	0/3	3/3	0/2	3/3	50.00 %
TM	3/3	1/3	3/3	2/3	0/3	2/3	61.11 %
UY	3/3	0/3	1/3	3/3	3/3	0/3	55.55 %
YM	3/3	3/3	2/3	1/4	1/3	3/3	70.83 %
Average	40 %	33.33 %	54.16 %	74.16 %	33.33 %	73.33 %	**51.38 %**

Fig. 27. Fisherface Results, First Try

*recall is a quality number used in machine learning; its formula is *recall = TP / (TP + FN)*, where TP = True Positives, FN = False Negatives

These aren't amazing results, but for emotion detection they are really good. Consider that a random prediction would have an average around *16.66%*. Just out of curiosity, after this I computed the recall when I just take one picture out of the training set, for each person and for each emotion (so prediction for people that are already in the training database). The results were amazing this time. You can see them in Fig. 28.

Person	Happiness	Sadness	Anger	Fear	Disgust	Surprise	Average
KA	OK	OK	OK	OK	OK	NO	83.33 %
KL	OK	OK	OK	OK	OK	OK	100 %
KM	OK	OK	NO	OK	OK	OK	83.33 %
KR	NO	OK	OK	OK	OK	OK	83.33 %
MK	OK	OK	OK	OK	OK	OK	100 %
NA	OK	OK	OK	OK	NO	OK	83.33 %
NM	OK	OK	OK	OK	NO	OK	83.33 %
TM	OK	OK	OK	OK	OK	OK	100 %
UY	OK	OK	OK	OK	OK	OK	100 %
YM	OK	OK	OK	OK	OK	OK	100 %
Average	90 %	100 %	90 %	100 %	80 %	90 %	**91.66 %**

Fig. 28. Fisherface Results, Second Try

This is what gave me the idea to create a *Train Window* for my program, so you can train it with your own pictures for much better results (and it really works now). Next I tried to optimize the JAFFE database for better results. You can see that the unprocessed images have extra details that doesn't help at all the emotion classification process. Even if the background is removed and even with LDA getting rid of many pixels, it's still better to manually remove the sections that don't have anything to do with emotions. The best database would be with just the face (so you could see the different contracted muscles). Using smaller images will also help the training process since there will be fewer dimensions and it will converge faster (the prediction should be faster as well). I found a nice Python script on the OpenCV website[x] that can extract only the face from multiple pictures. I used it and got a new database with images similar to the ones in Fig. 29.

Fig. 29. Examples from Modified JAFFE

The script did a good job, although not perfect. You can still see the neck in most pictures. The new image size is 127x174 so the number of dimensions was reduced from 65.536 to 22.098!

After doing the same tests again, I obtained the results from Fig. 30. I didn't expect such an improvement honestly, new model seems to be better at predicting with *+13.20%*.

Person	Happiness	Sadness	Anger	Fear	Disgust	Surprise	Average
KA	0/4	1/3	1/3	4/4	2/3	3/3	55.55 %
KL	3/3	3/3	1/3	0/3	0/4	2/3	50.00 %
KM	4/4	4/4	2/3	0/3	0/2	3/3	61.11 %
KR	0/2	2/3	3/3	0/3	0/3	3/3	44.44 %
MK	3/3	3/3	3/3	2/3	3/3	3/3	94.44 %
NA	3/3	2/3	3/3	2/3	2/3	2/3	83.33 %
NM	3/3	2/3	0/3	2/3	0/2	3/3	55.55 %
TM	3/3	0/3	3/3	2/3	0/3	3/3	61.11 %
UY	3/3	3/3	0/3	0/3	3/3	0/3	50.00 %
YM	2/3	3/3	3/3	3/4	3/3	3/3	90.27 %
Average	63.33 %	43.33 %	47.50 %	76.66 %	73.33 %	83.33 %	**64.58 %**

Fig. 30. Fisherface Results, Third Try

With this in mind I tried to improve the database even further. I tried to write a function that will do the perfect face crop and see what the results are then. My idea was to first detect the face, and then change it to the right size. For face detection I had 2 choices : OpenCV or Intel SDK. Intel SDK's Face Analysis module doesn't seem to work on static images and it didn't detect any face from JAFFE, so I just had to use OpenCV again. The face detection algorithm is Viola-Jones, and I will describe it later in this section. The problem with face detection was that the resulting face rectangle was not of constant ratio. Another problem was that in order to use Fisherface on new faces, I had to reshape them to the same size as the training images. The first thing I had to do was to set on a width and a height for my images. With a simple image editing software (I used paint.net[x], it's good and free) I tried to find out a good size that would fit the whole face in most images. After some experiments, my result was that 127x175 (22.225 dimensions) is a good size to use.

The problem was to convert any sized frame of *width x height* that doesn't have a fixed aspect ratio to a small 127x175 one that still has a normal looking face in it. A simple reshape would deform the face (must be same aspect ratio as 127:175, which is roughly 0.72). I tested several ideas that can achieve this and finally used the best one. The function that does this is called *get_face()* and receives an argument called *method* that can be any of the numbers 1, 2 or 3. My first attempt was to compute the center of the frame (usually it's the nose) and go half of 127 left and right, and half of 175 up and down so that the final image will be of 127x175 size. This was not a good idea since for big frames the result would be just the nose. However, you can still use this method by calling *get_face()* with *method = 3*. Next idea was to start with the height of the frame (I had problems with width, I needed a smaller one that would fit perfectly the face), see what's the closest multiple of 175 near it, take that size as new height and 0.72 of it as the new width and reshape it to 127x175 (same aspect ratio). The reshaping function is implemented in OpenCV and it uses linear interpolation. You can see the actual code in Fig. 31. This performed better that method 3, but if you

moved close to the camera and then far away, you would notice discrete jumps in the resulting frame. You can still use this method by calling *get_face()* with *method = 2.*

```
int k = (*face).height / height;
face_rect.height = k * height;
face_rect.width = k * width;
```

Fig. 31. Method 2 of get_face()

```
resize ((*src), (*dst), cv::Size (width,height), 0, 0, cv::INTER_LINEAR);
```

Fig. 32. Calling resize() in OpenCV

The last idea I tried was also the best and simplest one. Instead of finding the closest multiple of 175, just take the height and use for width (*0.72 * height)*. The resulting image has the same aspect ratio so I just had to call *resize()* to 127x175 from OpenCV. This gave the best results and it's currently the default method used in *get_face()*. If you want to use it, call the function with *method = 1* as argument.

Using method 1 for face cropping, I got a new JAFFE database, as seen in Fig. 33. This time there is no neck or shirt in the picture, just the face. This seems to be the best possible result so I stopped here.

Fig. 33. Examples from Modified JAFFE with Method 1

I did my initial test all over again and the results were surprising. You can see what I got in Fig. 34. Instead of an upgrade, the new rate was *4.86%* less accurate than before. My only explanation was that the bigger input size (22.225 dimensions vs 22.098) and luck had something to do with it. The Python script was reshaping the image, keeping the eyes positions constant so this probably made the rate a bit better as well. Even if I got worse recall, I kept using this function since it was implemented in my project, it was automatic and not just a script you had to manually edit to work with your new images. Anyway, instead of using Japanese women as the final training database for my program, I decided to create my own. It's called *Expressio Database* and it has 3 males, 1 female subjects, with a total of 141 images. It's not a consistent database (most subjects didn't take photos for emotions like fear or disgust) so I didn't run the average recall tests on it.

Person	Happiness	Sadness	Anger	Fear	Disgust	Surprise	Average
KA	0/4	3/3	1/3	3/4	3/3	3/3	68.05 %
KL	3/3	2/3	1/3	0/3	0/4	3/3	50.00 %
KM	4/4	4/4	2/3	3/3	1/2	3/3	86.11 %
KR	1/2	0/3	2/3	0/3	1/3	3/3	41.66 %
MK	3/3	1/3	3/3	1/3	3/3	3/3	77.77 %
NA	3/3	2/3	3/3	3/3	0/3	1/3	66.66 %
NM	0/3	2/3	0/3	3/3	0/2	3/3	44.44 %
TM	3/3	0/3	1/3	0/3	1/3	2/3	38.88 %
UY	1/3	1/3	0/3	3/3	1/3	0/3	33.33 %
YM	2/3	3/3	3/3	3/4	3/3	3/3	90.27 %
Average	65.00 %	56.66 %	53.33 %	58.33 %	45.00 %	80.00 %	**59.72 %**

Fig. 34. Fisherface Results, Fourth Try

Expressio Database

Males

Mihail Dunaev

Lorin Bobulișteanu

Females

Andreea Martinovici

Maria Pârcălăbescu

[pic] George Milescu TODO

Fig. 35. Expressio Database

In my system, I just load the *models/emo.yml* file generated with the Expressio Database and if the Emotion label is checked, in the main loop I first get the face region of size 127x175 using *get_face()* with *method = 1,* and apply the Fisherface prediction. If Emotion Sample Rate is greater than zero, I wait for a certain amount of time to pass between two consecutive emotion predictions. You can see the end results in the *Results & Conclusion* section. For gender detection I used Fisherface as well, trained with Expressio Database but with 2 classes only : males & females.

Object detection is another feature of my project. Even if I used procedures and XML files from OpenCV, I would like to say a few things about it. Like I already said many times, it uses Viola-Jones. I'll talk about the prediction part first. In a classical object detection algorithm we would first train our program with pictures containing the object and pictures containing other random stuff. In the prediction part, using a sliding window, we would move across the big image and see if we can predict our object, and then re-size it and go for the whole picture again and so on, just like in Fig.36 (the red window is the *sliding window*). Viola-Jones works in a similar way. The trick about it is that it tries to get rid of non-object windows as fast as possible. So instead of a classical predict on each window, it has several *stages* that do less computational work and can reject non-object windows faster. For example, if we look for a face in an image, we might detect one or even a couple of them, but if we take all possible sub-windows of the image, in most of them there won't be a face. So it's not a good thing to apply the most accurate prediction in each since this takes some time. Instead, we can apply a really weak prediction that is fast and if it passes we will move to a stronger one and so on. These concept of multiple prediction stages is called *cascading* (Viola-Jones is referred to as a *Cascade Classifier*). I illustrated the principle in Fig. 37.

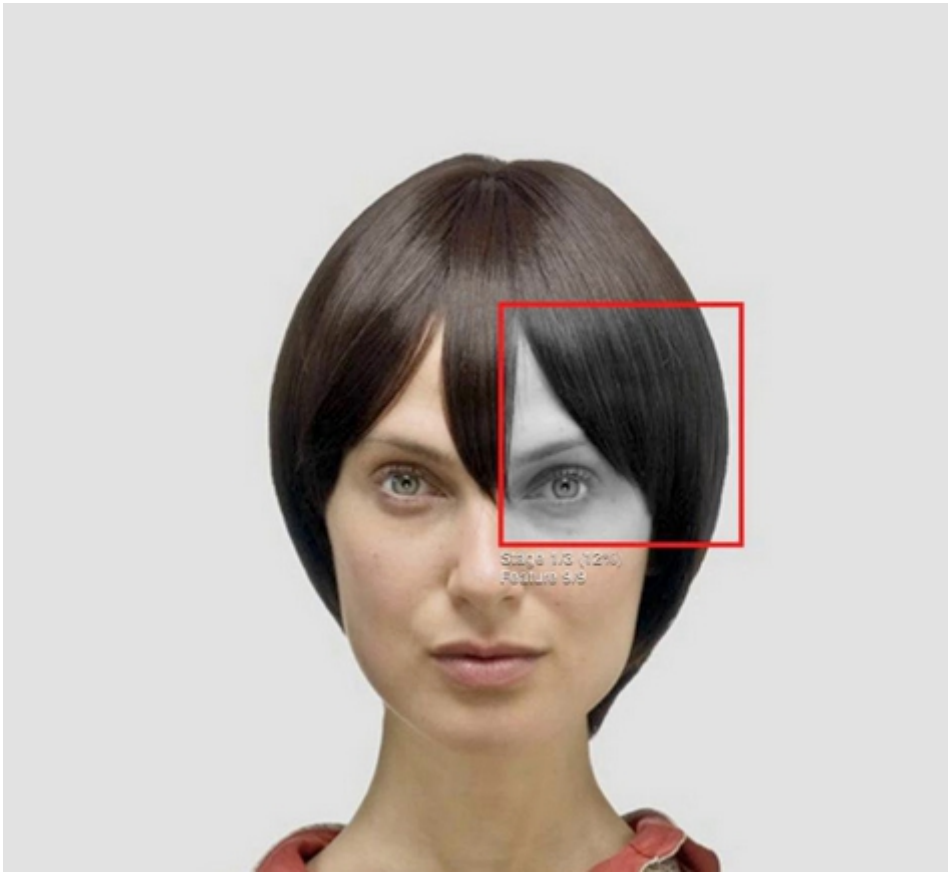


Fig. 36. The *sliding window* approach

In each stage of the cascade, Viola-Jones tries to apply certain features on the window. The features Viola-Jones is using are patterns of black and white pixels called Haar features (hence the name *Haar Cascade*). You can see examples of Haar features in Fig. 38. These Haar features can be of any size and positioned anywhere in the window. Applying a Haar feature means that we take the pixel values under its region (from the original image), and if the Haar pixel is black we add the value to a sum and if it's white we subtract it. For a window to *pass* a Haar feature the resulting sum must be under a certain threshold. What the algorithm does at a certain stage is to apply several Haar features on the image, and if it passes all of them (the computed value is always under the threshold), it moves to the next stage. Let us look at an example : let's say the current window is the

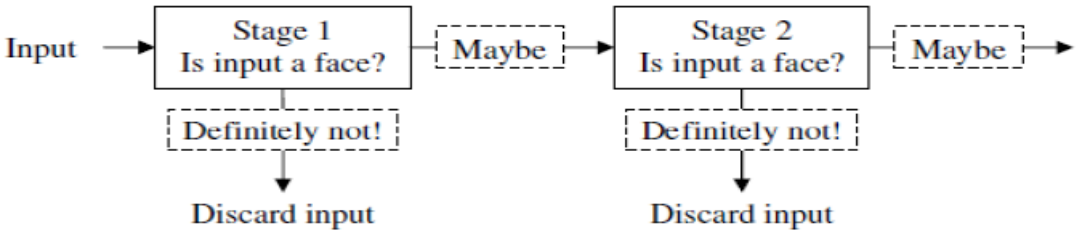


Fig. 37. Cascading in Viola-Jones

one in Fig. 39. The algorithm tries to detect a face, so it will try the feature displayed in the image [*] and compute the *feature value* by summing the pixel intensities under the black region and subtracting the ones under the white region. If the feature value is under a certain threshold it will move on to the next Haar feature, else it will stop and try a different window.

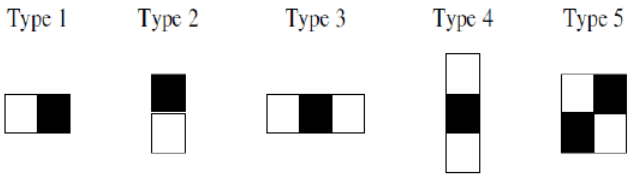


Fig. 38. Haar features used in Viola-Jones

Fig. 39. Haar feature applied in a window

What might still be confusing is the Haar features the algorithm applies on each stage, the threshold values and how many stages there are. All these details are determined in the training process. It's obvious that for different objects you will have different stages, with different combinations of features and different threshold values. When training to detect a particular object, Viola-Jones uses Adaboost, a machine learning algorithm that's rather complex and I will not cover in this paper. If you want to learn more about it, feel free to read the paper *A Short Introduction to Boosting*[x]. One more thing I would like to say about Viola-Jones is that in order to compute the feature value fast, it first transform the initial image to what's called *the integral form.* In integral form, all you need to compute the area of a region is the value at its corners. It basically sums up all pixel values from up and left into the current pixel value.

To use Viola-Jones with OpenCV is really simple. You first have to create a *CascadeClassifier* object, train it with your own data or load the values from an XML file and just use it to detect new objects of the same type in your image. You can see how I used it in my project in Fig. 40. The XML files I used are the ones from *haarcascades* directory in opencv (*sources/data/haarcascades/*).

[*] you can think that the feature looks for a nose between two black eyes if it's something similar to that, the computed value will be under the threshold

Expressio has several source and resource files. I will try to explain each of them.

main.cpp	-	contains the main function; it has code to deal with the GUI
main.h	-	header file for *main.cpp*; it has basic drawing functions used in *expressio.cpp* (it's included there as well)
expressio.cpp	-	contains the code that deals with emotion and object detection
expressio.h	-	header file for *expressio.cpp*; it's included in *main.cpp*
resource1.h	-	contains definitions used to identify GUI elements
expressio.rc	-	contains information about GUI elements, like position and size

Expressio starts running with the *wWinMain* function from *main.cpp*. The small *w* stands for *wide* and it means that the command line arguments are passed in UTF-16 (2 bytes per character) instead of ASCII format (1 byte per character). On Windows you can use both wide (if *UNICODE* and *_UNICODE* are defined) and normal characters and while some functions will take as arguments the wide type, others will work only with 1-byte type so throughout my program I had to do many conversions between the two.

After registering window controls and some initializations, the program will create all my dialog windows with a call to *CreateDialogW*. A dialog window is a window that can has a menu and a status bar. You can see an example in the *General* section. When designing a GUI in Windows, everything is a window. So you can use same functions for multiple elements. You address a window with a *HWND* variable (it's not a pointer, it's just a number in a table). You can see some code example in Fig. 41 and Fig. 42.

```
/* create main dialog window */
h_main = CreateDialogW (hInstance,MAKEINTRESOURCE(IDD_MAINFRAME),0,main_window_handler);

/* main window status bar */
h_status_bar = CreateStatusWindow (WS_CHILD|WS_VISIBLE,L"OK",h_main,IDC_STATUS);

/* create train dialog window */
h_train = CreateDialogW (hInstance,MAKEINTRESOURCE(IDD_TRAIN),0,train_dialog_handler);

/* train window status bar */
h_status_train = CreateStatusWindow (WS_CHILD|WS_VISIBLE,L"OK",h_train,IDC_STATUS_TRAIN);

/* emorec window */
h_emorec = CreateDialogW (hInstance,MAKEINTRESOURCE(IDD_EMOREC),0,emorec_dialog_handler);
```

Fig. 41. Creating Dialog Windows

```
SetWindowPos (h_main,HWND_TOP,wx_mw,wy_mw,width_mw,height_mw,SWP_NOZORDER);
SetWindowPos (h_train,HWND_TOP,wx_tw,wy_tw,width_tw,height_tw,SWP_NOZORDER);
ShowWindow (h_main,SW_SHOW);
```

Fig. 42. Changing Window Properties

In the initialization process it will also try to load the models needed for the machine learning algorithms. The program will look for the following files in *models/* directory :

face.xml	-	face detection
eye.xml	-	eyes detection
nose.xml	-	nose detection
mouth.xml	-	mouth detection
gender.yml	-	gender detection
emo.yml	-	emotion detection

If it doesn't find any of these files it will move on but won't be able to perform that detection. Next it will mark *h_main* as the current window and enter an infinite loop waiting for events (called messages) from the user. If the user clicks the Start button, a new thread will be created that will execute one of the following functions : *capture_video_opencv* or *capture_video_util_capture*. You can see the main window loop code in Fig. 43, the main opencv loop code in Fig. 44 and main util capture loop code in Fig. 45 (*Addendum*). *UtilCapture* loop uses Intel SDK methods to obtain data from the camera. You can notice the extra convert when using Intel SDK. To apply OpenCV algorithms I had to convert images from PXCImage structure (used by Intel SDK) to Mat structures (used by OpenCV). The convertion is straighforward, I just saved all the pixel values from one structure to another. You can see code example in Fig. 46 (*Addendum*).

```
/* main loop */
MSG msg;
int status;
h_current = h_main;
while (true) {
    status = GetMessageW(&msg,h_current,0,0);
    if (status == -1)
        return status;
    TranslateMessage (&msg);
    DispatchMessage (&msg);
}
```

Fig. 43. Main Window Loop

```
    /* video capture */
    cv::VideoCapture capture (device);
    if (!capture.isOpened())
        return -1;

    /* main loop */
    while (!g_stop) {

        capture >> frame;
        if (frame.empty())
            return -1;

        /* process it */
        process_frame (hwndDlg,frame);

    }
```

Fig. 44. Main OpenCV Loop

4. Results and Conclusion

In the *Implementation* section I stated that the *Expressio Database* is too inconsistent to try the *average recall* test on it. I think the best way to test the application is to actually start it and see how it performs on emotion detection in real-time and what feel you get from it. I was really happy with the results. The program was able to correctly identify all emotions I simulated, with few errors. In my experience the best value for *Emotion Sample Rate* was 200 ms which allowed for natural changes in the emotion detected. With a 0 ms *EMR* you might experience some fast oscillations between two emotion states with particular facial expressions. I think that the best way to show my results is with pictures. You can see in Fig. 47-52 all the emotions detected by Expressio.

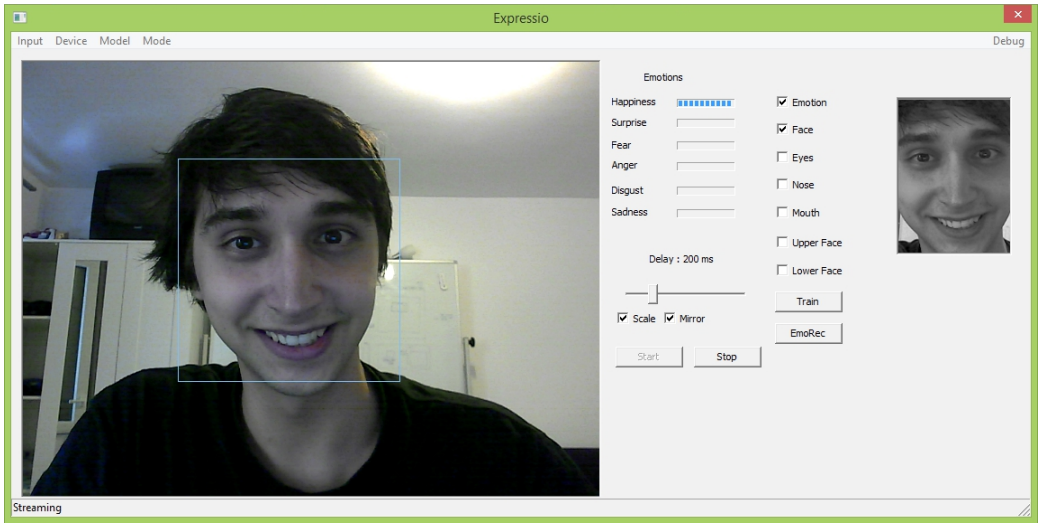


Fig. 47. Happiness

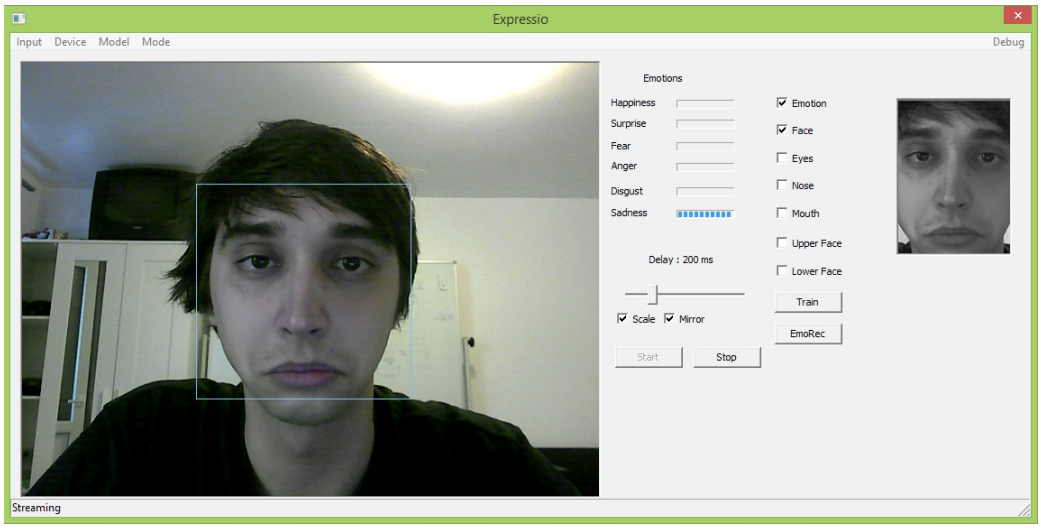


Fig. 48. Sadness

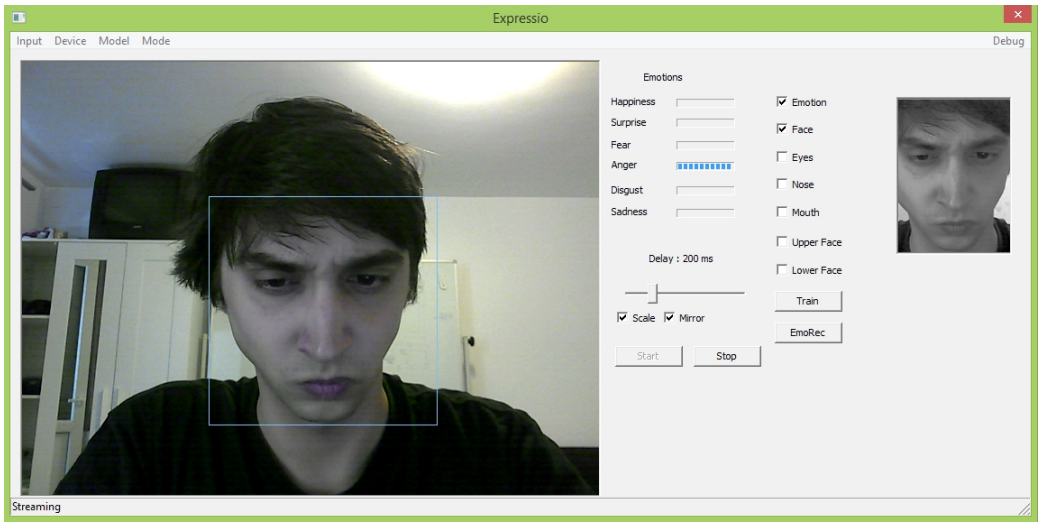


Fig. 49. Anger

Fig. 50. Disgust

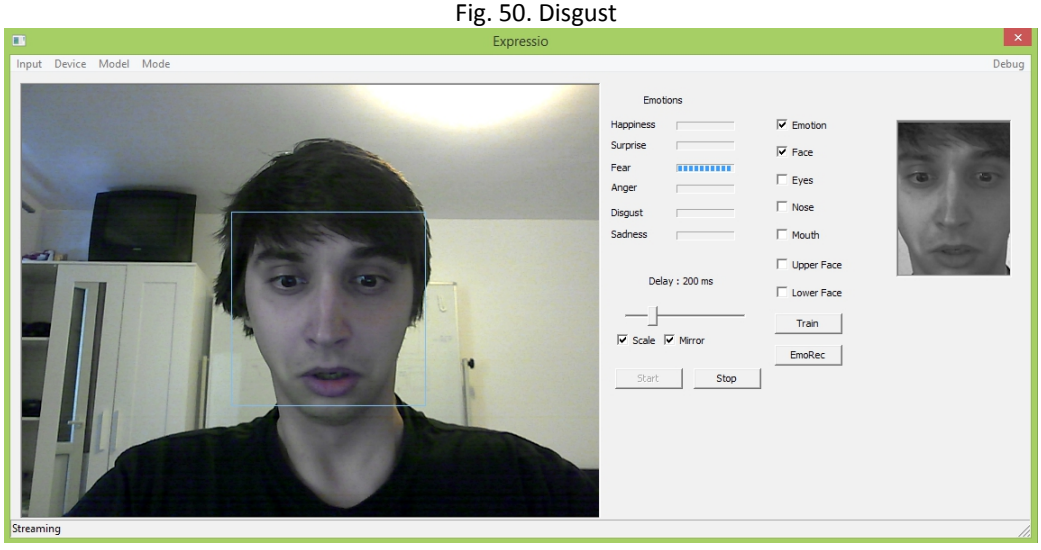


Fig. 51. Fear

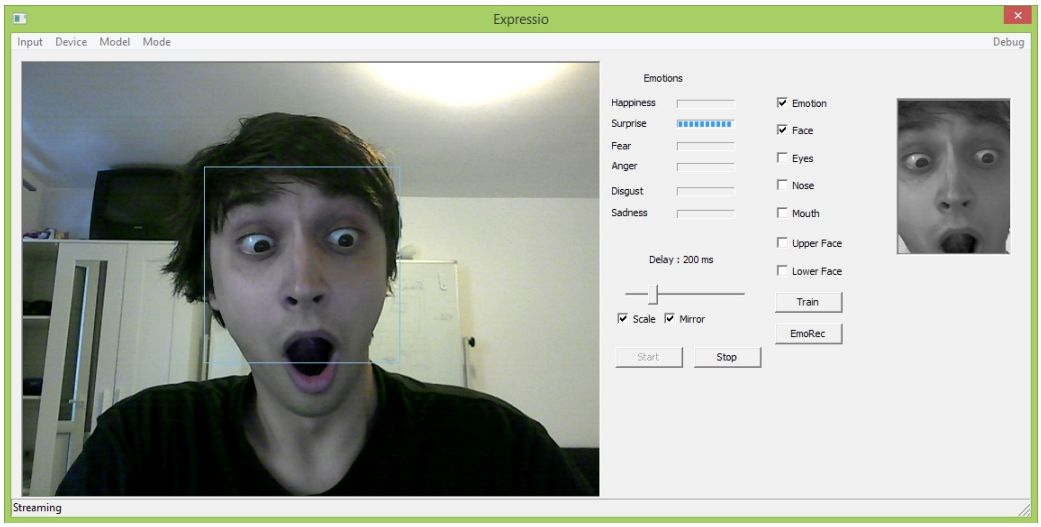


Fig. 52. Surprise

If you use all the features of Expressio (like emotion, face, eyes detection) you'll notice a drop in frame rate. This depends on the amount of light in the room. However, the program still has a real-time feel to it and this can be improved in the future with object tracking instead of object detection with each frame. To get good emotion detection rate I recommend that you train your program with as many images as possible, from as many angles as well. To get the above results I used 17 images with myself for each emotion from different angles.

In conclusion I am really satisfied with what I achieved. Even if the program won't have good emotion detection rate with the Expressio Database for everyone, the Train Window can overcome this by allowing the user to use his or her own pictures to train it. It gives you the possibility to create emotion labeled databases really quick. Looking into the future, there are few things that can be improved in Expressio. First, the cropped face can be split in two regions, the upper and the lower face. We could use those to train a Fisherface model with AU and predict those instead. This can result in an improved emotion detection rate. Another optimization would be to replace the Nearest Neighbor algorithm applied at the end with a better machine learning algorithm, like logistic regression. This would allow us to predict probabilities for emotions and give a more natural feel to Expressio.

Addendum

```
/* first create a session; PXCSmartPtr is just a pointer type */
PXCSmartPtr<PXCSession> ses;
PXCSession_Create(&ses);

/* info about module 0 */
int module = 0;
PXCSession::ImplDesc desc;
ses->QueryImpl(0,module,&desc);

/* create a "module object" with a PXCCapture cast */
PXCSmartPtr<PXCCapture> cap;
ses->CreateImpl<PXCCapture>(&desc,&cap);

/* as you can see we can't create impl from module number
 * we need the ImplDesc object first (object info) */

/* info about first device */
int device = 0;
PXCCapture::DeviceInfo dinfo;
cap->QueryDevice(device,&dinfo);

/* here we can just create it from device number */
PXCSmartPtr<PXCCapture::Device> dev;
cap->CreateDevice(device,&dev);

/* same goes for streams */
int stream = 0;
PXCCapture::Device::StreamInfo sinfo;
dev->QueryStream(stream,&sinfo);

/* create the stream object  */
PXCCapture::VideoStream::ProfileInfo pinfo;
PXCSmartPtr<PXCCapture::VideoStream> vstream;
dev->CreateStream<PXCCapture::VideoStream>(stream,&vstream);
```

Fig. 20. Getting your way around Intel SDK

```
cv::CascadeClassifier face_cascade;
cv::CascadeClassifier eye_cascade;
cv::CascadeClassifier nose_cascade;
cv::CascadeClassifier mouth_cascade;

...

/* face detection */
if (!face_cascade.load ("models/face.xml"))
    disable_face (hwndDlg);

/* eyes detection */
if (!eye_cascade.load ("models/eye.xml"))
    disable_eyes (hwndDlg);

/* nose detection */
if (!nose_cascade.load ("models/nose.xml"))
    disable_nose (hwndDlg);

/* mouth detection */
if (!mouth_cascade.load ("models/mouth.xml"))
    disable_mouth (hwndDlg);

...
face_cascade.detectMultiScale (frame_gray, faces, 1.1, 3, 0, cv::Size(127,175), cv::Size(1280,720));
...
eye_cascade.detectMultiScale (face_image, eyes, 1.4, 3, 0, cv::Size(45,45), cv::Size(60,60));
...
nose_cascade.detectMultiScale (face_image, noses, 1.1, 3, 0, cv::Size(55,55), cv::Size(70,70));
...
mouth_cascade.detectMultiScale (face_image, mouths, 1.1,40, 0, cv::Size(80,40), cv::Size(130,80));
```

Fig. 40. Object Detection in OpenCV with Viola-Jones

```
/* main loop */
while (!g_stop) {

    /* get frame */
    uc.QueryVideoStream(0)->ReadStreamAsync (&image,&sp);
    sp->Synchronize ();

    /* draw it to window */
    draw_bitmap (hwndDlg,image);
    update_panel (hwndDlg,BIG_IMAGE);

    /* convert it to cv format & process it */
    convert_pxc_to_cv (image, &image_cv);
    process_frame (hwndDlg, (*image_cv));

    /* free memory */
    delete image_cv;
    image->Release ();
}
```

Fig. 45. Main UtilCapture Loop

```
i = 0;
for (row = 0; row < image_cv->rows; row++)
    for (col = 0; col < image_cv->cols; col++) {
        if (info.format == PXCImage::COLOR_FORMAT_RGB24) {
            image_cv->at<cv::Vec3b>(row,col).val[0] = data.planes[0][i];
            image_cv->at<cv::Vec3b>(row,col).val[1] = data.planes[0][i+1];
            image_cv->at<cv::Vec3b>(row,col).val[2] = data.planes[0][i+2];
            i = i + 3;
        }
        else if (info.format == PXCImage::COLOR_FORMAT_GRAY) {
            image_cv->at<uchar>(row,col) = data.planes[0][i];
            i++;
        }
    }
```

Fig. 46. PXCImage to Mat Conversion

B I B L I O G R A P H Y

The Bibliography title is written in T_bibliography title, Calibri 12 pct, all caps, expanded spacing with 2 pt, centered.

References are numbered according to the order of their appearance in text. Text references are indicated by the number of the title between square brackets, such as [2].

Each bibliographic entry should mention: initial of the first name, family name, title, publishing house, place and year or periodical title, volume, number, year, initial and final pages. Authors names are written in italics (10 pct).

For Romanian books and articles a translation of the title will be inserted in parentheses.

[1] C. Soviany, Embedding Data and Task Parallelism in Image Processing Applications, PhD Thesis, Technische Universiteit Delft, 2003

[2] A. Mauthe, D. Hutchison, G. Coulson and S. Namuye, “Multimedia Group Communications Towards New Services”, in Distributed Systems Eng., vol. 3, no. 3, Sept. 1996, pp. 197-210

[3] V. I. Arnold, Metodele matematice ale mecanicii clasice (Mathematical models of classic mechanics), Editura Ştiinţifică şi Enciclopedică, Bucureşti, 1980

[4] *** COSMOS/M – Finite Element System, User Guide, 1995.