ESGI100: Gabor Filter Selection and Computational Processing for Emotion Recognition

Problem presented by

Matt Celuszak and Daniel Jabry

CrowdEmotion

Executive Summary

CrowdEmotion produce software to measure a person's emotions based on analysis of microfacial expressions detected using a webcam. The technology relies on a machine learning algorithm to recognize which features correspond with which emotions; it is trained on a labelled dataset. The features are derived by applying a bank of Gabor filters to a set of frames, determining the Local Binary Pattern (LBP) of each resulting pixel, and then averaging the results over three orthogonal planes (TOP), as outlined in [1]. CrowdEmotion challenged the study group to improve the accuracy, processing speed and cost-efficiency of the tool. In particular they wanted to know if a subset of the bank of Gabor filters was sufficient, and whether the image filtering stage could be implemented on a GPU. A framework for choosing the optimum set of Gabor filters was established, and preliminary testing performed. Different ways of implementing Gabor filters were explored. Some elements of the feature set give little information, thus ways of reducing the dimensionality of this were interrogated. Some steps in the procedure outlined in [1] seemed ad-hoc, in particular when taking a subset of LBPs and choosing a gridding pattern to perform the TOP step. Taking a subset of LBPs was found to be fully justified. Meanwhile choosing a gridding pattern is open to interpretation; we make some suggestions on how this choice might be improved. A short review of alternatives to using a SVM as a classifier is presented.

Version 1.0
May 18, 2014
iii+23 pages

Report author

Erhan Coskun (Karadeniz Technical University)
Torran Elson (Smith Institute)
Sean Lim (University of Oxford)
James Mathews (University of Cambridge)
Gruff Morris (Lancaster University)
Nikolai Nowaczyk (Universität Regensburg)
RafałProńko (Centre for Industrial Applications of Mathematics and Systems Engineering)
Patrick Nima Raanes (Univeristy of Oxford)
David Kofoed Wind (Technical University of Denmark)

Contributors

Fraser Campbell (CrowdEmotion)
Paul Dellar (University of Oxford)
Daniel Jabry (CrowdEmotion)
Imad Mudi El-Ddin (Prince Mohammed Faisal University)
Mathew Woolway (University of the Witwatersrand)

ESGI100 was jointly hosted by
Smith Institute for Industrial Mathematics and System Engineering
and the University of Oxford

with additional financial support from
Engineering and Physical Sciences Research Council
European Journal of Applied Mathematics
Oxford Centre for Collaborative Applied Mathematics
Warwick Complexity Centre

Contents

1 Introduction

1.1 Background

(1.1) The science of facial expression coding dates back to the 1970s and the research of psychologist Paul Ekman, who established the first taxonomy of universal human facial expressions and their corresponding emotions. In recent years, advances in computer vision have enabled the automation of facial expression coding, which has opened up new application areas for research and commercial purposes. CrowdEmotion, a technology company based in London, have developed a cloud-based platform for automated facial expression coding. The system processes videos of human faces and labels sequences of frames according to the detected facial action units (muscle contractions) and corresponding emotional states. Action unit and emotion libraries are trained on sets of labelled face videos, enabling ongoing refinement of the tool.

(1.2) CrowdEmotion use a technique based on Local Gabor Binary Patterns from Three Orthogonal Planes (LGBPTOP) [1] to process video frames. In brief, facial features and actions (features in time) cause local appearance changes over time and dynamic texture descriptors are used for detection.

1.2 Problem Statement

(1.3) Numerous opportunities exist to improve the current implementation of the procedure that CrowdEmotion use. Immediate challenges are:

1. LGBPTOP has shown strong performance compared with other methods of similar computational cost [1]. Certain aspects of the method are quite ad-hoc, however, indicating that there is room for improvement. Therefore a sound understanding of the method should be attained in order to suggest further areas of accuracy improvement.
2. Gabor Filter Selection: Identifying which Gabor filters perform better than others and to what degree. This would allow CrowdEmotion to eliminate filters with a low contribution to the overall accuracy and therefore improve performance. A key aim would be to enable smart feature selection by identifying an optimal subset of Gabor filters.
   Requirements for study:
   - Annotated data: In order to perform a proper study on Gabor filters (DISFA [2] and GEMEP-FERA databases [3]).
   - Machine learning accuracy: Alternative Force Choice (area under ROC curve approximation) has previously been used to estimate accuracy therefore in order to make a valid comparison it is likely this method should be used as part of the study. This methodology is discussed further in [1].
   - A relevant tutorial on Gabor wavelets is given in [4].

3. Parallel Computing: Design an algorithm intended to be executed on a massively parallelised computing device (i.e. GPU) based on the Gabor filters theory given in [1]. The aim in this case would be to achieve the maximum processing speed increase in comparison to the existing CPU-based implementation.
   Requirements for study:
   - Current Implementation: Access to the source code repository will be provided
   - Programming Tools: C++, NVIDIA CUDA toolkit

1.3 Objectives

(1.4) As discussed above, there are opportunities to make processing more efficient lending three benefits to the final product:

1. Accuracy: Improvement of the algorithm to include selection of appropriate predictive features.
2. End-User Processing Speed: Any improvement in turn-around time is attractive to end-users; while the ability to process in realtime will open up opportunities for new applications capable of measuring and responding to user expressions.
3. Processing / Cost Efficiency: Deploying this solution at scale as a cloud-based service will involve processing extensive hours of video for large numbers of users. Increasing the computational efficiency of this will therefore reduce costs associated with rented computational capacity. Significant efficiency improvements may also enable processing on mobile hardware.

1.4 Overview

(1.5) As described above, the system employed by CrowdEmotion closely follows [1]. Below we will briefly outline, at a high level, the process followed by CrowdEmotion, dip into some of the detail to prepare the way for later sections and then outline the approaches made to the problem. An overview of the process is shown in Figure 1.

(1.6) Facial movements may be classified using the Facial Action Coding System (FACS). This was originally developed by Swedish anatomist Hjortsjö [5] and refined by Ekman, Friesen, and Hager [6]. The latter work details how basic combinations of facial movements (called Action Units, AUs) can be interpreted as corresponding to certain emotional states. For instance, a combination of AU6 (cheek raiser) and AU12 (lip corner puller), would indicate happiness. Hence if AUs can be accurately identified by CrowdEmotion's classifier, the emotional state of the individual can be inferred.

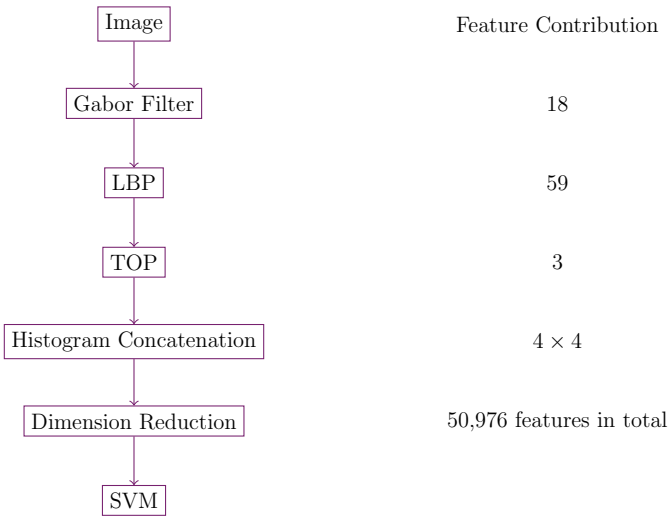

Figure 1: Overview of the steps taken in CrowdEmotion's classification scheme. The number of features is tallied up on the right-hand side.

(1.7) AUs are the activation or relaxation of facial muscles. On an image these can be recognized as edges. CrowdEmotion's process aims to take a set of images of a subject, identify the edges in the pictures and then classify the edge information.

(1.8) To do this, CrowdEmotion take a video and split it into batches of five frames. When analysing a whole video offline, one batch and the next overlap by four frames; when the software is used in real-time the next batch analysed is taken once the first batch has been processed. In real-time mode, the sequences of five frames are selected from the webcam feed when the previous set have finished processing and therefore many sequences are skipped unless the process is carried out on sufficiently powerful hardware.

(1.9) A bank of 18 Gabor filters is applied to each frame of the video. Gabor filters tend to be effective at identifying edges and characterising textures. They consist of a Gaussian envelope and a sinusoid; the type of filter applied by CrowdEmotion takes the following form:

$$\begin{aligned} G(x,y) &= \exp\left(\pi\sigma^2((x-x_0)_r^2+(y-y_0)_r^2)\right) \\ &\quad \cdot \exp\left(2\pi\imath\phi((x-x_0)_r+(y-y_0)_r)+P\right) \end{aligned} \tag{1}$$

where

$$\begin{aligned} (x-x_0)_r &= (x-x_0)\cos\theta + (y-y_0)\sin\theta \\ (y-y_0)_r &= -(x-x_0)\sin\theta + (y-y_0)\cos\theta \end{aligned}$$

(1.10) x_0 and y_0 are some coordinate points, θ is a spacial angle, and ϕ is a spacial frequency. P is a phase term. Since we are interested in the absolute

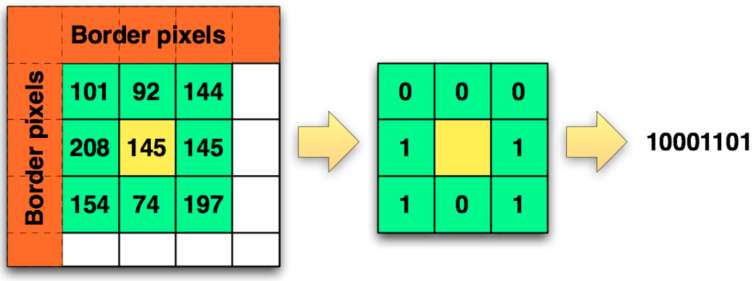

Figure 2: How we compute the LBP patterns. Picture from [1]

values of the output of the filter, the phase term can be disregarded. Note that the formulation in (1) differs from that in [1]. We think this is a transcription error in the paper. The 18 Gabor filters are chosen in [1] by taking combinations of six spacial angles and three spacial frequencies:

$$\begin{aligned} \phi &\in \left\{\tfrac{\pi}{2}, \tfrac{\pi}{4}, \tfrac{\pi}{8}\right\} && \text{Frequencies} \\ \theta &\in \left\{\tfrac{k\pi}{6} \mid k = 0, \ldots, 5\right\} && \text{Angles} \end{aligned}$$

(1.11) Sections 2.1 and 2.2 explore whether all 18 Gabor filters are necessary and suggests methods to determine an optimum subset. Section 2.3 gives a detailed mathematical description of how Gabor filters are implemented.

(1.12) Applying the Gabor filter mathematically amounts to taking the convolution of an image (function) and the filter (a Gabor impulse response). Since we are dealing with digital images this process is performed discretely. How this convolution might be better implemented is dealt with in Section 2.4.

(1.13) Each frame is convolved with the 18 Gabor filters. This leads to 18 Gabor Pictures (GPs) arising. A Local Binary Pattern (LBP) is determined for every pixel in each picture. A LBP is determined by comparing the (grayscale) value of the pixel with its eight neighbours. If the (grayscale) value of its neighbour is equal to or larger than the pixel's value it is labelled with a 1, otherwise it is labelled with a 0. Starting from the pixel to the left, these values are concatenated into an eight digit binary number. This is the LBP for that pixel. The process is shown in Figure 2. There are $2^8 = 256$ possible combinations. However it turns out that only 59 of them are significant. Why only 59 of them are significant is dealt with in Section 2.6. Overall this gives us the Local Gabor Binary Patterns (LGBP).

(1.14) The next step in the process is to take Three Orthogonal Planes (TOP) through the data. A batch of five frames, where the same Gabor filter has been applied to each, is taken. The three orthogonal planes are the xy, xt and yt slices. Currently these planes are split into 4×4 grids. For each

grid cell the occurance of each LBP is counted and then averaged over the third dimension (the one not in the slice). So for the xy slice, the count of LBPs in each element of the 4×4 grid is taken per frame. Then the count is averaged over the five frames. Sections 2.7 and 2.8 explore why a 4×4 grid is taken and discusses possible improvements that might be made to this choice. Combining the LGBP and TOP steps gives the method its name LGBP-TOP (as outlined in [1]).

(1.15) For each element of the 4×4 grid, a histogram is created where the bins are the 59 LBPs of interest and the height is determined by the average of the LBP count in that element. Since there are 18 GPs, each with 3 orthogonal planes containing 4×4 grids, there are $18 \times 3 \times 4 \times 4 = 864$ histograms created. These histograms are concatenated to a set of 50,976 features characterizing the five frame batch. Where 50,976 arises from there being 59 features per histogram i.e. $50{,}976 = 18 \times 3 \times 4 \times 4 \times 59$.

(1.16) This number of features is so large it is almost certain that using all these available dimensions in a model would lead to overfitting, particularly as the training data has fewer instances. By overfitting we mean fitting the excess features to noise in the training dataset. Hence a subset of features is chosen to train the data on. The process of choosing which features to use is outlined in [1]. In section 2.5, alternative ways of choosing these features are explored.

(1.17) The final part of the procedure is to train a Support Vector Machine (SVM) to recognize different action units (AUs) by finding correlations between action units and features. SVMs are a standard machine learning technique and are well understood, further detail can for instance be found in [7] and [8].

(1.18) To meet the objectives of the project, improving accuracy, increasing processing speed and assessing cost efficiency, a number of different approaches were taken. The most obvious step is to reduce the total number of features that are generated. This means, for instance, finding an optimum subset of Gabor filters to use. Reducing the total number of features would represent a cost efficiency, decrease the amount of processing required, and reduce the chance of over fitting the SVM to the training data and thereby increase accuracy. Sections 2.1 and 2.2 discuss a framework for picking an optimum subset of Gabor filters. Section 2.5 meanwhile investigates how the dimension reduction step is taken, how it might be improved and whether it can be used to determine which parameters are less important than others.

(1.19) As well as attempting to reduce the number of features, we assessed potential improvements to the methodology, paying particular attention to parts that seemed to be ad-hoc and in need of better understanding. The approaches taken in this direction are outlined in Section 2.3 where the struc-

ture of Gabor filters are explored, in Section 2.4 where the use of FFTs are investigated, in Section 2.6 where the methodology of using LBPs is investigated, and in Section 2.7 where the gridding methodology is considered. In addition the use of SVMs is discussed in Section 2.9.

2 Approaches to the Problem

2.1 Choosing the Optimum Set of Gabor Filters

(2.1) Gabor filters depend on various parameters, in particular a frequency ϕ and an orientation θ. In accordance with [1], one set of good parameters is

$$\phi \in \left\{\frac{\pi}{2}, \frac{\pi}{4}, \frac{\pi}{8}\right\} \qquad \text{Frequencies}$$
$$\theta \in \left\{\frac{k\pi}{6} \mid k = 0, \ldots, 5\right\} \qquad \text{Angles}$$

(2.2) This gives $3 \times 6 = 18$ possible combinations leading to 18 different Gabor filters. In the current implementation by CrowdEmotion of the algorithm LGBP-TOP [1] all 18 filters are applied and the resulting features (after they have been histogrammed) are concatenated. One of the main questions posed by CrowdEmotion was whether all 18 filters were necessary for obtaining a good accuracy when classifying action units. A reduction in the number of Gabor-filters used would lead to a direct reduction in the processing time for each image.

(2.3) We investigated how many of the given Gabor-filters were needed to obtain good classification rates. This was done by getting the main software developer from CrowdEmotion to alter the source code such that we could enable the filters one by one (unfortunately only one filter at the time). We were then able to obtain accuracy measures for the different filters individually, these are given in Table 1. By looking in depth at the numbers presented in the table (for example by computing the median performance for each angle or frequency), one can see that for AU1, a frequency of 22.5 seems to perform significantly better and that for AU27, an angle of 60 degrees seems to perform significantly better. It should be noted here that the accuracies obtained are not useful by themselves, but the fact that some filter-parametrizations outperform others is an indication that the choice of filters should be investigated further, and that it is very likely that a subset of the orignal 18 filters (or a smaller number of new filters) could lead to the same amount of accuracy while reducing the processing time.

(2.4) When the number of Gabor-filters is specified, one needs to find a set of optimal parameters for the filters. According to [9], a good approach to selecting optimal parameters for Gabor-filters is to "sample uniformly one of the parameters and perform a 2D search in the remaining dimensions".

Angle	Frequency	AU27 1	AU27 2	AU27 3	AU1 1	AU1 2	AU1 3
0	90	0.812087	0.790414	0.830341	0.488713	0.531759	0.5301
30	90	0.854423	0.877927	0.896334	0.483849	0.50977	0.45356
60	90	0.896141	0.86656	0.90626	0.474587	0.467798	0.51598
90	90	0.856489	0.859606	0.84175	0.494785	0.394204	0.50856
120	90	0.862764	0.85459	0.854361	0.47241	0.459189	0.43952
150	90	0.827946	0.857035	0.857391	0.488301	0.488352	0.5538
0	45	0.7866	0.795169	0.843658	0.462362	0.50294	0.44445
30	45	0.877915	0.875895	0.874561	0.532812	0.469046	0.47031
60	45	0.899448	0.914527	0.889571	0.528018	0.510238	0.51682
90	45	0.901605	0.824795	0.876945	0.459712	0.516778	0.46747
120	45	0.84645	0.866322	0.843059	0.503215	0.496067	0.48502
150	45	0.859431	0.849256	0.848155	0.475401	0.47505	0.55442
0	22.5	0.783895	0.689591	0.793548	0.497885	0.5061	0.48933
30	22.5	0.820737	0.804938	0.754799	0.596578	0.693722	0.60911
60	22.5	0.866132	0.88609	0.854865	0.755597	0.573194	0.57738
90	22.5	0.808892	0.862207	0.806295	0.516895	0.547131	0.52234
120	22.5	0.814439	0.859716	0.850162	0.510153	0.543315	0.57855
150	22.5	0.752165	0.737825	0.806632	0.640927	0.556124	0.58658

Table 1: Each of the 18 rows corresponds to a filter with parameters given in the first two columns. The next three columns (AU27 1, AU27 2 and AU27 3) are accuracy results for AU27 and the next three are results for AU1. The three columns for each action unit corresponds to three runs of the code (the results are not deterministic because of a random training/test split). The accuracy measurements are 2AFC-scores where a score of ± 1.0 is optimal and 0.5 is random.

Alternatively one could use a heuristic search (e.g. genetic algorithms) to search more intelligently for an optimal set of parameters.

2.2 Optimisation Formalism

(2.5) Framing decision problems as optimisation problems is a useful approach. One benefit of this is that the problem at hand is immediatly and accurately specified. Furthermore, it also makes available the vast number of highly developed optimisation algorithms from the literature. Several optimisation ideas were suggested, some of which were:

1. Frame the problem as a continuous optimisation problem. Suppose the information from the different Gabor filters can be merged using weights (which add up to one, or some arbitrary constant). One way of doing this is to average over the histograms from different Gabor filters. One can then specify an objective/cost function in terms of those weights and the filters, and use (gradient-less) continuous optimisation algorithms to find an optimum set of weights. It would then also seem justified to fully turn off (set the weights to zero) the filters with small weights. After this, the optimisation should be run again.
2. Genetic algorithms lend themselves to discrete optimisation and as such they are a good alternative to finding the optimal Gabor filter selection.
3. Manual tuning. This is related to section 2.1, and should be aided by investigations of covariances between the results from different Gabor filters.

2.3 Mathematical Formulation of Gabor Filters

(2.6) Gabor filters are bandpass filters commonly used in edge detection. The impulse response for a Gabor filter is given by the product of a Gaussian envelope (which plays the role of a bandpass filter) and a complex sinusoid (which acts as the kernel in a Fourier transform), i.e.

$$g(x, y; x_0, y_0, \sigma, \theta, \phi, P) = G(x, y; x_0, y_0, \sigma, \theta) S(x, y; x_0, y_0, \phi, P),$$

where the Gaussian envelope G is given by

$$G(x, y; x_0, y_0, \sigma, \theta) = \exp\left(-\pi\sigma^2((x - x_0)_r^2 + (y - y_0)_r^2)\right),$$

and the complex sinusoid S is given by

$$S(x, y; x_0, y_0, \phi) = \exp\left(2\pi i\phi((x - x_0)_r + (y - y_0)_r) + P\right).$$

The coordinates $(x - x_0)_r$ and $(y - y_0)_r$ are rotated coordinates, defined by

$$\begin{aligned}(x - x_0)_r &= (x - x_0)\cos\theta + (y - y_0)\sin\theta,\\ (y - y_0)_r &= -(x - x_0)\sin\theta + (y - y_0)\cos\theta.\end{aligned}$$

(2.7) We may also write the impulse response in matrix-vector form. Let $u = (x, y)^T$ and $u_0 = (x_0, y_0)^T$, so that u and u_0 are column vectors. Define the matrix R to be

$$R = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix},$$

Then, the impulse response for a Gabor filter is given by

$$g(u; u_0, \sigma, \theta, \phi, P) = \exp(-\pi\sigma^2(u - u_0)^T R^T R(u - u_0)) \exp(2\pi i\phi u^T \mathbb{1} + P),$$

where $\mathbb{1}$ is a column vector of ones. Note here that $R^T R = I$, where I is the identity matrix. Therefore, the impulse response function is now

$$g(u; u_0, \sigma, \phi, P) = \exp(-\pi\sigma^2(u - u_0)^T (u - u_0)) \exp(2\pi i\phi u^T \mathbb{1} + P),$$

i.e. the function g is independent of θ. In this case, the Gaussian envelope is independent of the rotation angle θ.

(2.8) Let $f(u)$ be a function. Then, the Gabor filter of f, denoted by $\mathcal{F}f$, is defined to be the convolution of f and the impulse response, i.e. we have

$$\mathcal{F}f(u) = \int g(\hat{u}; u_0, \sigma, \phi, P) f(u - \hat{u})\, d\hat{u} := (g * f)(u). \tag{2}$$

(2.9) From (2), we find that the Gabor filter is linear, since for every scalar $\alpha, \beta \in \mathbb{R}$ and two functions f_1 and f_2, we have

$$\mathcal{F}(\alpha f_1 + \beta f_2) = g * (\alpha f_1 + \beta f_2) = \alpha(g * f_1) + \beta(g * f_2) = \alpha\mathcal{F}f_1 + \beta\mathcal{F}f_2.$$

Furthermore, if $\mathcal{F}_1$ and $\mathcal{F}_2$ are two filter operators with impulse responses g_1 and g_2 respectively, then

$$\mathcal{F}_2[\mathcal{F}_1[f]] = g_2 * (g_1 * f) = g_1 * (g_2 * f) = \mathcal{F}_1[\mathcal{F}_2[f]],$$

i.e. the filtering process commutes, the order at which the filtering processes are applied does not matter. This follows from the commutative and associative properties of a convolution.

2.4 Using Fast Fourier Transforms

(2.10) As discussed in Sections 1.4 and 2.3, in a spatial coordinate sytem with coordinates (x, y), a Gabor filter is defined as the product of a Gaussian envelope and a sinosoidal carrier. In (3), a slightly different formulation than (1) for such a filter is given. The main difference is that there is a scale factor given to each direction, a normalization factor, and the spatial frequency is allowed to differ in the x and y directions.

$$G(x,y) = \frac{1}{2\pi\sigma_x\sigma_y}\exp\left(-\frac{1}{2}\left(\left(\frac{(x-x_0)_r}{\sigma_x}\right)^2+\left(\frac{(y-y_0)_r}{\sigma_y}\right)^2\right)\right) \cdot\exp\left(2\pi\imath\left(\frac{\mu_x(x-x_0)_r}{\sigma_x}+\frac{\mu_y(y-y_0)_r}{\sigma_y}\right)\right) \quad (3)$$

where $[(x-x_0)_r,(y-y_0)_r]^T$ is the vector obtained by the rotation of $[x-x_0,y-y_0]^T$ by an angle of θ in the clockwise direction, i.e.

$$\begin{bmatrix}(x-x_0)_r\\(y-y_0)_r\end{bmatrix}=\begin{bmatrix}\cos\theta & \sin\theta\\-\sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}x-x_0\\y-y_0\end{bmatrix},$$

(x_0,y_0) is the center of the filter; (σ_x,σ_y) determines the bandwidth; (μ_x,μ_y) is the frequency the sinusoidal component of the filter in the x and y directions respectively; θ is a parameter that determines the orientation of the filter, i.e. the orientation of the Gaussian, and $(\mu_x/\sigma_x,\mu_y/\sigma_y)$ determines the orientation of sinusoid within the filter. The form of the filter presented here differs slightly from the two-dimensional version proposed in [10] in that $((x-x_0)_r,(y-y_0)_r)$ is scaled with (σ_x,σ_y) both in envelope and carrier for a consistent scaling.

(2.11) For the parameters $\sigma_x=1$, $\sigma_y=2$, $\mu_x=\pi/8$, $\mu_y=\pi/4$, the real part of Gabor filters are displayed in Figure 3 over the region $[-4,4]\times[-4,4]$ corresponding to $\theta=[0,-\pi/4,-\pi/2,-3\pi/4]$, from left to right and top to bottom respectively.

(2.12) Notice the orientation of the Gaussian and sinusoid components of the filter in Figure 3 as θ values increases.

(2.13) In this report we focus on the use of Gabor filters in identifying the discontinuities in intensity of an image, a technique better known as edge detection. The images we are interested in are of the human face, as this is the focus of the study group problem.

(2.14) There are many different approaches to edge detection. We refer to a survey [11] for commonly used edge detection methods; Gabor filters belong to the family of Gaussian based methods. Different methods tackle the problem with some common issues such as reduction of noise and isolation of false edges that may arise during the edge detection precedure.

(2.15) Applying a filter to a signal is represented mathematically by convolving the signal and filter functions. Refer to [12] for an introduction to the topic. For a two-dimensional image I and a filter G, recall that the *linear convolution*

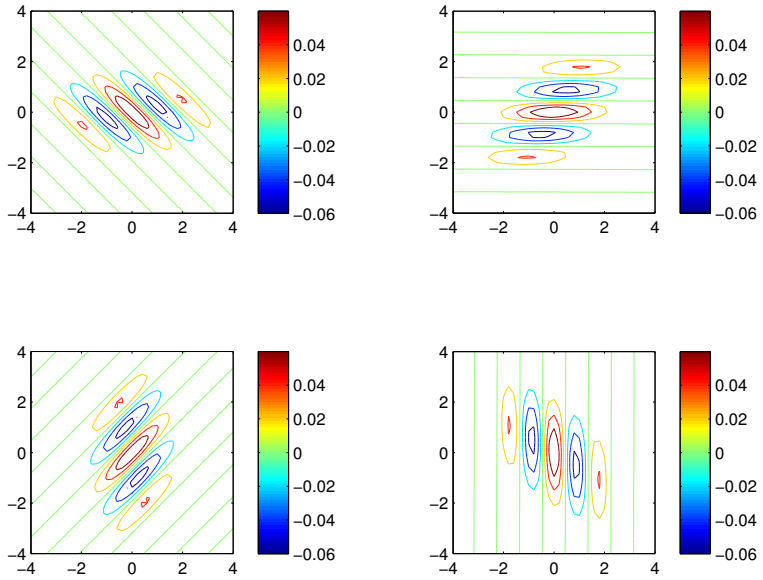

Figure 3: Real part of the Gabor filter corresponding to $\theta = [0, -\pi/4, -\pi/2, -3\pi/4]$ values from left to right and top to bottom respectively.

is defined as

$$[I * G][m, n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i, j)G(m - j, n - j) \tag{4}$$

(2.16) From (4) it is apparent that the convolution procedure modifies the image pixel values, indexed by the pair (m, n), by using a linear combination of neighboring pixel values. This combination is formed through the filter values, i.e. every pixel is weighted by the value of the filter at that point and the weighted values are added up to determine the new pixel value. The size of the contributing neighbours are determined by the size of filter and

$$\text{Filtered Image} = \text{Image} * \text{Filter}$$

where '$*$' represents the two dimensional convolution. In image filtering, the summation indices in (4) run over finite intervals. If I is of size $N \times N$ and G is of size $M \times M$, then the linear convolution (4) is of size $(N + M - 1) \times (N + M - 1)$.

(2.17) For a filter of size $M \times M$, modification of a single pixel value requires $O(M^2)$ operations (additions and multiplications). For an image of size $N \times N$, the complexity of convolution is around $O(M^2N^2)$ which gives a heavy burden on computational resources as the size of the image and the filter become 'large'. Furthermore, in the context of edge detection, filtering has to be perfomed more than once with different sets of parameters to

identify particular edges, so it is important that the convolution procedure is performed in the most efficient way.

(2.18) One way of reducing the complexity is to perform a Fast Fourier Transform (FFT) convolution using the discrete version of the well-known convolution theorem for circular convolution:

$$\begin{aligned} \text{FFT(Filtered Image)} &= \text{FFT(Image} * \text{Filter)} \\ &= \text{FFT(Image)} \times \text{FFT(Filter)} \end{aligned} \tag{5}$$

Applying the inverse FFT, say FFT^{-1}, to each side, we have

$$\begin{aligned} \text{Filtered Image} &= \text{Image} * \text{Filter} \\ &= FFT^{-1}(FFT(\text{Image}) \times FFT(\text{Filter})) \end{aligned}$$

(2.19) This procedure leads to a convolved image without performing a convolution at all. Here, the product '$\times$' in (5) means pointwise multiplication of complex numbers in the frequency domain. The complexity of this procedure for an image of size $N \times N$ is about $O(N^2 \log_2(N^2))$, which is a considerable reduction compared to $O(M^2N^2)$ for the traditional convolution and it is independent of the filter size.

(2.20) However, instead of circular convolution which may distort edges, we would like to compute the linear convolution, yet taking advantage of a Fast Fourier Transform. To do so, one approach is to use so-called zero-padding [13] where both the filter with size $N_1 \times N_2$ and image with size $M_1 \times M_2$ are padded with zeros, resulting in an extended image, Image_e and the extended filter, Filter_e, of the same size $(2^n, 2^m) >= (M_1 + N_1 - 1) \times (M_2 + N_2 - 1)$, namely,

$$\text{Image\_e}(i,j) = \begin{cases} \text{Image}(i,j) & (i,j) \in N \times N \\ 0 & \text{otherwise} \end{cases}$$

and

$$\text{Filter\_e}(i,j) = \begin{cases} \text{Filter}(i,j) & (i,j) \in M \times M \\ 0 & \text{otherwise} \end{cases}$$

(2.21) We require the extended image and filter to be of the size $(2^n, 2^m)$ so that we can take advantage of the speed of FFT's.

(2.22) One can then compute $\text{FFT}^{-1}(\text{FFT(Image\_e)} \times \text{FFT(Filter\_e)})$ which becomes the linear convolution Image$*$Filter, without having to compute the convolution itself. However, extra zeros resulting from zero padding have to be isolated appropriately. A MATLAB code implementing convolution using the FFT convolution procedure outlined above is given in Appendix A.1.

Image Size $(N \times N)$ $N = 100$	Filter Size $(M \times M)$	CPUTime(of 100 runs) (Traditional Convolution)	CPUTime(of 100 runs) (FFT Convolution)
	$M = 10$	0.4219	0.5313
	20	1.0156	0.5313
	30	1.9375	2.4219
	40	3.4531	2.4219
	50	4.9688	2.4375
	100	17.1406	2.4375

Table 2: Trial runs of convolutions implementing using a traditional and FFT convolution, $N = 100$.

Image Size $(N \times N)$ $N = 200$	Filter Size $(M \times M)$	CPUTime(of 100 runs) (Traditional Convolution)	CPUTime(of 100 runs) (FFT Convolution)
	$M = 10$	2.5156	2.5000
	20	4.1250	2.4375
	30	7.7813	2.4531
	40	13.7031	2.5000
	50	20.0156	2.5000
	100	68.7344	15.6563

Table 3: Trial run of convolutions implementing using a traditional and FFT convolution, $N = 200$.

(2.23) Table 2 and Table 3 show typical results from traditional convolution and FFT convolutions from random image and filters produced within the MATLAB environment. The indicated CPU times correspond to those of one hundred runs.

(2.24) The results for both image sizes indicate that FFT convolution is much faster than the traditional convolution in almost all the image and filter sizes. However, zeropadding upto power of two, as done here, is crucial, otherwise FFT convolution may not perform better.

(2.25) In Figures 4 and 5 we display contour plot of Gabor filters of computational size $[9 \times 9]$ and an FFT convolved imaged with parameters $\sigma_x = 1$, $\sigma_y = 2$, $\mu_x = \pi/8$, $\mu_y = 0$ and orientations $\theta = 0, -\pi/4$ and $\theta = -\pi/2, -3\pi/4$, respectively.

(2.26) In Figure 6, we display the contour plots of the superposition of the edges in Figures 4 and 5, which is a relatively good description of the corresponding image.

(2.27) Another approach for an efficient implementation of Gabor filters is the separation of the filter into product of one dimensional filters

$$G = G1 \times G2$$

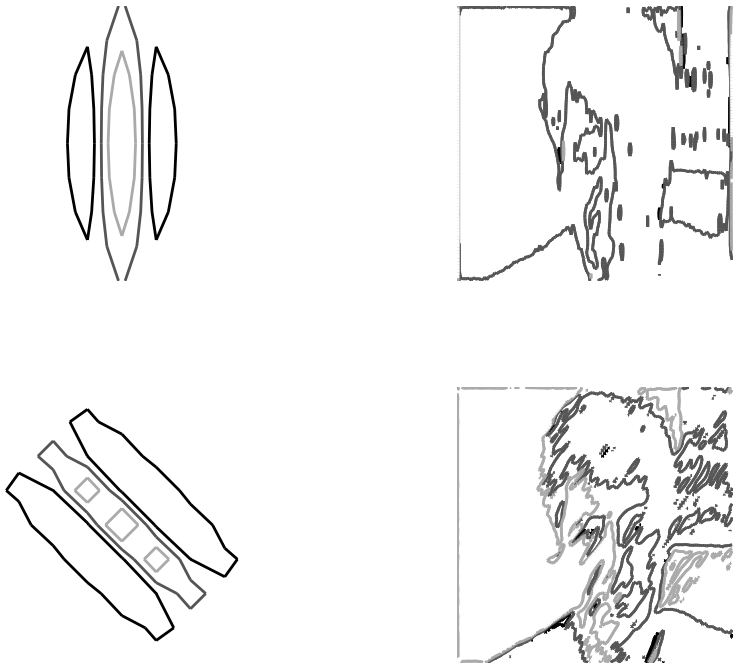

Figure 4: Gabor Filter and edge detection with $\theta = 0$ (top), $\theta = -\pi/4$ (bottom)

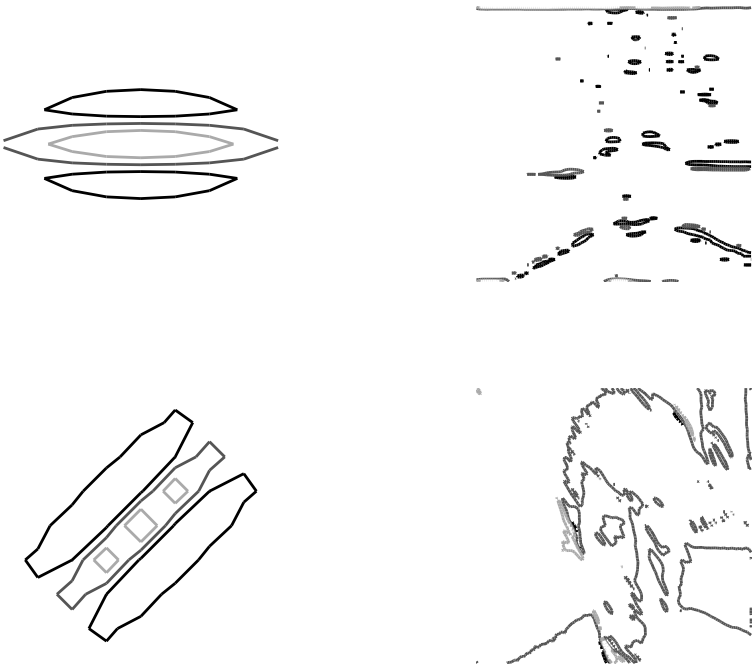

Figure 5: Gabor Filter and edge detection with $\theta = -\pi/2$ (top), $\theta = -3\pi/4$ (bottom).

Figure 6: Superposion of edges detected in Figure 4 and Figure 5

and then perform two one dimensional filters, instead of one two dimensional filter, see the relevant literature for details.

2.5 Dimension Reduction

(2.28) This work focussed on reducing the number of features after generating them from the LGBP. Given that others were working further up the processing chain, one of the core requirements was to ensure that the dimension reduction technique worked regardless of how many features were passed down from the upstream process. The purpose of reducing the number of features is to help improve the classifier performance. A classifier trained with more features than samples is not going to be a true representation of the entire range of possibilities; this can lead to overfitting (the classifier will find it difficult to classify previously unseen samples due to the huge variability created by having so many features - it will have fitted noise in the training data to excess features).

(2.29) The work was conducted in two stages:

1. Reduce the number of features using a repeatable and robust method.
2. Re-train the classifier for an action unit based on the reduced feature set.

(2.30) In order to reduce the number of features, we had to understand the char-

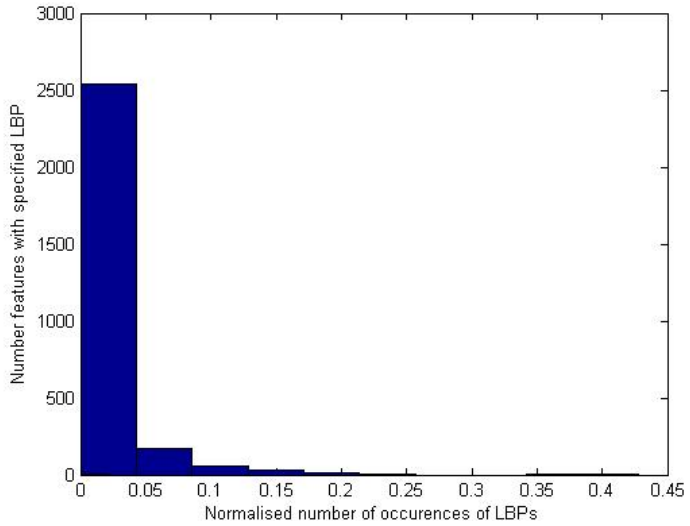

Figure 7: Gaussian normed graph of features from one Gabor filter.

acteristics of the features themselves. Thus we constructed a histogram of how many times each feature has a non-zero value i.e. exist in the data sample space (we used about 500 samples to conduct the analysis). Figure 7 shows a Gaussian normed graph of the features for one filter, and it is possible to see that the majority of the features occur 0 or very few times (up to 0.1).

(2.31) There are very few features that occur often (> 0.1). In this case < 0.1 was selected as a threshold for ignoring the features (a slightly arbitrary value, but based on the graph, most of the features that do not occur very often are < 0.1). By removing these features it was possible to reduce the feature set for Gabor filter one to 76 down from 2,832. This is a large reduction that is repeatable. One could argue that the features that occur rarely or not at all add more to the information than features that occur often and it is possible that this is true. However when conducting classification, the features with very low or 0 values will be given a lower weighting than features with a large contribution. The following formula, using Lagrange multipliers (α), determines the hyperplane (defined by $\mathbf{w}$ and b) used in the SVM. Hence, by considering this formula, the effect of feature frequency on classification can be determined:

$$\arg\min_{\mathbf{w},b}\max_{\alpha\geq 0}\left(\frac{1}{2}\|\mathbf{w}\|^2-\sum_{i=1}^{n}\alpha_i\big(y_i(\mathbf{w}\cdot\mathbf{x_i}-b)-1\big)\right) \tag{6}$$

(2.32) $\mathbf{x_i}$ represents the features for the sample where $\mathbf{x_i}\in\mathbb{R}^n$, y_i represent the classification of datapoint i. Thus a feature that occurs infrequently will contribute far less to the classifier. The small rare occurring features may

contain most variance and thus most of the information, but this is not considered in the SVM classifier when there are several thousand features used to train it.

(2.33) We were able to reduce the total number of features to 442 for all filters, significantly less than 50,976. This should also contribute to making the classifier more accurate as it will reduce the chance of misclassification.

(2.34) The second stage was to train the classifier with the reduced feature set and test its performance. Initially a 60/40 split of the available sample data was used to train and test the classifier respectively. For AU1 (the only one tested during the study group) of 2,000 samples, only 34 were activated (1s) and the remainder were deactivated (0s). By using this large disparity in training samples (21 true, 1,179 false), the classifier is heavily biased towards the false class, meaning that it is very difficult to classify any true classes correctly. To correct this bias, a five pass cross-verification scheme was used, where a random data set consisting of 500 false samples and 21 true samples were used to train on each pass, with a different test set of random data consisting of 300 false and 13 true points. By using this scheme and correcting the bias of the available true / false samples the classifier attained a 98% accuracy for this action unit using the reduced feature set.

2.6 Local Binary Patterns

(2.35) In [1] it is stated that only 59 of the possible 256 LBPs are of significance. We wanted to understand why this subset of possible patterns contain the most important information. The 256 possible LBP are shown in Figure 8.

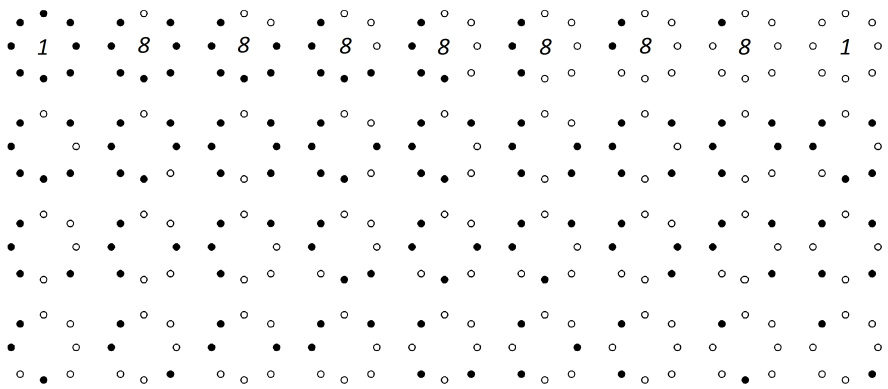

Figure 8: The 256 patterns represented as 36 different patterns up to rotational symmetry. Black and white dots represent ones and zeros. Picture from [14]

(2.36) In the literature the first row from Figure 8 are generally referred to as "uniform" patterns. These each have one block of 1's and one of 0's, or

have all 1's or all 0's. If there is more than one type of block there are eight patterns corresponding with the eight possible rotations. Hence there are $7\times8+1+1 = 58$ different "uniform" patterns. "Non-uniform" patterns have more than one block of 1's and 0's. "Uniform" patterns can be interpretted as being more likely to correspond to edges; one block of 0's and one of 1's indicate a strong transition from a light to dark area. While "non-uniform" patterns correspond to mixed areas, which are likely to not be edges. Since identifying edges helps locate the action units, the useful patterns to identify are those that correspond to edges. Moreover in [14] experimental data has shown that in various texture images roughly 90% of all LBP patterns come from these "uniform" patterns. Further experimental data (see for example [15]) has shown that the same proportional of patterns are uniform in facial images.

(2.37) As a result, nearly all current implementations using LBP, including the one considered in this report, look for these 58 uniform patterns, which are shown in Figure 9, and then group all the other patterns into a 59th category.

(2.38) It would seem that 58 is really the minimum that we could take to produce useful results, but we could perhaps increase performance slightly by ignoring the 59th category. This is because it is questionable whether amalgamating 206 patterns into one category and counting how many of these we have is useful. Experimental data could be used to test this, and implementation should be relatively easy.

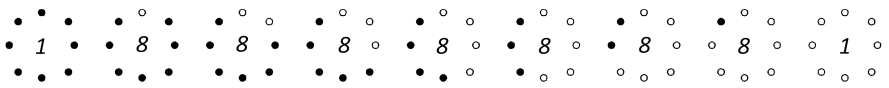

Figure 9: The 58 "uniform" patterns represented as 9 different patterns up to rotational symmetry. Black and white dots represent ones and zeros. Picture from [14]

2.7 Gridding

(2.39) A rectangular cutout of the region containing the face is obtained for each frame in the current implementation. This cutout is then partitioned by a 4x4 grid, providing some measure of locality. A relatively simple area for improvement could be to leave out a few of the grid cells that are peripheral to the face. Another possibility is non-rectangular cutouts and/or grid cells, although this may prove more complicated and/or costly due to higher complexity of such geometries.

(2.40) Currently we take a rectangle of cropped face, apply Gabor filters to it and then compute binary patterns for every pixel. We then split the image into

Figure 10: Applying the elliptic stencil to an image

a 4×4 grid, and see how many of each LBP pattern we have in each of the 16 regions.

(2.41) The reasoning for splitting the image into further regions is clear, since this gives us an opportunity to focus on just a couple of action units per region. However, the justification of why we choose a 4×4 grid, and not any other size is relatively unclear. In [17] it is noted that a 4×4 grid generally performs better than a 3×3 experimentally, but little other literature exists on the subject.

(2.42) If we decrease the number of regions then we decrease the number of features which will speed up the SVM step and also ensure we are not over-fitting the data.

(2.43) A final point to add (see [16]) is that the regions do not need to be the same size, they can overlap and they do not need to cover the whole image. It is this last part that inspires what follows.

2.8 Elliptic Stencil

(2.44) We note that the four corners of the rectangle are not going to be useful for emotion recognition, and indeed could even hinder it (since it could introduce edges outside the face). Our proposal is to create an elliptic stencil for the image, and then only perform LBP on the pixels within this elliptic stencil. This would mean we only perform the LBP on only $78\%\,(\pi/4)$ pixels, which represents considerable saving.

(2.45) Sample MATLAB code for this is detailed in the Appendix A.2. The code takes the image after applying the Gabor filters, and assigns a pixel value of -1 to all pixels outside of the ellipse stencil, while inside the pixel values are unchanged. We would then run the LBP scheme on all pixels with pixel values greater than zero. We would deal with the edges of the elliptic stencil in the same way as the edges of the rectangle before, which was not made clear but we presume the edges were ignored.

(2.46) We note that although we could in theory apply the elliptic stencil before the Gabor filters, this seems to change the results.

(2.47) The best way to split the elliptic stencil into further regions (such as the 4×4 grid) is an open question, which would require experimental testing. However, by applying the eliptical stencil we would hope you could use less than 16 regions.

2.9 SVM and Alternatives

(2.48) Support vector machines (SVM) are a very fast classification method once they have been trained. It is also very reliable, having been a standard method in Machine Learning for more than 20 years, and is fairly customisable, see [7] and [8] for further background. The advantage of an SVM lies in the training involving non-linear optimization, and the fact that the objective function is convex, so solving the optimization problem is direct. The number of prameters in the result ends up being smaller than the number of training points, but the number of these parameters is still quite large. An alternative approach is to set the number of parameters from above but allow them to be adaptive. This can be achieved through a feed-forward neural network [18]. For many applications the resulting model can be significantly more dense, and hence faster to evaluate than an SVM. The cost for this accuracy is that the likelihood function which forms the basis for training the neural network is no longer convex. In practice it is often worth investing extra computational resources during the training phase to obtain a denser model that is thus faster at processing new data.

(2.49) Methods also exist which attempt to obtain"intensity" measures of different emotions. This would be achieved by determining some sort of distance measure from the classification hyperplane in the SVM. According to the main developer, however, this has proven somewhat problematic. A straightforward alternative might be a logistic regression.

3 Conclusions and Recommendations

(3.1) The main focus of this report has been on how to reduce the feature set to improve the accuracy of classification by avoiding overfitting, and also decrease processing time as fewer calculation need to be implemented. In addition we interrogated the methodology set out in [1], finding that although some parts seemed ad-hoc on first reading, by and large the methodology was well justified.

(3.2) The preliminary results gained in Section 2.1 indicate that it is likely that a subset of the bank of 18 Gabor filters can be used to identify emotion states. Sections 2.2 gives three methods that might be used to determine

this subset: posing the problem as a continuous optimisation problem, using a genetic algorithm, and manual tuning. In all three approaches a significant amount of time would be required to determine the filters required to classify each AU. An intuitive argument can be given to understand why not all filters are necessary. Simply put, each Gabor filter is good at picking out edges at different orientations. This can be seen in Figures 4, 5, and 6, where edges in line with the filter are best identified. For a specific AU an edge with one particular orientation might be all that is required to classify it. For instance, smiling activates AU12 (lip corner puller); this may only require vertical edges to be identified to be correctly classified.

(3.3) Section 2.3 discussed mathematically how the Gabor filtering process is carried out, and 2.4 explored using FFT techniques to speed up calculations. The main result is that using the methodology in Section 2.4, padding images and filters with zeros, applying FFTs to the image and filter, performing the convolution in Fourier space (as a multiplication), and then taking the inverse FFT, would give a significant speed up compared with performing the convolution directly.

(3.4) A reduced set of features could be chosen using the method outlined in Section 2.5. Section 2.6 shows why taking 59 out of the possible 256 LBPs is a reasoned process to follow and not an ad-hoc procedure. Sections 2.7 and 2.8 suggest an alternative approach to gridding the image by discarding information that is not useful. The last section [7] discussed potentil alternatives to using an SVM as the classifier.

(3.5) As a final point the contribution of Paul Dellar should be mentioned. Over the week, he helped CrowdEmotion utilize the CUDA toolkit to diagnose memory allocation problems.

A Appendices

A.1 MATLAB Code for FFT

```
function Result=fftconv2(Image,Filter)
%Linear convolution with fft of size 2^n,
%Author:  Erhan Coskun, May, 2014.
[N1,N2]=size(Image);
[M1,M2]=size(Filter);
K1=N1+M1-1;              % Size of Linear Filter
K2=N2+M2-1;              %   "
n=pow_two_n(K1);     %smallest n such that 2^n >=K1
m=pow_two_n(K2);    %smallest m such that 2^m>=K2
two_to_n=2^n;
```

```
two_to_m=2^m;
Image_e=zeros(two_to_n,two_to_m);
Filter_e=zeros(two_to_n,two_to_m);
Filter_e(1:M1,1:M2)=Filter; % padded filter
Image_e(1:N1,1:N2)=Image;  % padded image
fftfilter=fft2(Filter_e);  % fft of padded filter
fftimage=fft2(Image_e);    % fft of padded image
Frprod=fftfilter.*fftimage; % pointwise multiplication of fft's
Result=ifft2(Frprod);       % Inverse fft
Result=Result(1:K1,1:K2);   %Isolate zero paddings
M1p=ceil((M1-1)/2);         % Indices for central part
M2p=ceil((M2-1)/2);         %   "
Result=Result((M1p+1):(N1+M1p),(M2p+1):(N2+M2p));
                            % central part of convolution
function n=pow_two_n(K);
test=1;n=0;
while test
 n=n+1;
test=2^n<K;
end
```

A.2 MATLAB Code for Eliptical Stencil

```
B=rgb2gray(imread('face2.jpg'));
s=size(B);
A=B+uint8(ones(s(1),s(2)));
c = fix(size(A) / 2);   %# Ellipse center point (y, x)
r_sq = [c(2), c(1)] .^ 2;  %# Ellipse radii squared (y-axis, x-axis)
[X, Y] = meshgrid(1:size(A, 2), 1:size(A, 1));
ellipse_mask = (r_sq(2) * (X - c(2)) .^ 2 + ...
    r_sq(1) * (Y - c(1)) .^ 2 <= prod(r_sq));
%# Apply the mask to the image
A_cropped = bsxfun(@times, A, uint8(ellipse_mask));
B_cropped=double(A_cropped)-ones(s(1),s(2)); %Image with -1's in cells we don't want
```

Figure 11: Matlab code for taking an image (in this case "face2") and creating an elliptic stencil

Bibliography

[1] Timur R. Almaev, Michel F. Valstar, *Local Gabor Binary Patterns from Three Orthogonal Planes for Automatic Facial Expression Recognition.* ACII '13 Proceedings of the 2013 Humaine Association Conference on Affective Computing and Intelligent Interaction, pp. 365-361, IEEE Computer Society Washington D.C., 2013

[2] http://www.engr.du.edu/mmahoor/DISFA.htm

[3] http://gemep-db.sspnet.eu

[4] http://mplab.ucsd.edu/tutorials/gabor.pdf

[5] C. Hjortsjö, *Man's Face and Mimic Language.* Studentlitteratur, 1969

[6] P. Ekman, W.V. Friesen, and J.C. Hager, *FACS Manual.* Network Information Research Corporation, May 2002

[7] Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." Machine learning 20.3 (1995): 273-297.

[8] C.M. Bishop, *Pattern Recognition and Machine Learning* Springer, 2006

[9] P. Moreno, A. Bernardino, J. Santos-Victor. *Gabor Parameter Selection For Local Feature Detection*, Proc. IBPRIA - 2nd Iberian Conference on Pattern Recognition and Image Analysis (Springer) , Estoril, Portugal, June 7-9, 2005.

[10] J. G. Daugman, *Complete Discrete 2-D Gabor Transforms by Neural Networks for Image Analysis and Compression* IEEE Transactions on Acoustics, Speech, and Signal Processing. Vol. 36. No. 7.JULY 1988

[11] M. A. Oskoei, and H. Hu, *A survey on Edge Detection Methods, Technical Report:CES-506*, School of Computer Science and Electronic Engineering, University of Essex, UK

[12] P. Prandoni, and M. Vetterli, *Signal Processing for Communications* EPFL Press, 2008

[13] J. G. Proakis and D.G. Manolakis, *Digital Signal Processing* Pearson Prentice Hall, 2007

[14] Timo Ojala, Matti Pietikainen, and Topi Maenpaa. *Multiresolution gray-scale and rotation invariant texture classification with local binary patterns.* Pattern Analysis and Machine Intelligence, IEEE Transactions on 24.7 (2002): 971-987.

[15] Timo Ahonen, Abdenour Hadid, and Matti Pietikainen. *Face recognition with local binary patterns.* Computer vision-eccv 2004. Springer Berlin Heidelberg, 2004. 469-481.

[16] http://www.scholarpedia.org/article/Local_Binary_Patterns

[17] Marko Heikkil, Matti Pietikinen, and Cordelia Schmid. *Description of interest regions with local binary patterns.* Pattern recognition 42.3 (2009): 425-436

[18] C.L. Fancourt, S. Haykin, S. Katagiri, J.T. Lo, J.C. Principe, I.W. Sandberg. *Nonliner Dynamical Systems: Feedforward Neural Network Perspectives* Wiley, 2001