Automatic Action Unit Recognition Using Computer Vision

Final Report

Derek Schultz

December 14, 2013

1 Introduction

Facial expressions are one of the primary methods humans use to convey and perceive emotions. In order for machines to be able to properly detect the emotional state of human users, facial expression recognition is an important capability. Using Paul Ekman's action unit system[1], emotions can be classified based on the actions of facial muscle groups. Automatic detection of these action units by means of computer vision is the goal of this project.

With this as my first experience working with computer vision, I relied heavily on the work of Pantic and Valstar[4] for guidance on which algorithms to use to achieve AU detection. In the interest of time, I elected to adopt many simplifications on their methods.

OpenCV[2] was used in order to access image processing APIs and an abstracted interface to pixel data. OpenCV provided functions for Haar cascade classifiers, boosting algorithms, filtering, and other computer vision and machine learning related primitives.

2 Methods

2.1 Face Detection

The image was first converted to grayscale and equalized, resized relative to the size of the face in the frame, and a Haar cascade classifier was used to locate the face in the image. Before specific landmarks could be detected, it was important to separate the detected face into regions of interest (ROIs). By increasing the precision of the ROIs, the machine learning phase had fewer chances to make mistakes and less computational effort was required. The method used by Pantic and Valstar made use of histogram data to locate the eyes and mouth. I ran into roadblocks early on with this method and quickly abandoned it, hoping to return later. I instead implemented a ratio-based method based on trial and error. I hand selected certain ratios for the placement of each region of interest. For example, the eyes are bounded on the top at 1/3 of the height of the face and on the bottom

at 1/2 of the height of the face. My restraints had to be quite loose to accommodate the variety of facial structures and the range of movement of the facial features during facial expressions. This was likely one of the reasons the performance of my program suffered, a topic I will go into with more detail in section 3.

2.2 Facial Landmark Detection

Within each region, the specific landmark points were identified using feature patch templates. The features were based upon the gray level intensities and properties of Gabor filters[3]. The Gabor filters helped to eliminate noise from uncertainties such as color and lighting conditions[4]. They work well for edge detection and are used extensively in computer vision because they approximate the function of a neuron found in the visual cortex.

A total of 48 gabor filters were used: eight rotations and six different frequencies. The 48 gabor filters were applied to 13x13 pixel feature patches. The 13x13x48 pixel values, along with another 13x13 for just the plain grayscale pixel values created one feature vector of size 8281.

During the training phase, these large feature vectors were generated for the indicated pixels, as well as the eight pixels immediately adjacent on each side. Nine pixels were chosen at random from the surrounding area to serve as negative training samples.

GentleBoost was used as the classifier, as per the recommendation of Pantic and Valstar. GentleBoost is reported to have better success than AdaBoost and other boosting algorithms in facial feature detection applications.

In total, there were 18 facial features, each with 8281 features. With 40 images in the training set, the training took several minutes and occupied almost 700MB on disk. Surprisingly, a training set of only 40 images did produce fairly accurate results and I decided this was all that was necessary for my purposes.

The 40 training images came from the Cohn-Kanade facial expression database[5]. This database provided video clips of 120 different subjects

performing a facial expression. Facial feature landmark data was provided. Action units for each expression sequence were also noted.

During the classification/prediction step, the trained GentleBoost classifier was run for every facial feature on every pixel in the appropriate ROI. While classification greatly outperformed training (as it should), the demands of classification were actually far greater, so the speed performance of the algorithm was not great. For every image, thousands of classifications needed to be made (one for each pixel). For each facial feature, the vote counts for the pixels that classified positively were compared and the pixel with the highest confidence was chosen to represent the landmark.

2.3 Action Unit Predictions

Once facial landmarks are established, there exists a set of data that can be used to recognize the action units, which is the core purpose of the program. There exist exactly 18 facial landmarks that were discovered in section 2.2. Because the data is coming from a video feed, there are $18n$ data points, where n is the number of frames captured.

Pantic and Valstar used that data to create some very more very large feature vectors, just like the ones from section 2.2. They compared the position of every pixel of every frame to their positions on the first frame. They also compared the distances of all the pixels with respect to one another, and the differences of those distances compared to the first frame. Here are those features expressed more formally:

$$f_1(p_i) = p_{i,y,n} - p_{i,y,1} \tag{1}$$

$$f_2(p_i) = p_{i,x,n} - p_{i,x,1} \tag{2}$$

$$f_3(p_i, p_j) = \| p_i - p_j \| \tag{3}$$

$$f_4(p_i, p_j) = f_3(p_i, p_j) - \| p_i, 1 - p_j, 1 \| \tag{4}$$

i and j represent any two pixels, x and y are the coordinates of either i or j and the n indicates the frame.

From these features, a boosting algorithm (for example, AdaBoost) selects the best performing ones. A support vector machine is run using the chosen features for each possible action unit.

It was at this point where I decided to stray extremely far from Pantic and Valstar's methods. There were many problems with this implementation for the learning of action units for my specific application. For one, my program is quite slow. Calculating all 18 points for all n frames of video would take a long time. And given the amount of time I had put into the project already and the amount of time I had left, using AdaBoost for feature selection and SVMs for classification seemed like a very difficult implementation hurdle.

Given that I was successful with GentleBoost in section 2.2 and that I was already familiar with the code to make it run, I decided to use a simple GentleBoost classifier again. I decided the most important points in time were the beginning of the clip and the end of the clip (the onset and apex phases of the expression). Everything in between those points is essentially a linear motion. My hypothesis was that I would maintain most of the important information about the transition by keeping just those two frames.

The construction of the feature vector was very simple. For each of the 18 points on the face, the difference in x and the difference in y were calculated and placed in a vector of size 36.

A subset of 21 action units were chosen for training. Excluded action units were either not represented by the training set, or not possible to be expressed with the limited set of 18 facial feature landmarks.

Each action unit had 6-10 positive examples and 6-10 negative examples assigned to it. GentleBoost was again used as the classifier. In the classification stage, the feature vector for the face (given first and last frames of the expression animation) was passed to each of the 21 GentleBoost classifiers for the 21 action units. Each classifier responded yes or no. All classifiers responding yes were output as being active during that image sequence.

3 Results

Although I wish I had numbers to put in this section for the accuracy of my machine learning systems, that would unfortunately be very difficult. For the facial feature landmarks, I would need to write scripts to compare the output to the files and compute the euclidean distances, then run some statistical analysis on that. All of the action unit came in an unhelpful organizational structure inside an Excel sheet. Even as I type this report, I am well past the due time and do not have time to organize the data and write my own statistical analysis.

From an unofficial standpoint, I can say that my system performs "pretty well". When testing, I chose image sequences that I had never seen before from the test set at random and tried to guess the output and the program was right much of the time.

I did not expect the level of accuracy from the system that I ended up with. For much of the duration of the project, I was not sure if I would ever be able to place points on the face with any sort of accuracy, and without that framework, I would not be able to detect action units any better than random. I was very pleasantly surprised by the results and it was very exciting to suddenly see my program make seemingly intelligent decisions as I made breakthroughs late in the process.

4 Future Work

I enjoyed thoroughly enjoyed working on this project, and it is definitely a piece of software that I look forward to improving. The most glaring problem at this point in time is the speed performance. There were several optimizations in the Pantic and Valstar implementation that I excluded from mine. I would like to implement the missing histogram-based method for ROI calculation, as I mentioned earlier. This will allow for more efficient scanning of smaller regions for the landmarks. Another major missing component was the facial landmark tracking. There was a section of the

Pantic and Valstar paper dedicated to explaining the use of a particle filter to track the motion of the facial landmarks. As a novice in the field of computer vision, I had a difficult time understanding this section. I plan to investigate the use of particle filtering to learn how it can be applied to this problem.

The program I wrote is already well organized. I put some effort into architecting the code in such a way that I will be able to maintain, extend, and reuse it. The command line interface is somewhat intuitive, could be improved. Adding a graphical user interface with webcam functionality for real time feedback would be ideal.

5 References

[1] Ekman, Paul, "FACS". http://www.paulekman.com/facs/

[2] OpenCV. http://opencv.org/

[3] G. Donato, M.S. Bartlett, J.C. Hager, P. Ekman, T.J. Sejnowski, "Classifying Facial Actions", *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 21, No. 10, pp. 974-989, 1999.

[4] M.Pantic, M.F. Valstar, "Fully Automatic Facial Action Unit Detection and Temporal Analysis", *In Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop*, 2006

[5] P. Ekman, W.V. Friesen and J.C. Hager, "The Facial Action Coding System: A Technique for the Measurement of Facial Movement", *San Francisco: Consulting Psychologist*, 2002

[6] M.Pantic, M.F. Valstar, R. Rademaker and L. Maat, "Web-based database for facial expression analysis", In ICME'05, pp. 317-321, 2005