Philipp Michel
Churchill College

Support Vector Machines in Automated Emotion Classification

CST Part II Project Dissertation

2003

Proforma

Name:	**Philipp Michel**
College:	**Churchill College**
Project Title:	**Support Vector Machines in Automated Emotion Classification**
Examination:	**Part II Computer Science Tripos, June 2003**
Word Count:	11,979[1]
Project Origin:	Discussion with Rana El Kaliouby
Supervisor:	Rana El Kaliouby

Original Aims of the Project

To apply Support Vector Machine learning in the context of emotion classification through facial expression by designing, implementing and evaluating an application to recognize emotions expressed in face images. The application should perform data extraction on a user-defined training set of face images grouped by emotion expressed. It should then use the data to train an SVM classifier and thus enable subsequent classification of unseen images. Finally, the results should be reported back to the user.

Work Completed

An image-based classification application satisfying all the aims above has been implemented. It allows for flexible evaluation of SVM classification of facial expressions for arbitrary user-defined emotion categories and various SVM settings.

An application to perform SVM-based recognition of spontaneous expressions in real time video, quoted speculatively as an extension in the proposal, was also implemented. Evaluation showed that it compares favorably to previous expression analysis systems in terms of accuracy and performance for a variety of interaction scenarios.

A poster resulting from project work will be presented at the 10th International Conference on Human-Computer Interaction in June 2003. A paper I co-authored (reproduced in Appendix A) has been submitted to the Fifth International Conference on Multimodal Interfaces in November 2003.

Special Difficulties

None.

[1]Word count automatically generated by running `detex` on the LaTeX source.

Declaration of Originality

I, Philipp Michel of Churchill College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed: ______________________________

Date: ______________________________

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides background information related to the project. The motivation behind the project is explained. A short overview of previous work is given. The conclusion describes the aims of the project.

1.1 Motivation

Charles Darwin was the first to recognize the superior expressive ability of the human face [10]. It provides one of the most powerful, versatile and natural ways of communicating our emotions and motivations [12]. Facial expressions play an important rôle during human social interaction, enabling us to provide communicative cues ascertaining our level of interest or signalling our desire to take a speaking turn. They also give our counterpart continuous feedback indicating that we have understood the information conveyed. Facial expression has been shown to account for 55% of the effect of a communicated message [25] and is hence the major modality in human communication.

Reeves & Nass [29] posit that we try to evoke our highly evolved social and affective skills when being confronted with unfamiliar technology capable of social interaction. This has given rise to the longstanding interest of the research community of enabling computer systems to make use of emotive information present in interaction. For the reasons above, inferring emotions from facial expressions in an automated manner has been the prime focus of research interest.

It turns out that while the human 'ceiling' in correctly classifying a given facial expression into a particular emotion category (e.g. anger) is very high (over 90%), computers have in the past performed significantly less well. It is only recently that a clearer understanding of the problem of facial expression analysis and advances in machine learning systems have enabled computers to exhibit comparable recognition rates.

Support Vector Machines (SVMs) are a machine learning system that is directly based on a branch of mathematics called statistical learning theory, which became properly formalized in the last decade. While SVMs have successfully been applied to a number of classification tasks and tend to often outperform classic neural network approaches, their application to facial expression analysis and emotion classification so far has been limited. The aim of this project is to apply SVMs to automated emotion classification.

1.2 Previous Work

Pantic & Rothkrantz [27] provide an extensive overview of the vast research area of facial expression analysis. Detail concerning the problems involved and various previous approaches is given in Chapter 2 and only some pointers to applications of Support Vector Machines in the context of expression analysis and at large is given here.

SVMs are widely used in pattern recognition. Osuna et al. [26] use SVMs to detect faces in images by considering the raw pixel information at various scales, while Vapnik and colleagues [22] perform handwritten digit recognition from scanned images of US Postal Service labels. Applications in Bioinformatics are numerous, for example protein homology detection [16] and gene expression data categorization from DNA microarrays [2]. Joachims [17] uses SVMs to perform text categorization.

Dumas [11] applies SVMs to expression recognition, comparing their performance to a neural network approach operating on the same data, presented by Dailey et al. [9], and reports high recognition rates. Littlewort et al. [23] use a similar technique to perform expression coding in preprocessed image sequences. Kapoor [20] employs SVMs as part of a larger framework for analyzing facial actions and head gestures.

1.3 Project Aims

1.3.1 Implementing an SVM-based Expression Analysis Application

The first general aim, as outlined in the proposal, was to integrate an SVM classifier into a graphical application in order to perform expression analysis from facial images and to evaluate the suitability of an SVM-based approach to emotion classification. It should allow the user to select suitable training examples for the classifier, process them to perform data extraction and use the application to classify unseen images into a range of emotion classes.

1.3.2 Investigating Different SVM Variants and Feature Extraction Methods

The second aim was to allow for experimentation with different flavors of SVM classification within the application in order to investigate the effectiveness of each for expression analysis from images. Furthermore, different ways of extracting data from the dataset of images were to be explored to decide how to provide numerical training data to the SVM classifier in the final application.

1.3.3 Implementing an SVM-based Real Time Video Expression Analysis Application (Originally an Extension)

The availability of a real time facial feature tracker (Eyematic Interfaces' `FaceTracker`) and performance results from the project work on image-based classification caused a major shift in project emphasis once the application described above had been implemented. It seemed that support vector machine classifiers were well suited for a video-based interactive classification application. What was originally a possible extension to the project became a core focus and took up considerable work time. The aim was to use the feature tracker to extract SVM training data from a live video stream and perform on-the-fly classification of spontaneous expressions using commodity hardware.

Chapter 2

Preparation

This chapter details the work done before coding was started. A requirements analysis for both the image-based and the video-based classification applications is presented. An overview of the problems involved in automated facial expression analysis is given. Support Vector Machines and their theoretical foundations are described. The approaches to facial feature extraction considered are detailed. Two different schemes for classifying facial expressions are given. The languages, tools and hardware used in the project are listed.

2.1 Requirements Analysis

2.1.1 Image-based Classification Application

1. Allow user to select images to be used for training
2. Perform feature extraction on the images to gather numerical SVM input
3. Allow labelling of training examples with the corresponding emotion classes (e.g. joy, sorrow, anger, etc.), defined by the user
4. Make SVM parameters (kernel function type, SVM algorithm used, etc.) user-controllable
5. Train an SVM instance
   (a) Gather training data from processed input images
   (b) Create an SVM model from the data
6. Classify unseen examples
   (a) Allow selection of an unseen example & perform feature extraction on it
   (b) Classify unseen example according to model
7. Report results back to user
8. Code should be modular enough to allow possible later use of a variety of feature extraction methods

9. It should be possible to import/export any lengthy work done by user
10. Performance should be adequate for a typical interactive application

2.1.2 Video-based Classification Application

1. Access video stream from camcorder or webcam attached to PC via Eyematic `FaceTracker`
2. Use motion of features tracked by `FaceTracker` to extract numerical data describing spontaneous expressions
3. Upon user request, capture new training example together with user-provided emotion class label
4. Make SVM parameters user-controllable
5. Train an SVM instance
   (a) Gather training data accumulated thus far
   (b) Create an SVM model from the data
6. Classify unseen examples — either upon user request ('snapshot' classification) or continuously for every frame
   (a) Use tracker to perform feature extraction
   (b) Classify unseen example according to model
7. Report results back to user, constantly for continuous classification
8. Performance *must* be high enough to appear instantaneous (continuous classification is useless if it does not appear to happen in real time)
9. It should be possible to import/export previously defined training data — this enables the user to skip training on subsequent sessions and aids person-independent classification[1]

2.2 The Problem of Automated Facial Expression Analysis

Automated facial expression analysis is a vast research field. Pantic & Rothkrantz [27] provide a comprehensive overview of the state of the art and define three core problems an approach to expression analysis needs to consider:

- Face detection in an image or image sequence
- Facial expression data extraction
- Facial expression classification

In both applications implemented, face detection is dealt with implicitly. For still images, it is assumed that these are of faces. For live video, `FaceTracker` automatically localizes facial features only if a face is present in the video stream.

[1]Where the person requesting classification is different from the person that supplied the training data.

Investigating the possibilities for facial expression data extraction formed a considerable part of the preparation for this project. While the inherently motion-based nature of `FaceTracker` limited the possibilities to the extraction of feature motion, the possibilities for extraction in images were numerous. The approaches considered are outlined in the remainder of this chapter.

Support Vector Machines were used for facial expression classification in both applications. Understanding their modus operandi and the theory behind them constituted the other major part of preparation for this project. A description of the SVM methodology is given below.

2.3 Terminology

Due to the mix of terms from machine learning, cognitive science and psychology often associated with facial expression analysis, some explanation is in order.

Features: Quantities introduced to describe data in a machine learning environment. For faces, these can range from low level features (raw pixel data, contours in edge detected images, etc.) to high level (representations for the nose, the eyebrows, etc.).

Landmarks: Points introduced to represent the position of certain face geometry (e.g. points superimposed on the nose tip or the pupils).

Example: The unit of data a machine learning algorithm deals with, e.g. the data extracted from a single face image.

Label: Designates what class a particular example belongs to. Given explicitly for training examples, calculated by a machine learning algorithm during classification for unseen examples.

Model: Representation of the training data constructed during the training phase of a machine learning algorithm.

2.4 Support Vector Machines in a Nutshell

Machine learning algorithms are systems that receive data as input during a training phase, build a model of the input and output a hypothesis function that can be used to predict future data. Given a set of labelled training examples

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)), y_i \in \{-1, 1\}$$

where the $\mathbf{x}_i$ are input data and the y_i are class labels, learning systems typically try to find a decision function of the form

$$h(\mathbf{x}) = sgn(\langle \mathbf{w} \cdot \mathbf{x} \rangle + b)$$

(where $\mathbf{w}$ is a vector of weights and b is called the bias) that yields a label $\in \{-1, 1\}$ (for the basic case of binary classification) for a previously unseen example $\mathbf{x}$.

The risk, or expected error, that the function learned makes wrong predictions on future data depends on both its performance on the training set (how many errors would be made if training examples were classified by the learned function) and its complexity. By analogy to the principle

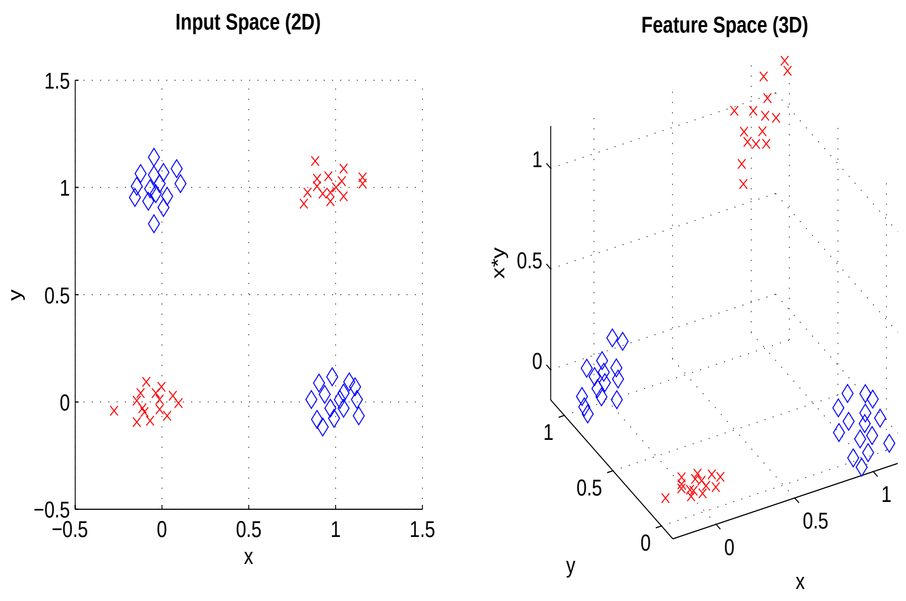


Figure 2.1: The 2-bit parity problem: data that is linearly inseparable in 2D (via a line) is linearly separable in 3D (via a plane).

of Occam's razor, simpler functions tend to exhibit greater generalization performance. Statistical learning theory, pioneered by Vapnik [34, 35], shows that this generalization performance depends on the complexity (encapsulated by the Vapnik-Chervonenkis (VC) dimension) of the class the function is chosen from, rather than the complexity of the function itself. The VC-dimension of a function class is measured by counting the number of possible data sets that functions from the class could perfectly explain without error.

Support Vector Machines, instead of being based on heuristics or analogies with natural learning systems, are based on results from statistical learning theory. Thus, theoretical guarantees can be made about their performance by placing an upper bound on the generalization error using the VC-dimension of the classes of the learned functions, essentially performing structural risk minimization. SVMs perform an implicit embedding of data into a high-dimensional feature space, where linear algebra and geometry may be used to separate data that is only separable with nonlinear rules in input space (Figure 2.1 illustrates using the classic parity problem). To do so, the learning algorithm is formulated to make use of kernel functions, allowing efficient computation of inner products directly in feature space, without need for explicit embedding. Given a nonlinear mapping ϕ that embeds input vectors into feature space, kernels have the form:

$$K(\mathbf{x},\mathbf{z}) = \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle$$

SVM algorithms separate the training data in feature space by a hyperplane defined by the type of kernel function used. They find the hyperplane of maximal margin, defined as the sum of the distances of the hyperplane from the nearest data point of each of the two classes. Statistical learning theory shows that generalization performance of a hyperplane depends only on its margin (which bounds the VC-dimension of the hyperplane), not on the dimensionality of the embedding space. Thus, SVMs avoid the "curse of dimensionality". The SVM methodology

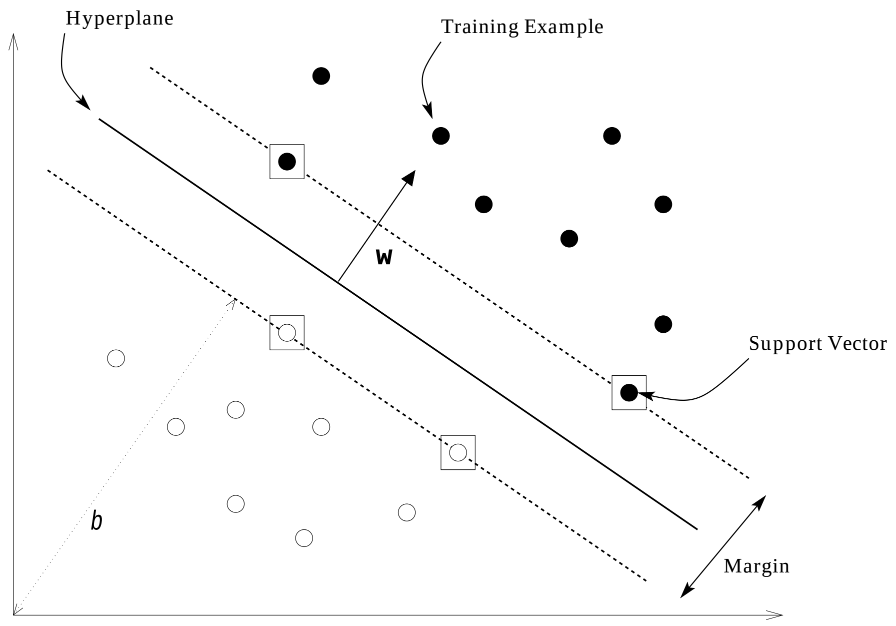


Figure 2.2: A linear maximal margin classifier. The hyperplane's normal vector **w** and the bias b define the decision function $f(\mathbf{x}) = sgn((\mathbf{w} \cdot \mathbf{x}) + b)$. The support vectors lie on the margin.

Kernel	**Formula**
Linear	$\mathbf{x} \cdot \mathbf{z}$
Polynomial	$(\gamma \mathbf{x} \cdot \mathbf{z} + c)^{degree}$
Radial Basis Function (RBF)	$\exp(-\gamma \lvert \mathbf{x} - \mathbf{z} \rvert^2)$
Sigmoid	$\tanh(\gamma \mathbf{x} \cdot \mathbf{z} + c)$

Table 2.1: Typically used kernel functions. $c, \gamma, degree$ are parameters used to define each particular kernel from the family given.

learns nonlinear functions of the form:

$$f(\mathbf{x}) = sgn(\sum_{i=1}^{l} \alpha_i y_i K(\mathbf{x}_i \cdot \mathbf{x}) + b)$$

where the α_i are Lagrange multipliers of a dual optimization problem. Hence, the training of SVMs amounts to convex quadratic optimization (solving a linearly constrained quadratic programming (QP) problem) and avoids the problem of local minima. It is possible to show that only some of the α_i are non-zero in the optimal solution, namely those arising from training points closest to the hyperplane, called support vectors. These induce sparseness in the solution and give rise to efficient approaches to optimization. Once a decision function is obtained, classification of an unseen example **x** amounts to checking which of the two subspaces defined by the separating hyperplane the example lies in. Figure 2.2 illustrates using a simple linear classifier.

The SVM approach is highly modular, allowing domain-specific selection of the kernel function used (Table 2.1 gives typical kernels). In contrast to previous "black box" learning approaches, SVMs allow for some intuition and human understanding. SVMs deal with noisy data and overfitting (where the learned function perfectly explains the training set but generalizes poorly) by introducing 'slack variables' to allow for some misclassifications. This handles data that is linearly inseparable even in higher space. Multi-class classification is accomplished by a cascade of binary classifiers together with a voting scheme.

SVMs have gained a large research following in the last few years. The interested reader is referred to [1], [3] as well as the excellent book [8].

2.5 Feature Extraction Approaches Considered

A variety of feature extraction methods for images were investigated. Due to time constraints, it was not possible to implement all of them. Given the nature of the dataset used[2], the possibility of integration with the live video application and published performance results [7], the approach based on landmark motion was implemented initially, as indicated in the proposal.

2.5.1 Gabor Filtering

Lades et al. [21] describe an approach to object recognition (including faces in images) based on an extension to artificial neural networks, in which Gabor wavelets are used to perform feature extraction. The idea is to convolve a gray-level input image with a series of Gabor wavelets, thereby performing a filtering operation. This can be done for the whole image but is in practice done selectively for particular locations. For each location, the magnitudes of the complex filter responses at a range of scales and orientations is combined into a vector (called a 'jet'). This jet characterizes the localized region of the face. Calculating such jets at discrete points in a lattice imposed on the face image (regularly spaced or adjusted to match a general face shape) yields a feature representation of the image.

2D Gabor wavelet filters are essentially sinusoidal planar waves restricted by a Gaussian envelope and can be described by

$$\psi_{\mathbf{k}}(\mathbf{x}) = \frac{\mathbf{k}^2}{\sigma^2} \exp\left(-\frac{\mathbf{k}^2\mathbf{x}^2}{2\sigma^2}\right) \left[\exp(i\mathbf{k}\mathbf{x}) - \exp\left(-\frac{\sigma^2}{2}\right)\right]$$

where the parameter $\mathbf{k}$ is the characteristic wave vector, which determines the wavelength and orientation of the filter as well as the width of the Gaussian window. It is defined as

$$\mathbf{k} = \begin{pmatrix} k_\nu \cos\phi_\mu \\ k_\nu \sin\phi_\mu \end{pmatrix}, \; k_\nu = 2^{-\frac{\nu+2}{2}}\pi, \; \phi_\mu = \frac{\pi}{8}$$

The filters possess both even (cosine) and odd (sine) parts. The convolution of the filter with the gray-level distribution $I(\mathbf{x})$ of an image is then given by

$$(\psi_{\mathbf{k}} * I)(\mathbf{x}_0) = \int\int I(\mathbf{x})\psi_{\mathbf{k}}(\mathbf{x} - \mathbf{x}_0)d\mathbf{x}$$

Figure 2.3 shows a Matlab plot of the real and imaginary parts of a typical Gabor wavelet. Figure 2.5 shows my Matlab results for convolving the face image in Figure 2.4 with a series of Gabor filters.

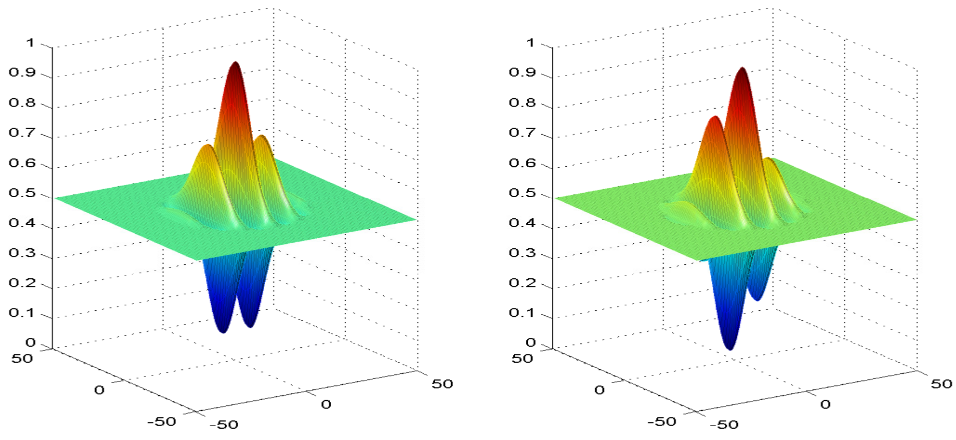


Figure 2.3: The real (cos, even symmetry) and imaginary (sin, odd symmetry) parts of a Gabor wavelet.

Figure 2.4: A typical face image taken from the Carnegie Mellon face database.

It has been argued that Gabor wavelets are a good model of the neural receptive fields in vertebrates such as cats [18]. They have been used in a number of feature extraction approaches such as [9] and [32]. Often, a very large number of features are extracted from an image (41,760 in case of the Gabor jet approach taken by Dailey et al. [9]), making efficient training of a machine learning algorithm very challenging. Application of this method to real time classification is thus infeasible in practice.

2.5.2 Eigenfaces

Turk & Pentland [33] first introduced a method for face recognition based on 'eigenfaces'. They take the information theoretical concept of extracting the relevant information from a face image and encoding it efficiently in order to compare it to others. Relevant information here is defined mathematically, instead of on human terms (e.g. position of the eyebrows, lips, etc.), which is seen as a safe bet given the poor understanding of what the human neural system considers relevant in a face image. The idea is to capture the variation among a number of face images and use it to compare individual images.

To do so, Turk & Pentland take a database of training images, considering each as a vector of

[2]See Section 2.7.

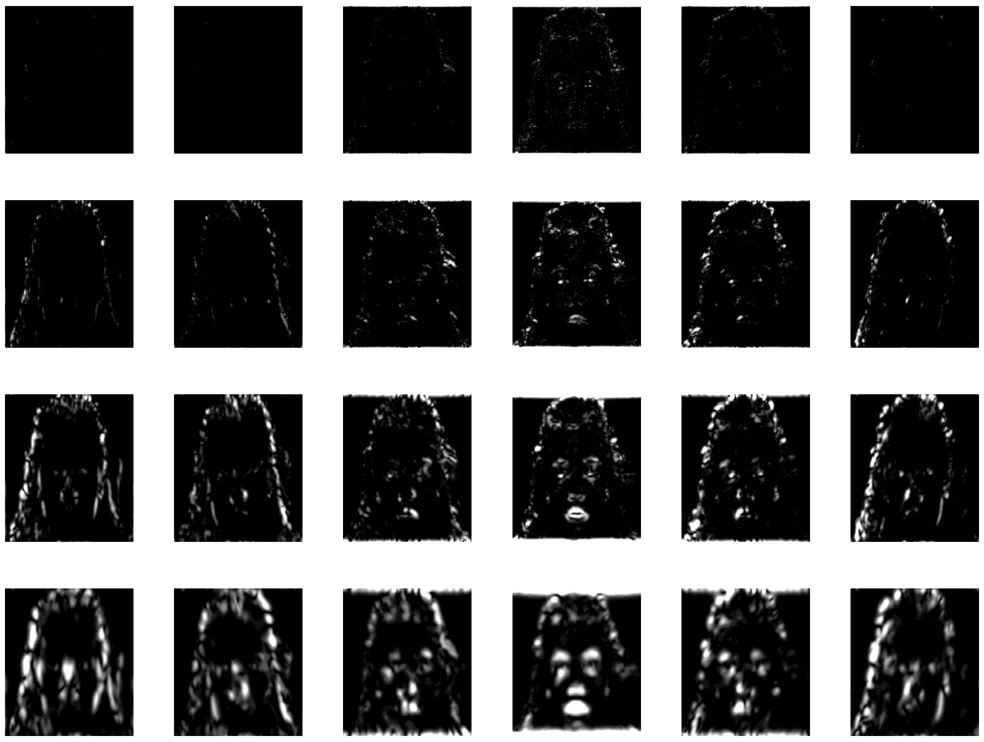


Figure 2.5: A face image after convolution with Gabor filters at different orientations (left-right) and different scales (top-down).

intensity values[3], and use the Karhunen-Loève expansion (also called principal component analysis, PCA) to find the eigenvectors of the covariance matrix of the set of images. Given, say, 128 $\times$ 128 pixel images, only some of all possible 16,384-dimensional vectors representing images will actually map to an image depicting a face and these will not be randomly distributed. PCA picks out those image vectors which best describe the distribution of face images in the space of all possible images. These hence define the subspace of face images, which Turk & Pentland coin the 'face-space'. The vectors themselves, being the eigenvectors of the covariance matrix of the images and having a ghostlike face appearance when displayed, are dubbed 'eigenfaces'.

Given a set of M training face images $\Gamma_1, \Gamma_2, \ldots, \Gamma_M$, one can calculate the "average" face by $\Psi = \frac{1}{M}\sum_{n=1}^{M} \Gamma_n$. The vector by which each face differs from the average is given by $\mathbf{\Phi}_i = \Gamma_i - \Psi$. PCA applied to the vectors $\mathbf{\Phi}_i$ finds M orthonormal vectors $\mathbf{u}_k$ and their eigenvalues λ_k which best describe the distribution. These correspond eigenvectors and eigenvalues of the covariance matrix

$$C = \frac{1}{M}\sum_{n=1}^{M} \Phi_n \Phi_n^T = AA^T$$

where $A = [\Phi_1\ \Phi_2\ \ldots\ \Phi_n]$. C is $N^2 \times N^2$, making calculation of the eigenvectors and eigenvalues computationally expensive. However, it is possible to solve an equivalent $M \times M$ problem, which is now determined by the size of the training set. The eigenvalues allow the eigenvectors to be ranked in order of importance in describing the variation among the images. Thus, it is possible to select M' "best" eigenfaces from the M eigenfaces calculated. These M' eigenfaces span the

[3]i.e. an $N \times N$ image is a vector of dimension N^2.

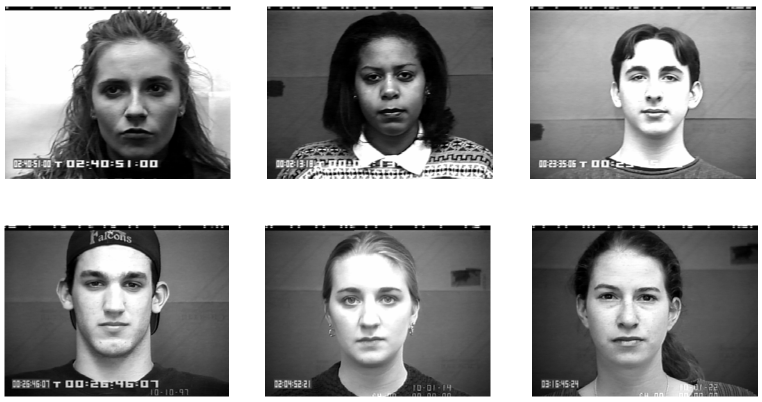


Figure 2.6: Training set of six images of different people displaying neutral expressions.

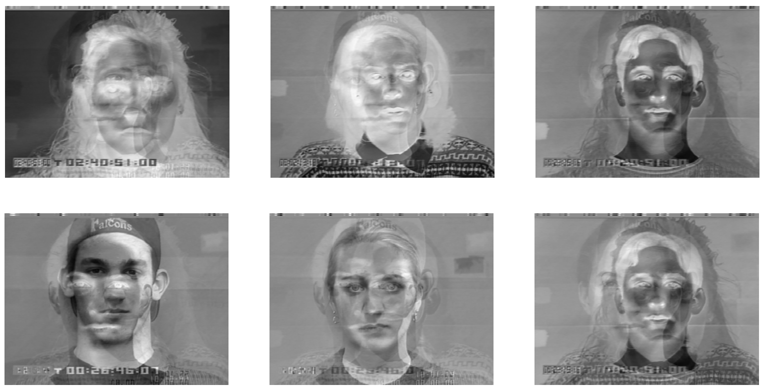


Figure 2.7: Six eigenfaces generated from the training images.

face-space. Figure 2.7 shows six eigenfaces generated in Matlab from the training set shown in Figure 2.6.

Each face image in the training set can be described exactly in terms of linear combinations of these eigenfaces. Thus, the set of contributions (or weight) of each eigenface in the decomposition of a face image in terms of the eigenfaces of the training set can be taken as the feature data extracted from an image. A new face image can be projected onto the face-space by calculating the weights $\omega_k = \mathbf{u}_k^T(\Gamma - \Psi)$ for each of the M' eigenfaces $\mathbf{u}_k$. Face recognition is performed by choosing the image in the training set with the weights that most closely match those of the new image. It is extensively reported that this approach works well under a variety of lighting conditions and head orientations.

To apply the technique to emotion recognition, one can calculate eigenfaces from a training set of images of one person expressing different emotions (Figure 2.9 shows the results for the training images in Figure 2.8), followed by the projection weights onto face-space of each training image. These weights can then be labelled and used as input to the training stage of a machine learning algorithm. Unseen images are then similarly projected onto face-space and the resulting weights used as input to the classification stage. Due to the template-based nature of the eigenfaces method, it is very difficult to use training images of different people for emotion classification. It is highly unlikely, for instance, that given (1) a joyful image of person A and (2) a sad image of person B as training data, a new face of person B with a joyful expression would be classified

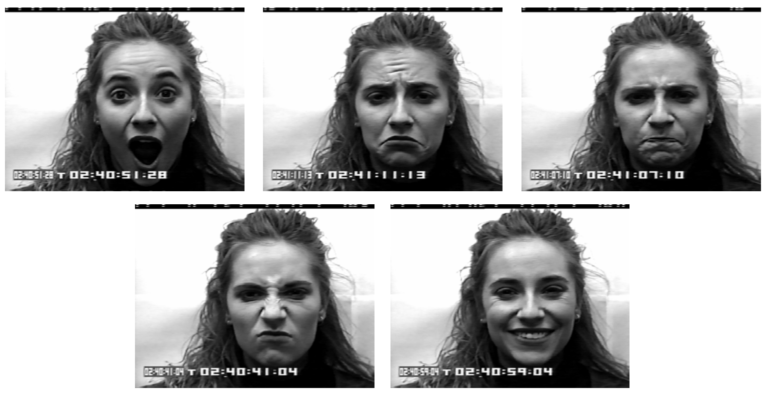


Figure 2.8: Training set of five images of the same person expressing different emotions.

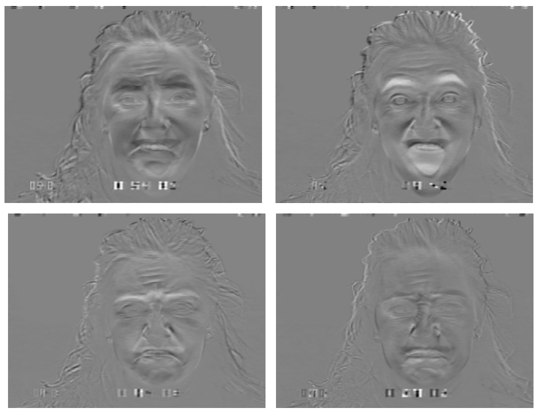


Figure 2.9: The top four eigenfaces generated from the training images.

as being joyful. The similarity to (2) stemming from being a picture of the same person far outweighs the similarity to (1) caused by the same emotion being expressed.

2.5.3 Facial Landmark Motion

Perhaps the most tried-and-true approach to feature extraction is to define a set of so called landmarks (or fiducial points) in the facial image according to the particular face model used and consider their motion across a sequence of images. By taking the displacements (Euclidean distances) of the landmarks between a neutral facial expression and the 'peak' of a particular emotive expression such as joy, one can establish a characteristic motion pattern for each emotion expressed, as shown in Figure 2.10. This method obviously relies on the presence of both neutral and peak images for each expression and is not applicable to individual still images.

The data extracted from an example thus consists of the vector of displacements of all landmark points. After training a machine learning algorithm on such data, classification essentially consists of matching a new, unseen motion pattern with the class of training patterns it most closely corresponds to. As evidenced in [27], a large number of expression analysis implementations use schemes related to facial landmark motion or similar lower level schemes like optical flow analysis. It has been shown that tracking the motion of landmarks is an effective approach to

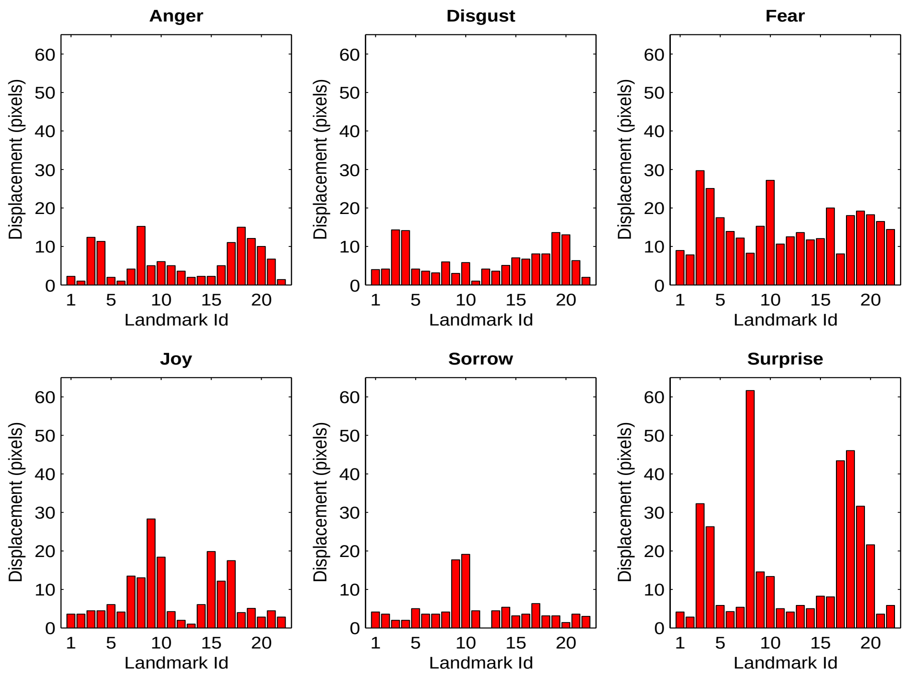


Figure 2.10: Landmark motion patterns for the six basic emotions of anger, disgust, fear, joy, sorrow and surprise.

expression analysis [7].

I decided to initially implement this data extraction method, using the same 22-point face model employed by Eyematic `FaceTracker` (shown in Figure 2.11).

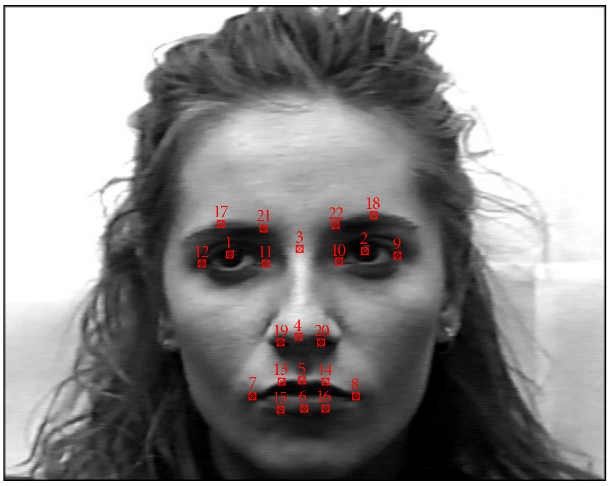
The possibility of integrating landmark position data output by `FaceTracker` later on in the project seemed appealing. The dataset used provided sequences of images for each expression, starting from a neutral frame and ending in a peak frame, thus allowing landmark motion to be calculated in the image-based application. Finally, correspondence with one of the authors of [9] suggested that a landmark motion approach was likely to yield better results than the Gabor wavelet method in particular.

2.6 Expression Classification Schemes

There are two main schemes according to which emotions are classified in expression analysis systems in the literature.

The most widely used scheme was originally proposed by Ekman [13]. He posits that there are six 'basic' emotions, namely anger, disgust, fear, joy, sorrow and surprise, from which more complex combinations can be constructed and which are easily identified (by humans) in facial expressions.

A much more complex but arguably more accurate and precise scheme is the Facial Action Coding System (FACS), due to Ekman and Friesen [15]. It codes expressions as a combination of



1. Right pupil
2. Left pupil
3. Nose root
4. Nose tip
5. Upper lip center
6. Lower lip center
7. Right mouth corner
8. Left mouth corner
9. Left outer eye corner
10. Left inner eye corner
11. Right inner eye corner
12. Right outer eye corner
13. Right upper lip
14. Left upper lip
15. Right lower lip
16. Left lower lip
17. Right eyebrow center
18. Left eyebrow center
19. Right nostril
20. Left nostril
21. Right inner eyebrow
22. Left inner eyebrow

Figure 2.11: The face model as used by Eyematic `FaceTracker`, consisting of 22 landmark points.

44 facial movements called Action Units (e.g. 'raised cheeks'). Human FACS experts are able to decompose a given expression into individual action units with very high accuracy.

Fully automatic classification at high precision by a computer according to FACS is an as-yet unsolved problem. Hence, most systems use the basic emotion (or a similarly simple) scheme. Because of this and in order to be able to compare results easily with previous approaches to emotion classification, I decided to use the basic emotion scheme wherever fixed, specific emotion categories were required.

2.7 Tools

2.7.1 SVM Library

As noted in the proposal, implementing an SVM algorithm from scratch is a huge task. Fortunately, there are a number of good, free SVM libraries available. After some research, I decided to use `libsvm` [4], which is widely used in the SVM community and of which equivalent C/C++ and Java implementations are available. It is very configurable, efficient, well documented and up-to-date.

2.7.2 Languages

During initial consideration before writing the proposal, I was concerned that the numerical computations (particularly large matrix operations) inherent in training an SVM or the various feature extraction techniques would mandate use of a compiled language like C or C++. After some further research, I decided to use Java to implement the image-based classification application for the following particular reasons:

1. Familiarity with the language
2. Availability of a cross platform, well documented GUI toolkit (AWT / Swing)
3. Availability of a a number of free image processing libraries & toolkits (e.g. Java2D)
4. Adequate performance with `libsvm` for SVM training & classification of moderate size data sets[4]

When I decided to attempt to implement the real time video classification application, Java was no longer an option. This is partly due to the soft real-time constraints placed on the training and classification performance of the SVM phase by a live video setting. More importantly, the interface to Eyematic `FaceTracker` is implemented as a Windows dynamically linked library, strongly suggesting C++ [31] as the implementation language. `FaceTracker` is written to interface with video hardware under Windows (using Microsoft's WDM video drivers), further dictating the use of C++ under Windows.

I used Visual Studio .NET as the IDE and compiler to implement the video application in C++. The Microsoft Foundation Classes (MFC) where used as the GUI toolkit. I had no prior practical experience with C++ or MFC. Because the opportunity to use `FaceTracker` to do video classification arose after I started implementing the image-based application, I had to learn both halfway through the project instead of doing so as part of the preparation stage.

There is widespread use of Matlab in the computer vision and expression analysis research communities, with a plethora of reference implementations existing only in Matlab and not in any of the more traditional languages. I decided that familiarity with writing Matlab code would be very useful particularly in the preparation stage to evaluate and visualize different feature extraction methods. I learned to use Matlab, which was of great help in gaining a better understanding of the various techniques by playing around with existing algorithms and rapidly writing prototype implementations.

2.7.3 Data Set

A large corpus of training and testing images was required as the basis for implementing and evaluating the image-based application. I used the released portion of the Cohn-Kanade facial expression database [19] developed at Carnegie Mellon University. It consists of over 2000 images of around 100 people. The data is organized into image sequences of subjects expressing each of the prototypic emotions outlined earlier, from the initial neutral frame up to the peak (or 'target') frame of the expression. Images are full-frontal 8-bit grayscale views of the subjects, with dimensions 640×480 or 640×490 pixels. They were captured using a standard video camera + recorder and timestamped. The example face images used throughout this dissertation are drawn from this database.

[4]As indicated by a number of users of the Java version of `libsvm` and evidenced by a simple program I wrote for classifying GIF images to be 'blue' or 'yellow' according to pixel color content.

2.7.4 Video Hardware

While developing the video application, I used a standard USB webcam to provide video at ∼15 frames per second for evaluation and debugging. Once the application was done, it was tested and evaluated using a Sony digital camcorder connected to a standard PC via IEEE Firewire. This setup provided significantly higher framerates (around 30 frames per second) and hence allowed `FaceTracker` to work more accurately.

2.7.5 Eyematic `FaceTracker`

`FaceTracker` is part of Eyematic Interfaces' `FaceStation` software package, used to animate faces of 3D characters in the modelling and animation software 3D Studio Max. It is normally directly integrated into 3D Studio Max as a plugin, but I was supplied with a C++ DLL and header files, allowing me to access the tracker output directly from within an arbitrary application.

`FaceTracker` uses a generic face template to initially locate the position of the 22 facial landmarks described earlier in the video stream and uses a filter to track their position over subsequent frames. It allows the position of the landmark points to be queried and provides an interface to the actual video stream. Its detailed mode of operation is described in Chapter 3.

Chapter 3

Implementation

This chapter details how the concepts introduced in Chapter 2 were employed to build both the image-based and the video-based classification applications. It is logically divided into two parts, each dedicated to one of the two applications implemented. Because both are heavily centered around user interaction, a high level description of each application in terms of the interface, the actions available to the user and the typical action sequence performed by the user is given first. Subsequently, the implementation is described at the program level. An overview of the application architecture is given, followed by a description of the program components and the techniques used during implementation.

3.1 Image-based Classification Application: Description

3.1.1 GUI

Figure 3.1 shows a screen capture of the application during a typical session. The main components of the graphical user interface are the following:

Title Bar: Displays the name of the application.

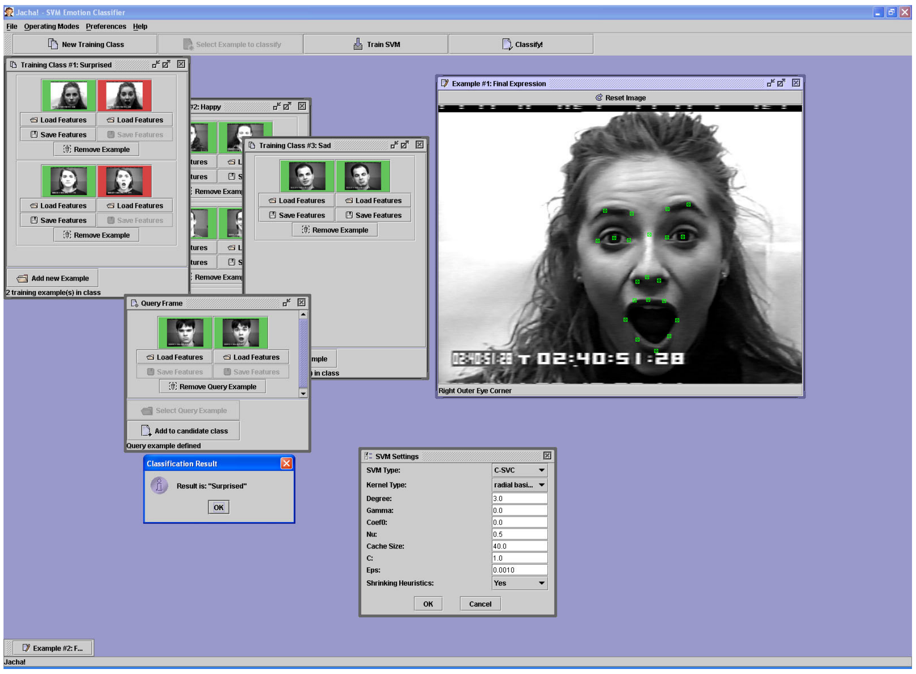
Menu Bar: Provides standard functionality, including an option to select between different operating modes[1] and a way to access a dialog to modify SVM parameters.

Toolbar: Provides buttons to access the main functionality of the application: definition of new training classes, selection of an unseen example to classify, training of the SVM and classification of the selected unseen example. It can be docked to a particular margin of the desktop or free-floating.

Main 'desktop' area: Contains internal frames for:

- Each defined training class
- The unseen example to be classified
- Landmark definition

[1]Included in order to make switching between different possible feature extraction methods simple.



2 training example(s) in class

Query example defined

Result is: "Surprised"

SVM Type:	C-SVC
Kernel Type:	radial basi...
Degree:	3.0
Gamma:	0.0
Coef0:	0.0
Nu:	0.5
Cache Size:	40.0
C:	1.0
Eps:	0.0010
Shrinking Heuristics:	Yes

Figure 3.1: A screenshot of the image-based classification application.

- SVM parameter selection
- Classification result reporting

Status bar: Reports various information back to the user.

3.1.2 Typical Action Sequence Walkthrough

A typical sequence of actions a user would perform during interaction is given below. Several GUI features present in Figure 3.1 are outlined along the way.

1. **User creates a new training class**. Done via a toolbar button. Classes normally correspond to a particular predefined emotion (e.g. joy) the user wishes to supply example images for, but could in principle be arbitrarily defined (the application has no knowledge of 'joy', but treats each class merely as a label to be attached to the examples). She is asked for a class label and a new internal frame bearing the class label as the title is spawned.

2. **User adds examples to the training class**. Via a file selection dialog, two images (a frame showing a neutral expression and the target frame of the expression) forming one example are added to the class. A variety of image formats are supported (GIF, JPEG, PNG, etc.). Iconified versions of the images are added to the training class frame (on a red background if landmarks have yet to be defined for the image, green otherwise), together with buttons to load/save landmarks for each image and to remove the example from the training class.

3. **User defines landmarks for each image**. Clicking on the iconified face images opens a new internal frame to perform landmark definition. A status bar in the new frame guides the user through the definition process, indicating which landmark (e.g. the nose tip) to select

Parameter	**Possible Settings, Explanation**
SVM Type	– C-SVC (Support Vector Classification, standard) – nu-SVC (controlled by different parameter) – one-class SVM – epsilon-SVR (Support Vector Regression) – nu-SVR
Kernel Type	– linear: $\mathbf{u} \cdot \mathbf{v}$ – polynomial: $(\gamma \mathbf{u} \cdot \mathbf{v} + c)^{degree}$ – radial basis function: $\exp(-\gamma \vert \mathbf{u} - \mathbf{v} \vert^2)$ – sigmoid: $\tanh(\gamma \mathbf{u} \cdot \mathbf{v} + c)$
Degree	Value of *degree* in kernels above
Gamma	Value of γ (the variance) in kernels above
Coef0	Value of c in kernels above
Nu	Parameter used in nu-SVC, one-class SVM and nu-SVR
Cache Size	Size (in MB) of the `libsvm` cache for sub-matrices of matrix Q used in quadratic optimization
C	Cost parameter used in C-SVC, epsilon-SVR and nu-SVR (penalty of misclassification)
Eps	The tolerance of the termination criterion ε
Shrinking Heuristics	Whether or not to use shrinking heuristics (an optimization reducing the size of the QP problem)

Table 3.1: The user-modifiable SVM parameters and their purpose.

next and signalling when definition is complete. The user clicks on the location in the image corresponding to each landmark to define it. Once definition is complete, the status of the iconified image in the training class frame is updated to reflect this (turned green). A reset button allows the user to start over. Once definition is complete, individual landmark points can be redefined by clicking on them and subsequently clicking on a new location.

4. **User saves/loads defined features**. Work done during landmark definition can be exported and later imported in the form of simple text files. This provides a convenient way to avoid duplicate work and reuse results from previous sessions, saving time particularly for large training sets.

5. **User modifies SVM parameters**. These can be accessed through a dialog as shown in Figure 3.1. They influence the operation of `libsvm` during the training and classification phases. Table 3.1 illustrates the settings that can be modified[2].

6. **User trains SVM instance**. Done via one of the toolbar buttons. Various sanity checks on the data are performed and the user is informed if anything (e.g. incomplete landmarks for an example) prevents the SVM from being trained.

7. **User selects unseen example for classification**. A new internal frame capable of holding a single example is created. A similar landmark definition process as described above takes place for the unseen example.

[2]See Chapter 4 or even [4] for a detailed explanation of each parameter.

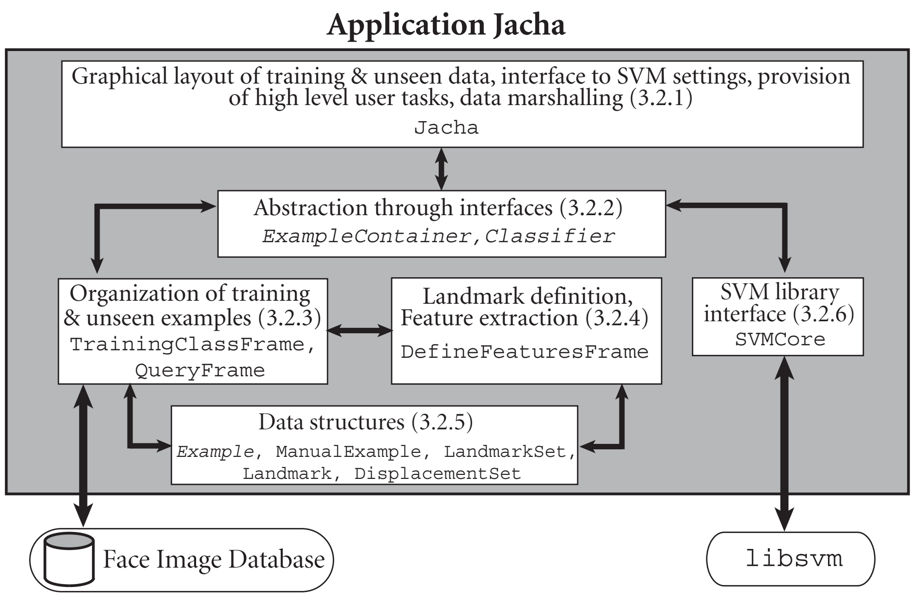

Application Jacha
Graphical layout of training & unseen data, interface to SVM settings, provision of high level user tasks, data marshalling (3.2.1)
Jacha
Abstraction through interfaces (3.2.2)
ExampleContainer, Classifier
Organization of training & unseen examples (3.2.3)
TrainingClassFrame, QueryFrame
Landmark definition, Feature extraction (3.2.4)
DefineFeaturesFrame
SVM library interface (3.2.6)
SVMCore
Data structures (3.2.5)
Example, ManualExample, LandmarkSet, Landmark, DisplacementSet
Face Image Database
libsvm

Figure 3.2: `Jacha` system architecture overview. Numbers give sections detailing each component. Class names implementing components are given in `typewriter` font. Additionally, interface names are *`italic`*.

8. **User requests classification**. Done via a toolbar button which becomes active only once a trained SVM instance and a fully defined unseen example are present.

9. **Result is reported**. A dialog opens reporting the classification result back to the user. He now has the option of adding the newly classified example to the training class reported as the result.

3.2 Image-based Classification Application: Structure

The abstract architecture of `Jacha`[3], the image-based application, is shown in Figure 3.2. Figure 3.3 gives the UML static structure diagram indicating composition and inheritance relationships for the application, including the most important class fields and methods.

The following pages detail the components shown in Figure 3.2 and the classes implementing them. To preserve readability, I have confined descriptions of the functionality of the methods and fields of each class to Appendix C.

3.2.1 Graphical Layout and High Level Tasks

At what is conceptually the 'top level' of the application, the following tasks are provided to the user:

- Definition of emotion classes to group training examples into

[3]Costa Rican slang meaning 'face'.

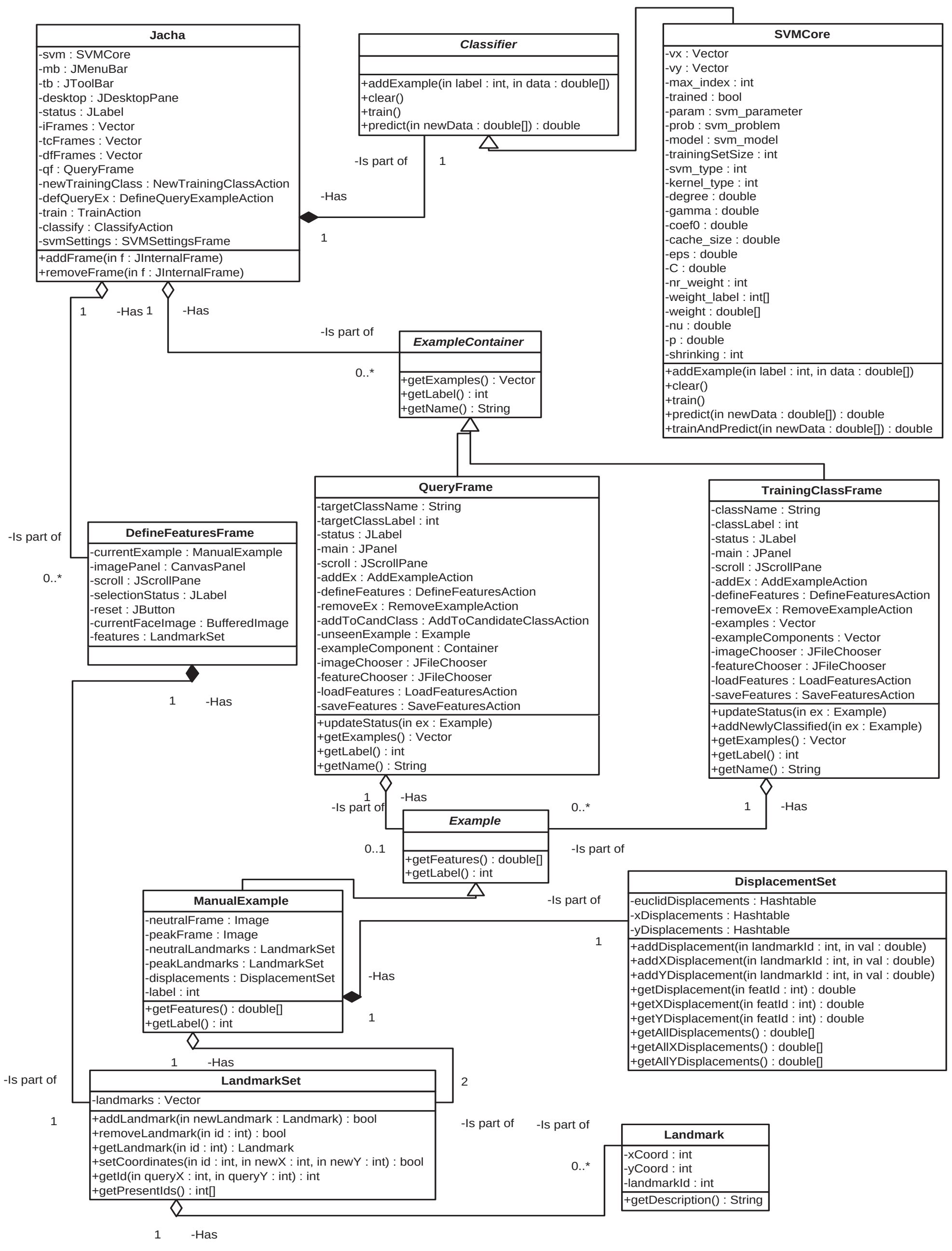

Jacha
-svm : SVMCore
-mb : JMenuBar
-tb : JToolBar
-desktop : JDesktopPane
-status : JLabel
-iFrames : Vector
-tcFrames : Vector
-dfFrames : Vector
-qf : QueryFrame
-newTrainingClass : NewTrainingClassAction
-defQueryEx : DefineQueryExampleAction
-train : TrainAction
-classify : ClassifyAction
-svmSettings : SVMSettingsFrame
+addFrame(in f : JInternalFrame)
+removeFrame(in f : JInternalFrame)
Classifier
+addExample(in label : int, in data : double[])
+clear()
+train()
+predict(in newData : double[]) : double
SVMCore
-vx : Vector
-vy : Vector
-max_index : int
-trained : bool
-param : svm_parameter
-prob : svm_problem
-model : svm_model
-trainingSetSize : int
-svm_type : int
-kernel_type : int
-degree : double
-gamma : double
-coef0 : double
-cache_size : double
-eps : double
-C : double
-nr_weight : int
-weight_label : int[]
-weight : double[]
-nu : double
-p : double
-shrinking : int
+addExample(in label : int, in data : double[])
+clear()
+train()
+predict(in newData : double[]) : double
+trainAndPredict(in newData : double[]) : double
-Is part of
1
-Has
1
1
-Has
1
-Has
-Is part of
0..*
ExampleContainer
+getExamples() : Vector
+getLabel() : int
+getName() : String
QueryFrame
-targetClassName : String
-targetClassLabel : int
-status : JLabel
-main : JPanel
-scroll : JScrollPane
-addEx : AddExampleAction
-defineFeatures : DefineFeaturesAction
-removeEx : RemoveExampleAction
-addToCandClass : AddToCandidateClassAction
-unseenExample : Example
-exampleComponent : Container
-imageChooser : JFileChooser
-featureChooser : JFileChooser
-loadFeatures : LoadFeaturesAction
-saveFeatures : SaveFeaturesAction
+updateStatus(in ex : Example)
+getExamples() : Vector
+getLabel() : int
+getName() : String
TrainingClassFrame
-className : String
-classLabel : int
-status : JLabel
-main : JPanel
-scroll : JScrollPane
-addEx : AddExampleAction
-defineFeatures : DefineFeaturesAction
-removeEx : RemoveExampleAction
-examples : Vector
-exampleComponents : Vector
-imageChooser : JFileChooser
-featureChooser : JFileChooser
-loadFeatures : LoadFeaturesAction
-saveFeatures : SaveFeaturesAction
+updateStatus(in ex : Example)
+addNewlyClassified(in ex : Example)
+getExamples() : Vector
+getLabel() : int
+getName() : String
-Is part of
0..*
DefineFeaturesFrame
-currentExample : ManualExample
-imagePanel : CanvasPanel
-scroll : JScrollPane
-selectionStatus : JLabel
-reset : JButton
-currentFaceImage : BufferedImage
-features : LandmarkSet
1
-Has
1
-Has
-Is part of
0..*
1
-Has
Example
+getFeatures() : double[]
+getLabel() : int
0..1
-Is part of
ManualExample
-neutralFrame : Image
-peakFrame : Image
-neutralLandmarks : LandmarkSet
-peakLandmarks : LandmarkSet
-displacements : DisplacementSet
-label : int
+getFeatures() : double[]
+getLabel() : int
-Is part of
1
-Has
1
DisplacementSet
-euclidDisplacements : Hashtable
-xDisplacements : Hashtable
-yDisplacements : Hashtable
+addDisplacement(in landmarkId : int, in val : double)
+addXDisplacement(in landmarkId : int, in val : double)
+addYDisplacement(in landmarkId : int, in val : double)
+getDisplacement(in featId : int) : double
+getXDisplacement(in featId : int) : double
+getYDisplacement(in featId : int) : double
+getAllDisplacements() : double[]
+getAllXDisplacements() : double[]
+getAllYDisplacements() : double[]
1
-Has
-Is part of
1
LandmarkSet
-landmarks : Vector
+addLandmark(in newLandmark : Landmark) : bool
+removeLandmark(in id : int) : bool
+getLandmark(in id : int) : Landmark
+setCoordinates(in id : int, in newX : int, in newY : int) : bool
+getId(in queryX : int, in queryY : int) : int
+getPresentIds() : int[]
2
-Is part of
-Is part of
0..*
Landmark
-xCoord : int
-yCoord : int
-landmarkId : int
+getDescription() : String
1
-Has

Figure 3.3: UML static structure diagram for the image-based classification application.

- Selection of an unseen example to classify
- Adjustment of SVM settings (as shown in Table 3.1)
- Training of an SVM instance
- Classification of an unseen example

In the main program class `Jacha`, each of these tasks is presented as a GUI action and implemented using Java Actions (via inner classes). These provide a clean way to encapsulate functionality triggered by GUI events and allow GUI elements to be easily created from them.

The graphical user interface makes use of Java Swing's [24] internal frames to display and organize information. They provide a convenient way to encapsulate data in each training class and handle layout of large training sets visually while allowing layout customizability by the user. Class `Jacha` controls the spawning and deletion of the various internal frames used for representing training classes, during landmark definition, etc. It also acts as a single point of reference for the individual internal frames and the SVM classifier, marshalling communication and data exchange among them.

3.2.2 Abstraction through Interfaces

Looking ahead at the possibility of using different feature extraction methods, I aimed to describe the interface between the training data and the application in terms of abstract classes or Java interfaces. Concrete instances of these can thus implement different feature extraction methods to obtain numerical data from images (e.g. displacements, Gabor filtering, etc.) and be easily swapped in. The same principle should hold for the interface between the particular classifier used and the application.

`ExampleContainer` is an interface specifying the operations any class containing examples should provide. This allows for a variety of concrete implementations. Both `TrainingClassFrame` and `QueryFrame` implement this interface. A scheme whereby a large training class could be exported/imported to/from disk as a concrete class `SerializedExampleContainer` and used in conjunction with `TrainingClassFrames` defined within the application could be imagined.

Similarly, the interface *`Classifier`* specifies the operations any machine learning classifier used by the application has to support. It ensures that different SVM implementations as well as non-SVM approaches to classification (e.g. a neural network) can easily be swapped in.

3.2.3 Organization of Training and Unseen Examples

Training examples belonging to the same user-defined emotion class (created via one of the top level actions in the application) are grouped inside an internal frame as depicted in Figure 3.4. The single unseen example to be classified at any time is contained in a similar internal frame.

The class `TrainingClassFrame` implements grouping of training examples. It does so both in terms of containing the actual objects representing examples as well as visually within the application by acting as an internal frame. As seen in Figure 3.4, it provides functionality for

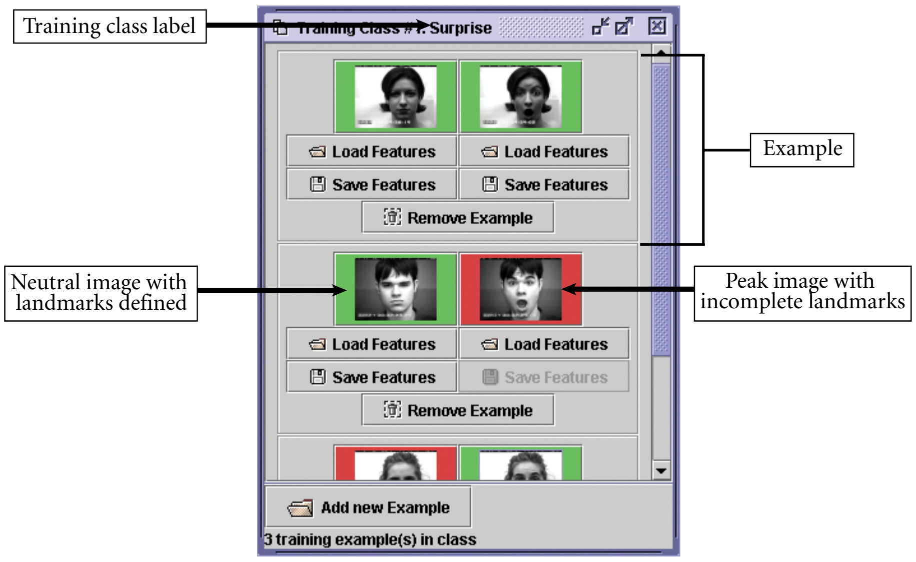

Training class label
Training Class #1: Surprise
Example
Neutral image with landmarks defined
Peak image with incomplete landmarks
Load Features
Save Features
Remove Example
Add new Example
3 training example(s) in class

Figure 3.4: Training examples of the same emotion class grouped inside an internal frame.

adding/removing examples, for importing/exporting defined features and for accessing the feature definition parts of the application.

The class `QueryFrame` is very similar in structure to `TrainingClassFrame`, with the obvious proviso that it can only contain the single example to be classified.

Both classes interact with the application through an additional layer of abstraction as outlined above, by implementing the interface *`ExampleContainer`*.

3.2.4 Feature Extraction and Landmark Definition

As outlined in Section 2.5.3, the features extracted from each example are the displacements of facial landmarks between a neutral and a peak expressive frame. The user defines landmarks for each image in an internal frame as shown in Figure 3.5.

An instance of the class `DefineFeaturesFrame` is created for each landmark definition process, spawning a new internal frame.

The class implements a simple state machine guiding the user through the manual definition of facial landmarks on top of a series of 22 facial points. Errors made during definition can be corrected by redefining individual landmarks.

Upon creation, each `DefineFeaturesFrame` is given the `ManualExample` object landmarks are being defined for. It generates a `LandmarkSet` for either the neutral or the peak frame of the example, which is subsequently stored in the `ManualExample` object.

3.2.5 Data Structures

Figure 3.6 illustrates the interaction of the data structures used during extraction of numerical data from images in order to supply it to the SVM for training or classification.

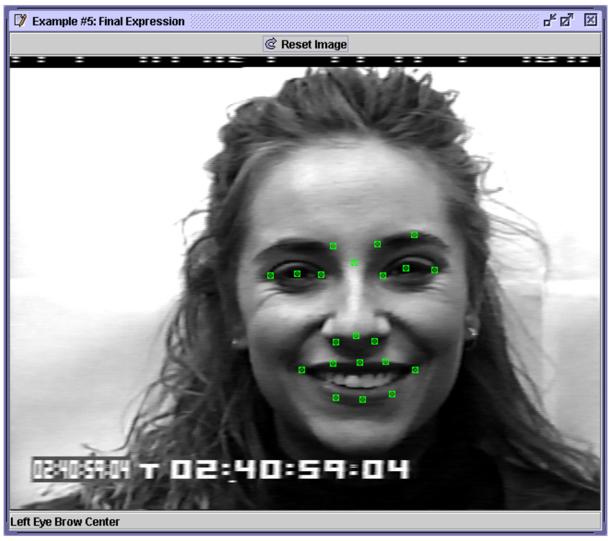


Figure 3.5: A typical landmark definition frame.

Acting as a layer of abstraction to the examples used as training or unseen data, the interface `Example` specifies the two operations any concrete instance implementing it must support. `getFeatures()` should return a `double[]` of values constituting the actual numerical example (vector $\mathbf{x}_i$ in the SVM equations in Section 2.4). `getLabel()` should return the numerical label of the class the example belongs to (y_i in the SVM equations). Just how numerical data is extracted from an example is determined by the particular instance implementing the interface.

A concrete instance of `Example`, class `ManualExample` represents examples with features created by manual landmark definition. It encapsulates the neutral and peak frame images of the example, the landmarks defined for each image as well as the displacements between the landmarks which constitute the features extracted from the example.

The class `LandmarkSet` holds `Landmark` objects and maps between landmark ids and the (x,y) pixel coordinates of each landmark. Each `Landmark` object in turn contains the (x,y) pixel coordinates and the unique id (given by the face model of 22 points) for a particular landmark, as well as a textual description (e.g. 'nose root').

Finally, the class `DisplacementSet` implements a simple data structure to hold the displacements of each landmark (along the x and y axes as well as in terms of Euclidean distance). It is from `DisplacementSet` that numerical data is extracted when calling `getFeatures()` on a `ManualExample` object.

3.2.6 SVM Library Interface

While `libsvm` was originally written in C++ and has since been ported to Java, neither of these versions of the library are particularly object-oriented. `libsvm` is often used together with auxiliary C programs provided by the authors for command-line training and classification of large numerical data sets contained in text files, such as medical measurements (e.g. heart-rate). There is essentially no state kept with the library and the entire interface to the library's training and classification functionality is given by the following methods:

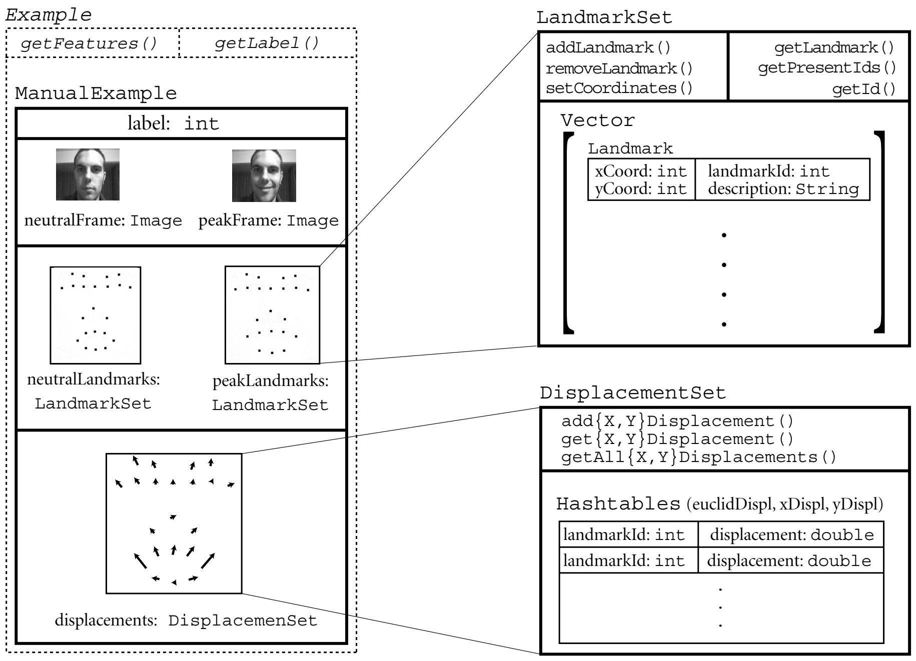


Figure 3.6: The data structures used in the image-based application.

```
svm_model svm_train(svm_problem prob, svm_parameter param)

double svm_predict(svm_model model, svm_node[] x)

void svm_save_model(String model_file_name, svm_model model) throws IOException

svm_model svm_load_model(String model_file_name) throws IOException

String svm_check_parameter(svm_problem prob, svm_parameter param)
```

The class `SVMCore` was written to encapsulate the library's functionality and provide a clean, object-oriented interface to the SVM. The motivation for doing so was twofold:

1. The emotion classification application should not rely on a particular SVM library. There is no need for the application to know the details of `libsvm`'s internal data formats (e.g. `svm_problem`, `svm_model`, `svm_parameter` above) and it should be possible to substitute `libsvm` by a different SVM library with relative ease.

2. It should be possible to keep training data and parameters with the SVM instance. Using the functionality provided by `libsvm`, it is only possible to export the model of the trained SVM (via `svm_model`) to disk and subsequently use it as-is, but it is not possible to augment this model by new training examples. This, however, is very useful if the user wishes to simply load an SVM instance already trained for emotion recognition of the basic emotions and augment it herself with new training examples to work for more complex emotion classes. Implementing a wrapper class allows the entire state of the classifier, including training data and SVM algorithm parameters, to be exported for later use by virtue of object serialization[4].

[4]This became particularly useful in the video application in order to perform person-independent classification.

`SVMCore` implements the *`Classifier`* interface, which specifies the operations any classifier used by the application should support. Training examples are first added to `SVMCore` and later converted to an `svm_problem` object for use by `libsvm` when training is requested. Similarly, `SVMCore` has fields for the individual parameters affecting SVM operation (such as the kernel type) and provides methods to modify them. An suitable `svm_param` object for `libsvm` is created upon training. The `svm_model` returned by `libsvm` after training is kept within `SVMCore` and used for subsequent classification. Unseen examples are converted to the appropriate `svm_node[]` before classification. Finally, `SVMCore` checks the parameters to the SVM before every training run, reporting any inconsistencies.

3.3 From Images to Live Video

Once the basic image-based application was implemented and the fundamental aims outlined in the proposal were met, I intended to continue work by integrating a more automated feature extraction method into the application, thus minimizing the amount of user interaction necessary to prepare training data. The Gabor filter approach outlined in Section 2.5.1 seemed particularly promising. I wrote a class implementing a Gabor filter bank and tested it by convolving some face images (using the `ConvolveOp` class provided by Java2D) in a simple test program.

However, after noticing that both training the SVM and subsequent classification in the image classification application took very little time even for large training sets, my supervisor suggested the possibility of combining a similar SVM approach with Eyematic `FaceTracker` and investigating classification for real time video. I hence changed project emphasis and concentrated my efforts on implementing `SVMTrack`, which is described next.

3.4 Video-based Classification Application: Description

3.4.1 GUI

Figure 3.7 shows a screen capture of the application during typical operation. The following main components are visible:

Title Bar: Displays the name of the application and gives the name of any trained SVM representation (imported from a file) that is being used.

Menu Bar: Provides standard Windows functionality, a menu to switch between different tracking modes and a menu to access a dialog for modifying SVM settings. Also allows the SVM to be cleared of any training data.

Toolbar: The buttons present are used for (left-to-right):

- Importing/Exporting SVM state from/to disk
- Accessing the three tracking modes ('play' = video output only, 'record' = track features, 'stop' = tracking off)
- Capturing a neutral expression
- Capturing training examples for each basic emotion

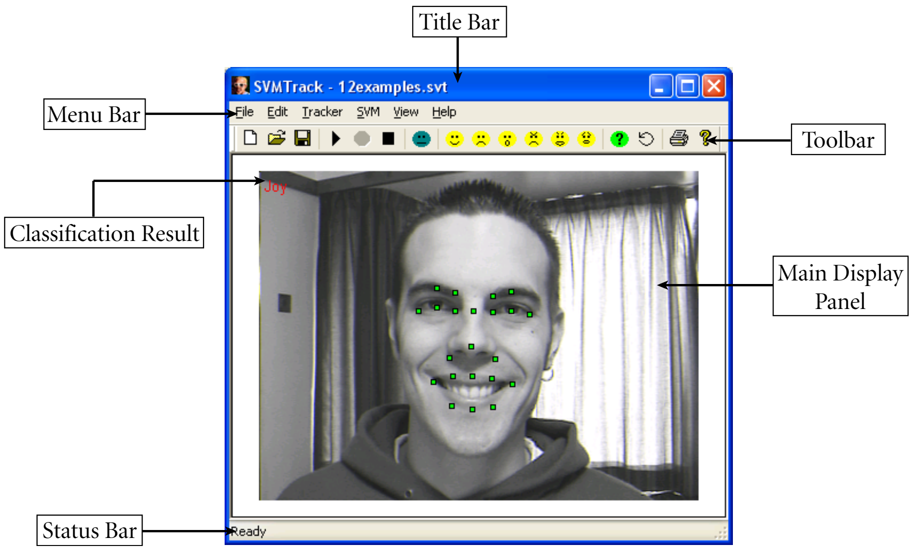


Figure 3.7: A screenshot of the SVMTrack, the video-based classification application.

- Requesting classification of the current frame ('snapshot' classification)
- Toggling continuous classification (on every frame)
- Printing the current display panel contents
- Accessing help

Main Display Panel: Video output is displayed here. Landmarks tracked by `FaceTracker` are superimposed on the video stream, as is the last classification result.

Status Bar: Reports various information back to the user.

3.4.2 Typical Action Sequence Walkthrough

The video-based application relies even more on user interaction, with the user now not only selecting the training data, but actually creating it by virtue of capturing his/her own facial expressions from video. Below is a typical sequence of actions a user would perform during a session, starting with an untrained SVM.

1. **User places tracker into 'play' mode.** This causes video to be output in the display panel without landmarks being tracked. Serves to adjust position and framing of the face in the video (should have a full frontal view of the entire face for the tracker to work optimally).

2. **User places tracker into 'record' mode.** As soon as `FaceTracker` has localized the 22 landmarks of the face model in the video stream, these are tracked across subsequent frames and drawn on top of the video output in the display panel.

3. **A neutral expression is captured.** This can be done either manually via a toolbar button or automatically when landmark motion is below a certain threshold over a number of frames (i.e. the user is expressing a still neutral expression over a few seconds).

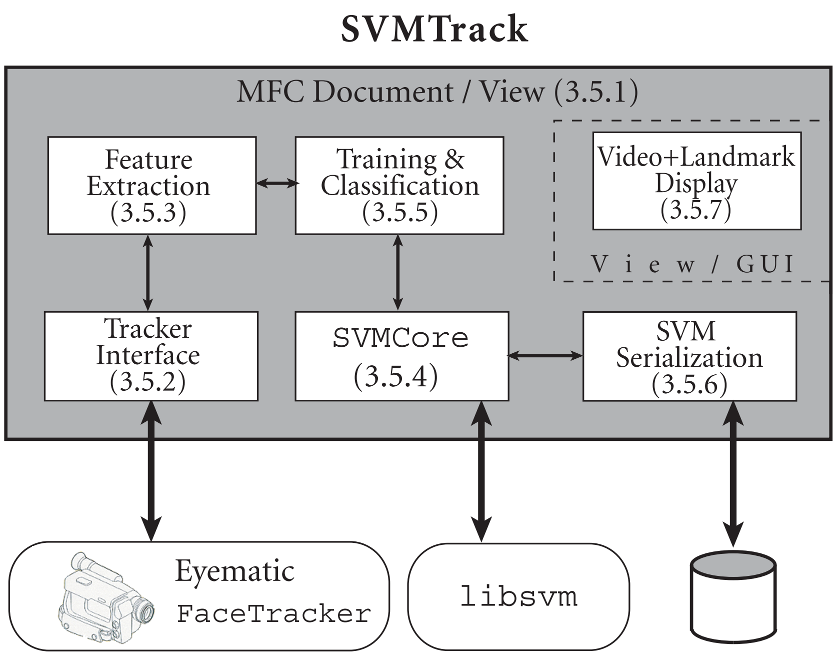

SVMTrack
MFC Document / View (3.5.1)
Feature Extraction (3.5.3)
Training & Classification (3.5.5)
Video+Landmark Display (3.5.7)
View / GUI
Tracker Interface (3.5.2)
SVMCore (3.5.4)
SVM Serialization (3.5.6)
Eyematic FaceTracker
libsvm

Figure 3.8: `SVMTrack` system architecture overview. Numbers give sections detailing each component.

4. **User expresses an emotion and requests creation of a training example at the peak frame**. This is done via one of the toolbar buttons corresponding to each basic emotion. Each time an example is added, the SVM is re-trained.

5. **An unseen emotion expressed by the user is classified**. This can be done either upon request ('snapshot' classification) via a toolbar button or for every new frame if continuous classification mode is active.

6. **User exports trained SVM to disk**. It can then be imported in subsequent sessions and the user can proceed straight to the classification phase.

3.5 Video-based Classification Application: Structure

Figure 3.8 shows the abstract architecture of `SVMTrack`, the video-based application. The UML static structure diagram for some of `SVMTrack`'s most important program components is shown in Figure 3.9. The remainder of this chapter details the main components shown in Figure 3.8 and the approaches taken to implement each.

3.5.1 MFC Document/View Model

As a single-windowed MFC application, `SVMTrack` is implemented in terms of the Document/View model suggested for Windows applications. This essentially calls for all data processing and event handling to be done within a single document class of the application which

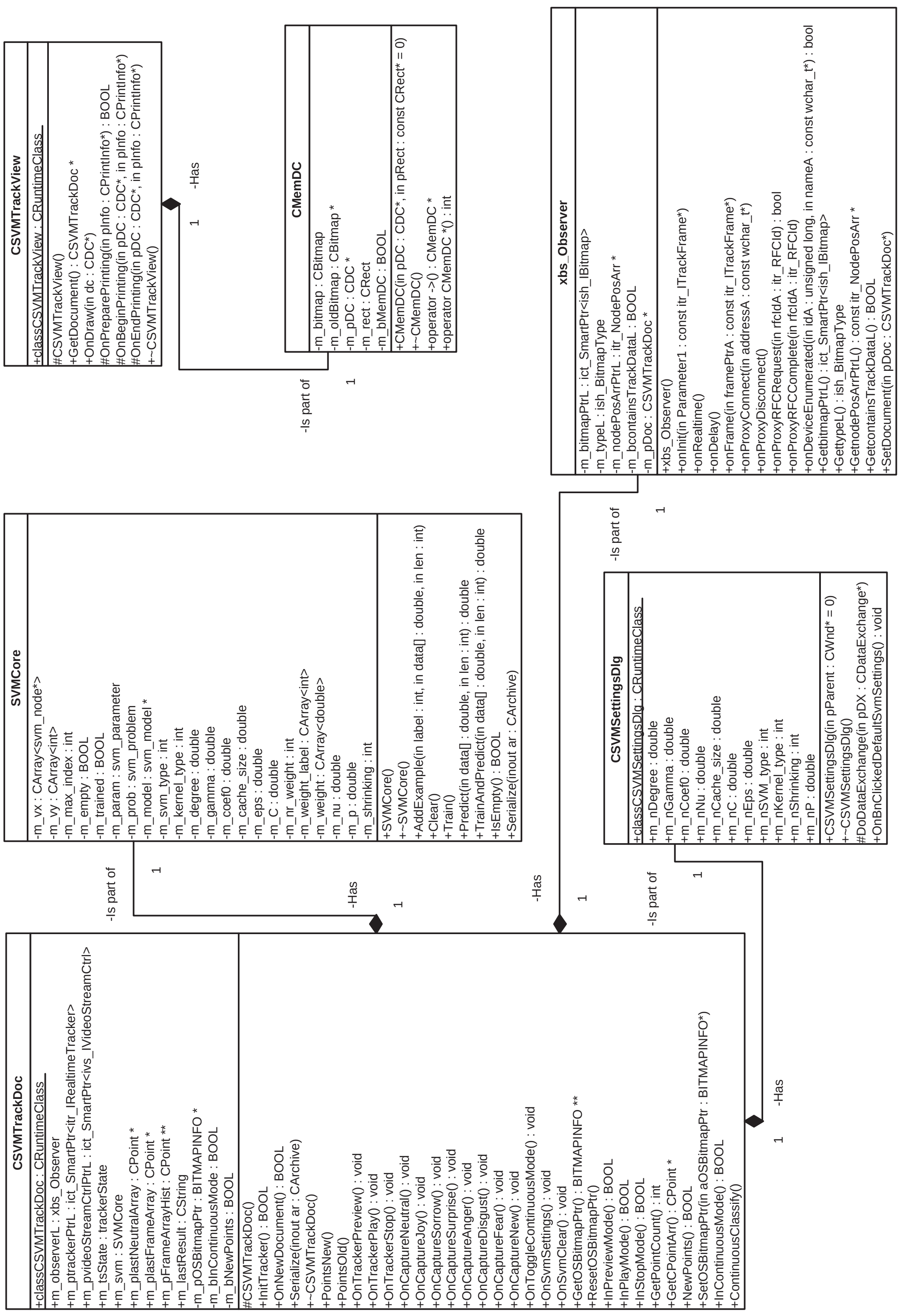

CSVMTrackView
+classCSVMTrackView : CRuntimeClass
#CSVMTrackView()
+GetDocument() : CSVMTrackDoc *
+OnDraw(in dc : CDC*)
#OnPreparePrinting(in pInfo : CPrintInfo*) : BOOL
#OnBeginPrinting(in pDC : CDC*, in pInfo : CPrintInfo*)
#OnEndPrinting(in pDC : CDC*, in pInfo : CPrintInfo*)
+~CSVMTrackView()
CMemDC
-m_bitmap : CBitmap
-m_oldBitmap : CBitmap *
-m_pDC : CDC *
-m_rect : CRect
-m_bMemDC : BOOL
+CMemDC(in pDC : CDC*, in pRect : const CRect* = 0)
+~CMemDC()
+operator ->() : CMemDC *
+operator CMemDC *() : int
xbs_Observer
-m_bitmapPtrL : ict_SmartPtr<ish_IBitmap>
-m_typeL : ish_BitmapType
-m_nodePosArrPtrL : itr_NodePosArr *
-m_bcontainsTrackDataL : BOOL
-m_pDoc : CSVMTrackDoc *
+xbs_Observer()
+onInit(in Parameter1 : const itr_ITrackFrame*)
+onRealtime()
+onDelay()
+onFrame(in framePtrA : const itr_ITrackFrame*)
+onProxyConnect(in addressA : const wchar_t*)
+onProxyDisconnect()
+onProxyRFCRequest(in rfcIdA : itr_RFCId) : bool
+onProxyRFCComplete(in rfcIdA : itr_RFCId)
+onDeviceEnumerated(in idA : unsigned long, in nameA : const wchar_t*) : bool
+GetbitmapPtrL() : ict_SmartPtr<ish_IBitmap>
+GettypeL() : ish_BitmapType
+GetnodePosArrPtrL() : const itr_NodePosArr *
+GetcontainsTrackDataL() : BOOL
+SetDocument(in pDoc : CSVMTrackDoc*)
SVMCore
-m_vx : CArray<svm_node*>
-m_vy : CArray<int>
-m_max_index : int
-m_empty : BOOL
-m_trained : BOOL
-m_param : svm_parameter
-m_prob : svm_problem
-m_model : svm_model *
-m_svm_type : int
-m_kernel_type : int
-m_degree : double
-m_gamma : double
-m_coef0 : double
-m_cache_size : double
-m_eps : double
-m_C : double
-m_nr_weight : int
-m_weight_label : CArray<int>
-m_weight : CArray<double>
-m_nu : double
-m_p : double
-m_shrinking : int
+SVMCore()
+~SVMCore()
+AddExample(in label : int, in data[] : double, in len : int)
+Clear()
+Train()
+Predict(in data[] : double, in len : int) : double
+TrainAndPredict(in data[] : double, in len : int) : double
+IsEmpty() : BOOL
+Serialize(inout ar : CArchive)
CSVMSettingsDlg
+classCSVMSettingsDlg : CRuntimeClass
+m_nDegree : double
+m_nGamma : double
+m_nCoef0 : double
+m_nNu : double
+m_nCache_size : double
+m_nC : double
+m_nEps : double
+m_nSVM_type : int
+m_nKernel_type : int
+m_nShrinking : int
+m_nP : double
+CSVMSettingsDlg(in pParent : CWnd* = 0)
+~CSVMSettingsDlg()
#DoDataExchange(in pDX : CDataExchange*)
+OnBnClickedDefaultSvmSettings() : void
CSVMTrackDoc
+classCSVMTrackDoc : CRuntimeClass
+m_observerL : xbs_Observer
+m_ptrackerPtrL : ict_SmartPtr<itr_IRealtimeTracker>
+m_pvideoStreamCtrlPtrL : ict_SmartPtr<ivs_IVideoStreamCtrl>
+m_tsState : trackerState
+m_svm : SVMCore
+m_plastNeutralArray : CPoint *
+m_plastFrameArray : CPoint *
+m_pFrameArrayHist : CPoint **
+m_lastResult : CString
-m_pOSBitmapPtr : BITMAPINFO *
-m_bInContinuousMode : BOOL
-m_bNewPoints : BOOL
#CSVMTrackDoc()
+InitTracker() : BOOL
+OnNewDocument() : BOOL
+Serialize(inout ar : CArchive)
+~CSVMTrackDoc()
+PointsNew()
+PointsOld()
+OnTrackerPreview() : void
+OnTrackerPlay() : void
+OnTrackerStop() : void
+OnCaptureNeutral() : void
+OnCaptureJoy() : void
+OnCaptureSorrow() : void
+OnCaptureSurprise() : void
+OnCaptureAnger() : void
+OnCaptureDisgust() : void
+OnCaptureFear() : void
+OnCaptureNew() : void
+OnToggleContinuousMode() : void
+OnSvmSettings() : void
+OnSvmClear() : void
+GetOSBitmapPtr() : BITMAPINFO **
+ResetOSBitmapPtr()
+InPreviewMode() : BOOL
+InPlayMode() : BOOL
+InStopMode() : BOOL
+GetPointCount() : int
+GetCPointArr() : CPoint *
+NewPoints() : BOOL
+SetOSBitmapPtr(in aOSBitmapPtr : BITMAPINFO*)
+InContinuousMode() : BOOL
+ContinuousClassify()
-Has
1
-Is part of
1
-Has
1
-Is part of
1
-Has
1
-Is part of
1
-Has
1
-Is part of
1

Figure 3.9: UML static structure diagram for the video-based classification application.

maintains all state, while one or more view classes are responsible only for presenting the information provided by the document visually within a GUI.

Implementing the application in accordance with the Document/View model proved to be very helpful, allowing me to easily make use of much functionality offered by MS Visual Studio and MFC such as serialization, event handling, etc. that relies on the application being present in a Document/View structure. The Document/View development process is also extensively documented, making it simple to refer back to the development guidelines when in doubt.

3.5.2 Interface with `FaceTracker`

`FaceTracker` is accessed through an SDK that Eyematic call a Core Technology Interface (CTI), which is similar in behaviour to Microsoft's COM framework, but is intended to be more lightweight and cross-platform. CTI provides one main template class (`ict_SmartPtr`) from which all objects encapsulating `FaceTracker`'s functionality are spawned. To do so, an `ict_SmartPtr` class is first specialized to the interface of the object we wish to create by defining its template parameter. Subsequently, an object of the desired class is created from the specialized pointer by calling the `createObject` method with the class id (defined by CTI) of the desired class. It is also possible to request a different interface to an already existing object. For example, an object representing the real time tracker can be accessed through the `ivs_IVideoStreamCtrl` interface to control attached video devices, access the frame content, and so forth. The following code illustrates creation of objects representing the real time tracker and the video stream:

```
ict_SmartPtr<itr_IRealtimeTracker> trackerPtr;
ict_errorCode ec = trackerPtr.createObject(itr_RealtimeTracker_ID);
if (ec == ict_OK) {
    // trackerPtr now points to a real time tracker object
}

ict_SmartPtr<ivs_IVideoStreamCtrl> vidStreamPtr;
vidStreamPtr.getInterfaceFrom(trackerPtr);
// vidStreamPtr now points to a video stream object
```

The CTI provides a variety of other interfaces to access `FaceTracker` functionality, for example the capability to track landmarks offline in an AVI file or to customize the real time tracker via configuration files. I particularly used `ish_ISerializable` to load definition files representing the 22-point face model and `ish_IBitmap` to access the raw bitmap of every video frame and hence output the video stream in the display panel.

Once a tracker object is instantiated and initialized, it will generate a number of events whenever a new video frame is present, new landmark positions are available, etc. To handle these events, an object of class `xbs_Observer` must be created and registered with the tracker. It provides methods such as `onInit`, `onFrame`, etc. that act as event handlers and can also handle remote network access to the tracker output. In `SVMTrack`, the observer is part of the document and is responsible for updating the document's data structures holding frame bitmap data or landmark positions as new information is made available by the tracker. Furthermore, it triggers view updates so that updated frame data and landmark positions are displayed in the panel as they become available.

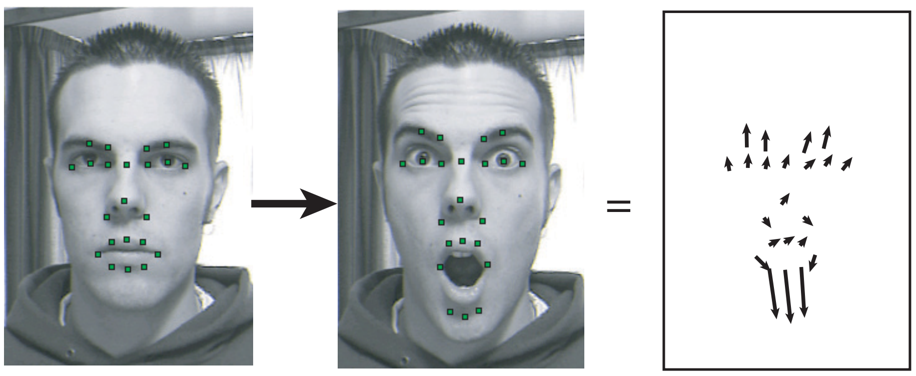


Figure 3.10: Feature extraction in video. The landmark displacements between a neutral and a peak frame of an expression constitute the data extracted.

3.5.3 Feature Extraction

`SVMTrack` uses landmark displacements as the features used to construct SVM training examples. To do so, it is necessary to capture landmark positions for frames corresponding to a neutral and a peak expression, as illustrated in Figure 3.10. It is assumed that whenever the user selects one of the basic emotion training buttons, he will be at the peak frame of the expression. Capturing a neutral expression is more subtle. The user is able to explicitly specify capture through one of the toolbar buttons, but this is rather inelegant (requiring two captures for every training example created).

I hence implemented a scheme whereby neutral landmark positions are automatically captured whenever landmark motion over the last *n* frames is below a certain experimentally determined threshold (i.e. the user has remained still for some time, presumably with a neutral face expression). This is done by keeping a small history of landmark positions over the last couple of frames in a FIFO buffer. Whenever a new set of landmark positions is made available by the tracker, it is added to the buffer (with the oldest set of positions being removed). The amount of motion between consecutive sets of landmark positions in the history is then calculated. If overall motion is below the threshold, the latest set of landmark positions is stored in the document as representing the last neutral expression. Landmark displacements between it and subsequent expressive frames can then be extracted.

Head motion is a significant problem when using landmark displacements. We often inadvertently move our head during facial expression (e.g. tilting our head up while expressing surprise). This obviously introduces noise into the landmark displacements by augmenting them with motion not actually corresponding to facial movement.

To deal with head motion, `SVMTrack` normalizes all landmark displacements with respect to a singe fiducial landmark that approximates head motion well. I used the nose root, since it is not affected by any major facial muscles. Motion of the nose root in the video should thus exclusively be due to head motion.

3.5.4 Adapting `SVMCore`

I used the C++ version of `libsvm` within `SVMTrack`. Just like in the image-based application, there should be a clean object oriented interface to the library. I hence rewrote the class `SVMCore` in C++ and used it for all training & classification in the video-based application. The operations it supports are identical to the Java version. The actual implementation of the C++ version of `SVMCore` was made rather more tricky due to the explicit memory allocation/deallocation necessary in C++ as well as the fact that interaction with the C++ version of `libsvm` relies heavily on pointer-based data structures. This held true especially when implementing serialization of `SVMCore` (see below).

3.5.5 Training and Classification

The features (displacements) extracted from the video stream together with the label of the basic emotion associated with the button selected by the user during capture constitute one training example. Examples are added to the set of training data kept in `SVMCore`. The SVM is trained each time a new example is created. This ‘incremental training’ allows the SVM to be used for classification as soon as even one example is available, eliminating the need for an explicit training action after the examples have been created.

There are two ways of classifying unseen expressions. The user can request ‘snapshot’ classification manually as soon as a trained SVM is available. At that point, displacements are extracted relative to the last neutral expression as before and used as the unseen example to be classified. Again, this is rather inelegant, not really corresponding to the continuous, real-time nature of the input video stream. I hence tried to find a way of triggering classification automatically at the appropriate moment in the expression.

Facial motion leading to a particular target basic expression can be described in terms of three phases. Initially, the landmarks are static, corresponding to the neutral expression. This is followed by a period of high landmark motion, while the facial features (eyebrows, lips, etc.) move towards their rough destination locations. Finally, there is a ‘homing-in’ stage where small, person-specific adjustments to the landmark positions are made. Landmark motion is significantly less intense during this stage. Triggering classification during this stage should enable landmark displacement to be extracted accurately enough for the classifier to distinguish between each emotion’s characteristic displacement pattern. I extended the idea used to automatically capture the neutral expression at appropriate points in time to work similarly for triggering classification. Conceptually, it works as outlined in Pseudocode 1.

I found this to be a bit fragile, especially since the thresholds in the method (e.g. how do you quantify “minimal landmark motion”) had to be determined by trial-and-error. I decided to attempt the simplest approach of just triggering classification for *every* frame when in continuous mode and found that classification overhead was so small that this was very feasible, even for video at 30 frames per second. This is due to the fact that most overhead lies in training the SVM, while classification basically reduces to a simple evaluation of the decision function (as outlined in Section 2.4). The slight drawback of this approach is that misclassifications occur for frames where the expression is either close to neutral or moving towards a peak expression for some emotion but not yet pronounced enough to be clearly distinguished from other expressions. This causes the classification result to “jitter” between the various basic emotions. It can be

Pseudocode 1: Neutral expression capture and triggering classification

```
{Counters used to forcibly insert delays between repeated capture of neutral expression(k) / triggering of classification (j)}
Initialize counter $k := history\_size$
Initialize counter $j := history\_size$
for (every new set of landmark positions produced by tracker) do
  Update landmark position history with new data
  if $k > 0$ then
    $k := k - 1$
  end if
  if $j > 0$ then
    $j := j - 1$
  end if
  if (there is minimal landmark motion across entire history AND $k = 0$) then
    Capture neutral expression
    Reset $k := history\_size$
  else if (history indicates landmark motion across consecutive frames was high and is now decreasing AND current landmark positions are far enough from neutral to be an emotive expression AND $j = 0$) then
    Trigger classification
    Reset $j := history\_size$
  else
    Do nothing for this iteration
  end if
end for
```

remedied somewhat by explicitly including ‘neutral’ among the classes trained for. Any frame whose expression is still closer to neutral than the peak of any basic expression will then be classified as neutral.

The resulting basic emotion of the last classification is reported back to the user superimposed on top of the video stream.

3.5.6 Serialization of SVM State

In order for someone to just be able to use the video-based application to classify his or her expressions without having to go through any lengthy training stage, it should be possible to import a fully trained SVM from a previous session and proceed straight to the classification phase. This also makes handling person-independent classification significantly easier by using training examples created by a certain user to classify the emotions of someone else. To evaluate `SVMTrack`, I wanted to particularly find out how well it performed for the person-independent case.

Because all state of the SVM classifier, including parameters and the training data, is encapsulated within `SVMCore`, serializing an `SVMCore` object was the most straightforward approach to exporting SVM state to disk. To be able to serialize objects of a particular class in MFC, the class needs to implement the `Serialize` method, which operates on a `CArchive` object provided by MFC. `CArchive` can then be treated conceptually as a stream of bytes which is written to or read from (using overloaded operators $<<$ and $>>$). All marshalling from program data structures to a sequence of bytes and back needs to be done explicitly by the programmer. Serializing `SVMCore` thus required an ordered traversal of all its fields and internal data structures, many of which make heavy use of pointers. Explicit delimiters (e.g. indicating the size of an array) had to be put into the byte stream for later reconstruction of data. To import a serialized `SVMCore` representation, a new `SVMCore` object is instantiated and its internal data is then reconstructed from the byte stream, with the appropriate amounts of memory being allocated on the heap. It can then be used for classification or augmented by further training examples.

The Document/View structure of `SVMTrack` then allowed me to use standard Windows file open/close dialogs to import/export SVM state. The `*.svt` files generated by `SVMTrack` can be assigned custom icons and are automatically associated with `SVMTrack`, launching the application when they are double-clicked.

3.5.7 Displaying Video and Landmarks

Each time a new video frame is present, the observer object registered with the tracker causes the view to be updated. In order to output the video in the display panel, the view's `OnDraw` method was overridden to do the following:

1. Access the raw packed bitmap (array of pixel intensity values) of the current frame (via a pointer stored in the document) and convert it to an MFC `CBitmap` object.
2. Use `BitBLT` to efficiently copy the bitmap directly to display memory.
3. If the tracker is in 'record' mode (i.e. actively tracking landmarks), step through the latest set of landmarks and draw them on top of the bitmap in the correct locations.
4. If classification was requested before, draw the name of the target basic emotion on top of the bitmap.

All GUI drawing in MFC is done through a device context object (of class `CDC`), which provides all necessary methods. The standard device context class does not support double buffering and, while suitable for most drawing, does not handle video at 30 frames per second well. The initial video output was rather choppy and showed lots of flicker. A derived device context class was thus used within `SVMTrack`. It inherits the usual functionality from the standard class `CDC`, but implements double buffering internally using two bitmaps, redirecting all drawing to an off-screen bitmap which is then instantaneously swapped on-screen, resulting in no flicker.

Chapter 4

Evaluation

One of the prime motivations behind implementing the two applications described in Chapter 3 was to evaluate SVMs in the context of facial expression recognition and emotion classification. As such, this chapter addresses the following questions:

- What are the issues involved in using landmark displacements as data for training & classification?
- What general recognition accuracy can be achieved using the SVM / landmark displacement approach? How does it vary for each basic emotion? How is it affected by adjusting SVM parameters, particularly kernel functions? How does it compare to previous approaches quoted in the literature?
- What is the computational overhead of training an SVM classifier on a typical set of facial expression examples? What is the overhead of subsequently classifying unseen examples? Is real time performance achievable?
- What recognition accuracy can be achieved for video-based classification? How does it vary between person-dependent and person-independent classification?

In addition, the software testing performed on the applications implemented while establishing answers to the above questions is outlined.

4.1 Evaluation Methodology

I initially evaluated the landmark displacement / SVM approach to emotion classification using images instead of directly from video in order to establish classification accuracy bounds as exactly as possible. Subsequent trials with the video application showed that the accuracy achieved for still images with manually defined landmarks transfers fairly well to video, but generally serves as an upper bound to what is achievable. This should not be surprising, due to the presence of many factors in a video environment that can adversely affect classification accuracy, such as head motion, different lighting conditions, distance to the camera, and so forth.

I used the CMU face database described in Section 2.7.3 to evaluate classification accuracy. Using these images recorded under carefully specified conditions allowed me to establish reproducible

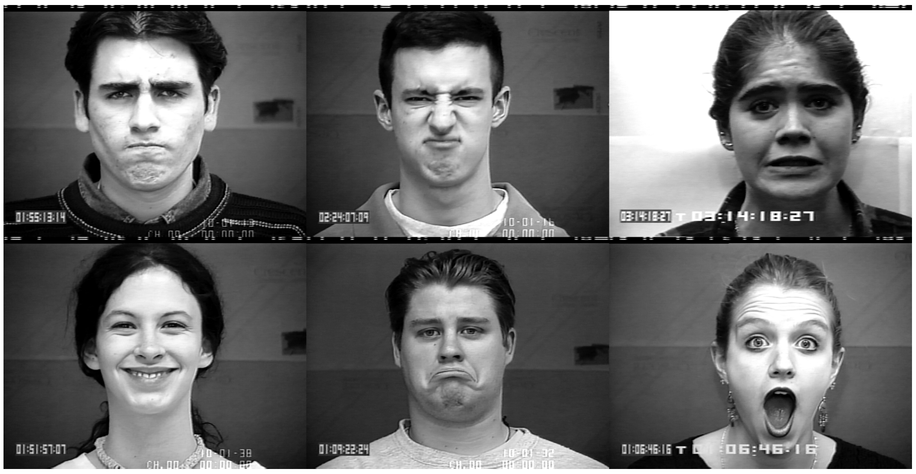


Figure 4.1: Six typical example images in the evaluation set expressing anger, disgust, fear, joy, sorrow and surprise.

results while still providing sufficiently varied data to minimize the dependence of accuracy results on gender, ethnicity or 'expressive enthusiasm' of particular subjects. A set of 20 training examples for each of the basic emotions was drawn from the face database and subsets used as training data for runs of varying size. Each example was taken from a different person. A number of unseen examples to be classified were drawn from the same database. As with all machine learning evaluation, training and test data were kept separate throughout. Figure 4.1 shows six typical face images in the training set, each corresponding to a different basic emotion.

Table 4.1 gives the mean and standard deviation for the landmark displacements of each basic emotion as extracted from the training data used for evaluation. Figure 4.2 summarizes visually. This information, particularly the large standard deviation associated with each displacement, shows how widely the expression of each emotion varies from person to person. An expression with fairly low total landmark motion (e.g. disgust) expressed by someone with a tendency for enthusiastic facial movement is thus very easily confused with an expression of higher total motion (e.g. fear) in a less enthusiastic individual. However, Table 4.1 and Figure 4.2 also expose that there are certain motion properties of expressions that are more or less universal across individuals. For instance, note the large displacements of the mouth corners (corresponding to a smile) for expressions of joy or the very large displacement of the lower lip center (corresponding to an open mouth) for surprise.

4.2 Classification Accuracy

The properties of the evaluation data given above already hint that certain combinations of emotions (e.g. disgust vs. surprise) may be distinguishable with higher confidence than others (e.g. anger vs. fear). My measurements of classification accuracy confirmed this.

To evaluate classification accuracy, I trained a series of binary (two-class) SVMs for every possible pairing of the basic emotions. Since multi-class classification is essentially implemented using a cascade of binary SVMs and a voting scheme, results gathered for binary classification also give insight into the classification accuracy for multi-class classification but can be interpreted more

Landmark	*Emotion*					
	Anger	**Disgust**	**Fear**	**Joy**	**Sorrow**	**Surprise**
R. pupil	6.98/5.00	3.69/2.62	4.50/2.90	2.69/2.09	2.59/1.60	9.82/5.54
L. pupil	6.23/4.29	3.72/2.87	4.57/3.73	3.09/2.39	3.74/1.78	10.14/5.32
R. inner eyebrow	12.40/6.85	9.67/4.80	11.21/5.56	3.93/2.15	7.09/6.21	24.89/10.45
L. inner eyebrow	13.76/6.89	9.58/5.25	10.86/5.10	3.48/2.61	6.02/4.71	25.23/10.87
Nose root	7.78/5.42	5.74/5.39	6.84/4.52	4.21/2.59	4.18/1.87	13.94/7.21
Nose tip	8.77/7.09	4.99/3.64	5.90/5.03	3.16/1.94	3.79/2.88	12.92/7.74
Upper lip center	7.58/6.41	8.68/4.18	6.69/4.78	8.95/3.35	5.05/3.57	11.19/7.74
Lower lip center	9.07/5.94	9.25/4.22	8.16/3.90	7.88/3.53	7.33/4.90	52.90/14.81
R. mouth corner	7.75/4.21	8.89/3.39	11.33/5.44	22.16/3.20	12.94/4.04	21.29/15.71
L. mouth corner	8.45/4.93	8.29/2.59	11.39/4.48	22.22/3.54	12.61/3.68	18.91/14.83
L. outer eye corner	6.38/4.26	5.91/2.88	5.93/3.72	4.15/2.20	3.72/2.12	11.64/5.77
L. inner eye corner	6.98/4.82	4.03/2.55	5.74/4.16	3.49/1.74	3.27/2.10	11.06/5.85
R. inner eye corner	6.90/5.93	3.95/2.70	5.07/3.82	3.33/1.77	3.75/2.04	10.55/5.21
R. outer eye corner	8.14/5.06	4.30/2.85	5.55/3.89	3.77/2.61	3.77/2.36	10.84/4.87
R. upper lip	7.47/6.21	9.57/4.19	9.22/4.79	13.39/3.75	5.61/2.31	9.75/7.38
L. upper lip	8.37/6.37	10.04/3.55	8.64/4.84	11.67/3.97	4.90/3.53	11.15/6.57
R. lower lip	8.64/4.97	8.95/3.13	10.27/2.69	11.08/3.34	5.62/4.26	36.57/11.69
L. lower lip	10.02/5.25	9.27/4.54	10.02/4.48	9.40/4.09	7.07/4.22	38.80/12.99
R. eyebrow center	13.07/5.52	8.96/4.39	8.81/4.85	4.36/1.96	5.67/4.77	25.12/10.83
L. eyebrow center	13.62/6.56	9.43/5.12	10.04/4.45	3.96/1.64	6.16/3.95	24.82/10.14
R. nostril	8.12/5.86	6.60/3.04	6.08/4.84	4.34/2.34	3.72/2.02	11.08/7.26
L. nostril	7.62/6.25	7.15/3.04	6.47/5.13	4.24/2.44	4.02/2.05	11.06/7.33

Table 4.1: The mean (μ) / standard deviation (σ) (in pixels) for the characteristic displacements of the 22 facial landmarks for each basic emotion.

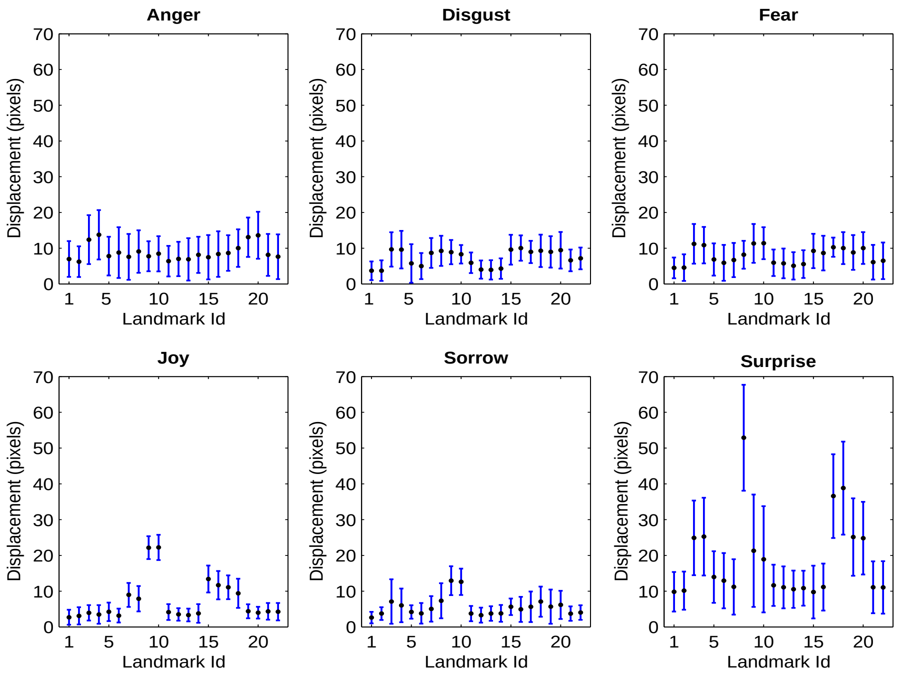


Figure 4.2: Graphs of the mean (dots) and standard deviation (bars surrounding mean) for the characteristic displacements of the 22 facial landmarks for each basic emotion.

intuitively. Because I wanted to establish a baseline measure of accuracy, I chose the simplest possible SVM formulation consisting of the standard C-SVC algorithm (with C, the penalty of a misclassified example, set to its maximum value) together with a linear kernel. In order to see how accuracy varies with the amount of training data given, I evaluated for training sets of size 1, 4, 7 and 10. Defining sets of these sizes is within the amount of effort that could reasonably expected from a user during a typical application session. Table 4.2 shows the results.

Several conclusions can be drawn from these results:

1. Some combinations of emotions are indeed harder to distinguish than others.
2. While bigger training sets do result in better accuracy generally, adding more training data does not necessarily always increase accuracy (for instance, note the drop in accuracy for 'fear' from 78.3% to 76.2% between the second and third trial).
3. Indeed, increasing training set size beyond a certain point may even bring down total accuracy. Smaller sets with "high quality" examples (i.e. very pronounced and distinct expressions for each emotion) are often better than very large sets of lesser "quality".
4. The SVM classifier quickly reaches a mean total accuracy of around 86% as training set size grows. Once this value is reached, increasing training set size further has only minor effect on the accuracy. This is in accordance with the widely stated SVM property of performing well even when only little training data is available.

I found the C-SVC algorithm to consistently outperform the nu-SVC formulation for various values of ν and C. The parameter ν to the nu-SVC algorithm controls the number of support vectors, while the parameter C to the C-SVC algorithm gives the penalty of misclassifying an example. The latter is a more natural way to describe many problems. While Chang & Lin [5] posit that the two algorithms are essentially different problems with the same solution set, C-SVC has been used much more widely, in part because efficient methods of solving large scale nu-SVC problems are not yet known.

4.3 Choice of Kernel

As stated in Section 2.4, the selection of an appropriate kernel function is the most important choice to be made when customizing an SVM classifier to a particular application domain. Figure 4.3 shows Matlab plots illustrating pictorially how some of the more common kernel choices affect the decision boundary for a 2D problem that is not linearly separable.

To evaluate the effect of different kernels on classification accuracy, the C-SVC algorithm was used on a training set of 7 examples per basic emotion. I considered linear, polynomial, radial basis function and sigmoid kernels. Table 4.3 shows the results, per kernel, in terms of accuracy for each basic emotion and overall accuracy. Whenever I could customize a particular kernel (e.g. changing the degree of the polynomial kernel or the width of the radial basis function), I chose the most basic, default setting for each parameter.

Surprisingly, the linear kernel performed best during these measurements. The sigmoid kernel showed poor performance for all scales and offsets I tried. It has not gained widespread use in other classification systems. The performance of both the polynomial and the RBF kernels is very close to that of the linear kernel, even for the default degree and width parameters. This, together

$n = 1$	**Anger**	**Disgust**	**Fear**	**Joy**	**Sorrow**	**Surprise**	*Overall*
Anger	—	61.5%	47.8%	85.1%	68.2%	93.6%	71.2%
Disgust	61.5%	—	58.0%	92.2%	68.8%	94.1%	75.0%
Fear	47.8%	58.0%	—	57.8%	52.4%	91.1%	61.4%
Joy	85.1%	92.2%	57.8%	—	88.4%	93.5%	83.6%
Sorrow	68.2%	68.8%	52.4%	88.4%	—	93.0%	74.1%
Surprise	93.6%	94.1%	91.1%	93.5%	93.0%	—	93.1%

Total accuracy: 76.4%

$n = 4$	**Anger**	**Disgust**	**Fear**	**Joy**	**Sorrow**	**Surprise**	*Overall*
Anger	—	56.5%	55.0%	97.6%	81.6%	97.6%	77.2%
Disgust	56.5%	—	65.9%	95.6%	88.1%	100.0%	81.1%
Fear	55.0%	65.9%	—	94.9%	77.8%	100.0%	78.3%
Joy	97.6%	95.6%	94.9%	—	89.2%	97.5%	95.0%
Sorrow	81.6%	88.1%	77.8%	89.2%	—	100.0%	87.4%
Surprise	97.6%	100.0%	100.0%	97.5%	100.0%	—	99.0%

Total accuracy: 86.2%

$n = 7$	**Anger**	**Disgust**	**Fear**	**Joy**	**Sorrow**	**Surprise**	*Overall*
Anger	—	65.0%	50.0%	100.0%	81.3%	97.1%	78.4%
Disgust	65.0%	—	76.3%	94.9%	83.3%	100.0%	83.9%
Fear	50.0%	76.3%	—	87.9%	66.7%	100.0%	76.2%
Joy	100.0%	94.9%	87.9%	—	96.8%	97.1%	95.3%
Sorrow	81.3%	83.3%	66.7%	96.8%	—	100.0%	85.6%
Surprise	97.1%	100.0%	100.0%	97.1%	100.0%	—	98.8%

Total accuracy: 86.3%

$n = 10$	**Anger**	**Disgust**	**Fear**	**Joy**	**Sorrow**	**Surprise**	*Overall*
Anger	—	70.6%	53.6%	100.0%	88.5%	100.0%	82.2%
Disgust	70.6%	—	71.9%	97.0%	83.3%	100.0%	84.6%
Fear	53.6%	71.9%	—	74.1%	58.3%	100.0%	71.7%
Joy	100.0%	97.0%	74.1%	—	96.0%	96.4%	93.0%
Sorrow	88.5%	83.3%	58.3%	96.0%	—	100.0%	85.4%
Surprise	97.1%	100.0%	100.0%	96.4%	100.0%	—	99.3%

Total accuracy: 86.0%

Table 4.2: Binary classification accuracy for each pairing of basic emotions. Size of training set (in examples per class) is given by *n*. Rightmost column gives overall mean accuracy for each row's emotion.

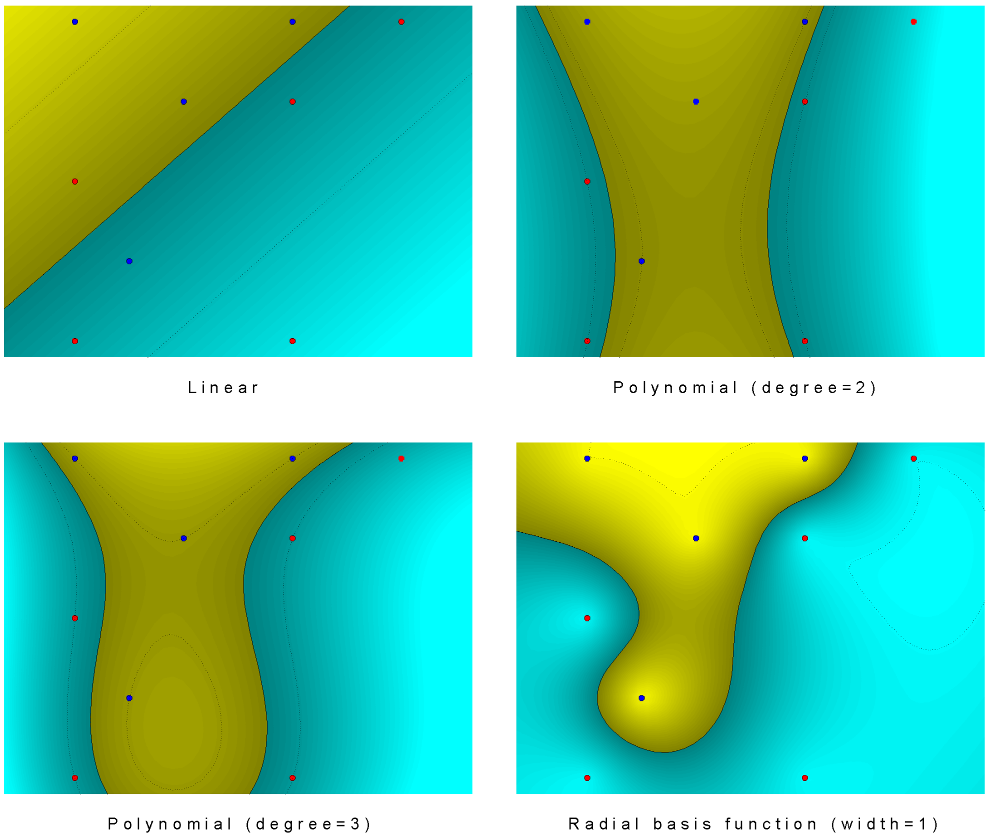


Figure 4.3: A 2D classification problem that is not linearly separable. Kernel choice affects the shape of the decision boundary.

	Anger	Disgust	Fear	Joy	Sorrow	Surprise	*Overall*
Linear	78.4%	83.9%	76.2%	95.3%	85.6%	98.8%	86.3%
Polynomial (degree=2)	81.8%	83.3%	76.2%	91.9%	83.8%	88.4%	84.2%
RBF (width=1)	80.7%	82.3%	73.8%	93.0%	86.9%	93.6%	85.0%
Sigmoid (scale=1, offset=0)	48.3%	44.3%	47.6%	47.1%	43.8%	49.4%	46.7%

Table 4.3: Effect of kernel choice on classification accuracy. Columns give recognition accuracy for each basic emotion. Rightmost column gives overall accuracy for each kernel.

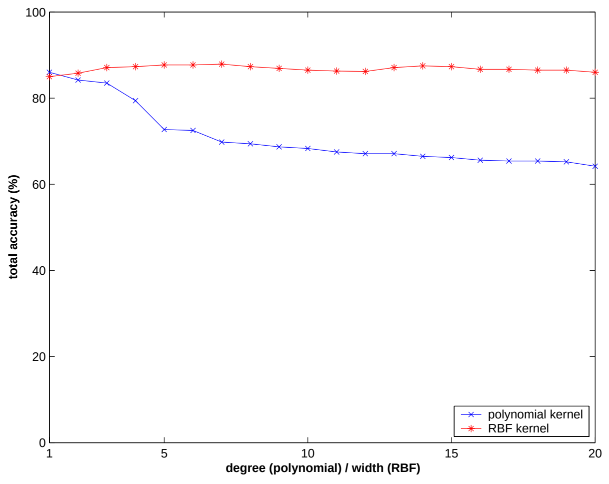


Figure 4.4: Total accuracy for the polynomial and RBF kernels as the parameters *degree* and *width*, respectively, are varied.

with the fact that Joachims [17] and others report polynomial kernels achieving highest accuracy for a variety of applications, led to experimenting with the parameters of the polynomial and RBF kernels. Figure 4.4 shows how total accuracy over all basic emotions varies for these kernels as the parameters are adjusted. The polynomial kernel was not able to outperform the linear kernel for any degree. However, the RBF kernel did outperform the linear kernel, achieving a peak accuracy of 87.9% for a width parameter value of 7. This was the highest overall recognition accuracy achievable for the displacement / SVM approach on the evaluation data set.

4.4 Comparison to Other Systems and Human Ceiling

The optimal accuracy of 87.9% achieved by the combined landmark displacement / SVM approach using an RBF kernel compares favorably to other expression classification systems described in the literature.

Dumas [11] reports classification accuracy of up to 85.7% using a Gabor filter preprocessing/SVM combination, while Dailey et al. [9] use Gabor filter preprocessing together with neural networks to achieve 85.9%. Pantic & Rothkrantz [28] use an elaborate system based on hybrid frontal/sideways views of the face to achieve up to 91% accuracy. None of these systems do classification in video.

For preprocessed, Gabor-filtered video, Littlewort et al. [23] report between 87% and 91% accuracy using an SVM classifier. Sebe et al. [30] use a Cauchy Naïve Bayes classifier to achieve 80% person dependent and 63% person-independent accuracy in video.

A series of trials conducted by Ekman & Friesen [14] established the human ‘ceiling’ in classifying facial expressions into the six basic emotions at 91.7%.

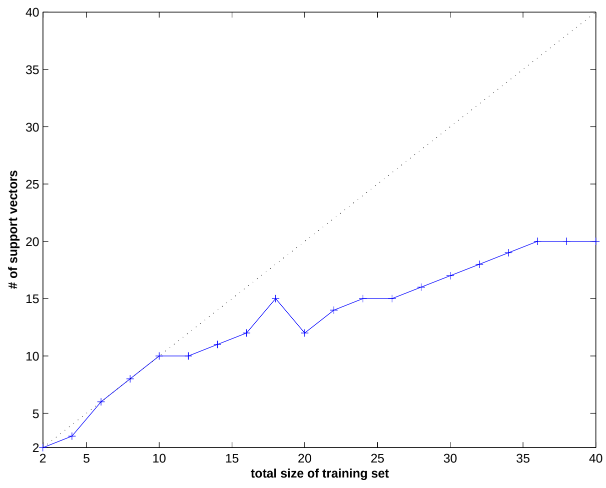


Figure 4.5: Number of support vectors defining the decision surface for anger vs. joy as the training set size is increased.

4.5 Speed

In hindsight, my initial concerns about the performance of a Java-based SVM classifier proved to be somewhat unfounded.

All examples used in both the image and video-based applications are fixed vectors of 22 feature displacements. Unless the amount of training examples used does get very large (in the 100s or 1000s — very unlikely given that each example is individually defined by the user during interaction), there is little difference in the time taken to train the SVM.

Running a profiler[1] on the image-based application showed that initial SVM training takes between 200–300ms for any typically sized training set (on the order of 5–20 examples per class). Subsequent training runs then take significantly less time (around 10–20ms) due to JIT-compiled cached code being executed. Classification of an unseen example (a simple evaluation of the decision function established by training) took under 1ms.

For the video application, profiling results[2] showed that for compiled code, SVM training takes around 9μs, remaining at roughly that level for any reasonably sized training set. Subsequent classification occurs in around 11μs (close to 12,000 clock cycles). This very short time easily allows for classification to be done in real time once per frame of video, around 30 times per second. The classification & result reporting thus appears instantaneous to the user.

The high efficiency partially stems from the fact that the decision function the SVM algorithm produces is dependent only on the support vectors, as outlined in Section 2.4. Figure 4.5 depicts how the number of support vectors grows as the size of the training set is increased. It shows the support vectors of the decision surface for one of the binary classification problems evaluated previously, namely anger vs. joy. It can be seen how the number of support vectors which define the decision surface increase sub-linearly as the training set size is increased. In practical terms, if there is a lot of redundancy in the training set, the SVM classifier picks out those training

[1]EclipseColorer (`http://eclipsecolorer.sourceforge.net`), a plugin to IBM's Eclipse IDE.

[2]Using DevPartner Profiler, a plugin to Visual Studio.

examples that are truly characteristic for each class. The remaining, redundant examples are then simply ignored and do not hamper training or classification performance.

4.6 Video-based Classification Results

I wanted to quantify the penalty in recognition accuracy obviously incurred when moving to video-based classification. I also wanted to ascertain the suitability of the combined SVM / landmark displacements approach for 'ad-hoc' classification, whereby an inexperienced user is confronted with the application and asked to express the basic emotions naturally, leading to greater variability than is present in the CMU dataset used above, where actors were asked to essentially give their interpretation of characteristic expression of each basic emotion (e.g. an open mouth and raised eyebrows for 'surprise'). Finally, I wanted to see whether it is possible to use training data from one person to classify another person's expressions, thus performing person-independent classification. This would allow users to skip the training stage altogether by simply using someone else's training data.

Three kinds of trials were carried out with the video-based application:

1. To establish the optimum recognition accuracy, an 'expert' user thoroughly familiar with the application[3] supplied sets of training data with 1, 4 and 10 examples per basic emotion which was subsequently used to classify 12×6 unseen expressions in 'snapshot' mode.
2. To establish the recognition accuracy for ad-hoc classification, six users unfamiliar with the application and automated expression analysis in general were asked to provide one training example per basic emotion and subsequently supply a number of unseen examples for classification.
3. To establish the recognition accuracy for person-independent classification, examples from the two expert users were imported into the application and used each time by the expert user that did not provide them as training data to classify 12×6 unseen examples.

Figure 4.6 shows a typical video-based classification session and depicts the experimental setup used during these trials.

4.6.1 Expert Classification

Table 4.4 gives the results for the first of these trials. It shows that for an expert user who performs all expressions enthusiastically and consistently with minimal head motion, the recognition accuracy is very similar to the results presented earlier in this chapter in Table 4.2. The higher accuracy for the smallest (n=1) training set in Table 4.4 can be attributed to the fact that all six examples here were from the same person.

4.6.2 Ad-hoc Classification

Table 4.5 gives the results for ad-hoc classification. As can be seen, the accuracy achieved is significantly lower. Apart from the intentionally more unconstrained setup of this trial in terms

[3]Either myself or my supervisor.

$n = 1$	**Anger**	**Disgust**	**Fear**	**Joy**	**Sorrow**	**Surprise**	*Overall*
Anger	6	0	3	0	3	0	50.0%
Disgust	0	12	0	0	0	0	100.0%
Fear	2	0	10	0	0	0	83.3%
Joy	0	1	0	8	3	0	66.6%
Sorrow	0	0	3	0	9	0	75.0%
Surprise	0	0	0	0	0	12	100.0%

Total accuracy: 79.2%

$n = 4$	**Anger**	**Disgust**	**Fear**	**Joy**	**Sorrow**	**Surprise**	*Overall*
Anger	10	0	0	0	0	2	83.3%
Disgust	0	12	0	0	0	0	100.0%
Fear	3	0	9	0	0	0	75.0%
Joy	0	0	0	10	0	2	83.3%
Sorrow	0	0	3	0	9	0	75.0%
Surprise	0	0	0	0	0	12	100.0%

Total accuracy: 86.1%

$n = 10$	**Anger**	**Disgust**	**Fear**	**Joy**	**Sorrow**	**Surprise**	*Overall*
Anger	10	0	0	0	0	2	83.3%
Disgust	0	12	0	0	0	0	100.0%
Fear	2	0	10	0	0	0	83.3%
Joy	0	0	0	9	0	3	75.0%
Sorrow	0	0	2	0	10	0	83.3%
Surprise	0	0	0	0	0	12	100.0%

Total accuracy: 87.5%

Table 4.4: Confusion matrices for expert video classification. Each cell gives the number of examples of the row's emotion classified as the column's emotion. Size of training set (examples per basic emotion) is given by *n*. Rightmost column gives overall mean accuracy for each row's emotion.

$n = 1$	**Anger**	**Disgust**	**Fear**	**Joy**	**Sorrow**	**Surprise**	*Overall*
Anger	19	4	3	1	3	2	59.4%
Disgust	2	18	1	2	5	3	58.1%
Fear	7	3	15	0	3	1	51.7%
Joy	1	5	1	21	4	1	63.6%
Sorrow	2	3	7	3	18	0	54.4%
Surprise	2	3	1	0	2	25	75.8%

Total accuracy: 60.7%

Table 4.5: Confusion matrix for ad-hoc classification performed by multiple subjects. One training example per basic emotion per person. Rightmost column gives overall mean accuracy for each row's emotion.

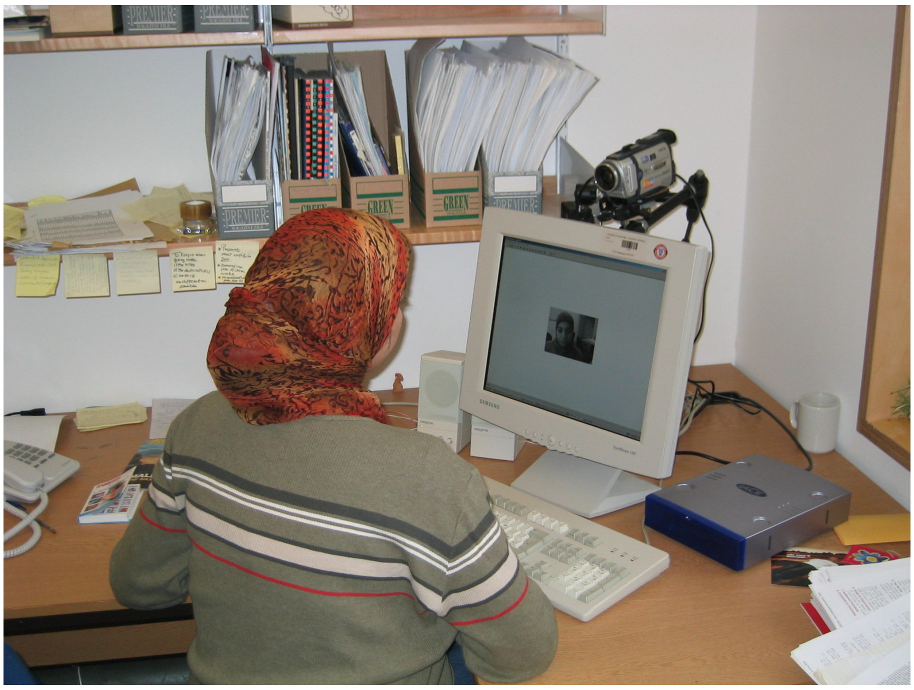


Figure 4.6: Picture of a typical interactive session during evaluation of video-based classification.

of distance to the camera, posture, etc., this can be attributed in great part to head motion by the subjects giving inaccurate landmark displacements and the fact that subjects found it hard to enthusiastically portray each expression distinctly on command. Fear in particular was quoted as being difficult to consciously express upon request.

4.6.3 Person-independent Classification

Table 4.6 gives the results for person-independent classification. It can be seen that those emotions having expressions that are generally very uniform across subjects (such as the universal smile of a 'joy' expression or the open mouth and raised eyebrows for 'surprise') exhibit the highest recognition rates. The trials showed that the more expressive the person supplying the training examples, the better the resulting accuracy in classification. It was also significant that both the supplier of training examples and the tester were aware of the characteristic expressions of each basic emotion and that there was hence fairly high overall concurrency between the expressions of the two individuals. Further short trials taking examples from the ad-hoc training data (where overall concurrency of expression was much less pronounced) showed that while person-independent recognition rates are high when training and classification is done in a carefully specified manner, just taking anyone's training examples to classify someone else's expressions in an ad-hoc manner introduces too much noise and succeeds only when the trainer's expressions are essentially imitated.

	Anger	**Disgust**	**Fear**	**Joy**	**Sorrow**	**Surprise**	*Overall*
Anger	8	0	2	0	2	0	66.7%
Disgust	3	9	2	0	0	0	64.3%
Fear	2	0	8	0	2	0	66.7%
Joy	0	1	0	11	0	0	91.7%
Sorrow	0	2	4	0	10	0	62.5%
Surprise	2	0	0	0	0	10	83.3%

Total accuracy: 71.8%

Table 4.6: Confusion matrix for person-independent classification. Two examples per basic emotion. Rightmost column gives overall mean accuracy for each row's emotion.

4.7 Testing

4.7.1 Module Testing

The object-oriented structure of both applications was of great benefit for early module testing. Where possible, I attempted to test each class in isolation as I implemented it. This turned out to be somewhat easier for the Java application than for the video-based application.

Image-based application

For the image-based application, the GUI oriented classes (e.g. `TrainingClassFrame` or `QueryFrame`) could only be properly tested during system testing after they were integrated in the overall application. However, `SVMCore` and the classes implementing data structures were tested early.

To test `SVMCore`, I rewrote a simple application to classify small GIF images as 'yellow' or 'blue' according to pixel color content I had written during the preparation phase to make use of the class through the *`Classifier`* interface for all training and classification.

To test the data structures used to represent landmarks and displacement-based examples, I wrote a separate small Java application (`FeaturePicker`) to perform manual landmark definition on an image in the same way it should be done within the final application. Once I was sure landmark definition and the associated data structures were working correctly, it was easy to integrate the functionality of this program into the final application.

Video-based application

Due to the close integration of the application components components into the Document/View structure, module testing could only be easily carried out for `SVMCore`. I modified small sample programs shipped with the C++ version of `libsvm` to use `SVMCore` for training and classification of sample heart-rate data (which is also provided with `libsvm`). This proved to be useful particularly to ensure that memory allocation / deallocation and pointer handling were working properly.

4.7.2 System Testing

The gathering of the results presented in this chapter provided ample opportunity to thoroughly perform system testing for both the image and video based applications.

Image-based application

All training and testing examples used in evaluating classification accuracy above were defined in the application and subsequently used for training and classification. The ability to cope with many training examples in different classes as well as numerous subsequent training/classification runs with different SVM parameters was thus implicitly tested during evaluation. Being able to reuse work from previous sessions as well as add correctly classified examples to their target classes saved me a lot of time.

Video-based application

Again, the user trials carried out during evaluation of accuracy enabled be to thoroughly stress-test the functionality of the application. I found and was able to fix various memory leaks associated with improper memory de- and reallocation for the bitmap data used to display the video stream, which resulted in video not being drawn after some minutes of operation. A memory/pointer bug in reconstructing `SVMCore` objects after loading a previously exported SVM representation during serialization was also found and fixed while evaluating person-independent classification in video.

Chapter 5

Conclusions

The prime motivation behind this project was to apply Support Vector Machine learning to the problem of automated emotion classification in facial expressions. All aims outlined in both Chapter 1 as well as the project proposal have been met. A number of what were originally project extensions in the proposal, including real time classification in video, were also implemented successfully.

5.1 Achievements

An application to perform SVM-based emotion classification from facial images satisfying the requirements of Section 2.1.1 was designed, implemented and tested successfully. It allows for flexible evaluation of SVM classification of facial expressions for an arbitrary set of user-defined emotion categories and various SVM settings. Its implementation can be extended in a straightforward way to incorporate different methods of feature extraction as well as different classifiers.

A number of approaches to feature extraction from face images were explored, including Gabor wavelets, eigenfaces and landmark displacements.

The combination of feature extraction using landmark displacements and SVM classification yielded high classification accuracy and low training & classification delays on image-based data, causing investigation of the same approach to emotion classification in video.

An interactive video-based expression recognition application satisfying the requirements of Section 2.1.2 was thus designed, implemented and tested successfully. It allows for spontaneous expressions to be SVM classified in real time. It enables training data to be conveniently supplied incrementally and reused across sessions, allows experimentation with different SVM formulations and runs on commodity hardware.

To evaluate the suitability of the combined landmark displacements / SVM classification approach adopted in this project, a number of experiments were carried out. The accuracy in classifying the basic emotions of anger, disgust, fear, joy, sorrow and surprise on images was evaluated for various training & testing data and different SVM parameters. Human trials were carried out for different interaction scenarios to evaluate classification accuracy in live video.

The performance of both image-based and video-based classification was evaluated in terms of training and classification delay and scalability as the amount of training data increases.

5.2 Publications arising from Project Work

Combining landmark displacements extracted from live video with SVM classification presents a novel and promising approach to automatic and unobtrusive expression classification in real time. The results presented in Chapter 4 compare favorably with previous approaches to expression recognition, having in addition the complete lack of preprocessing and the accuracy achieved even for very unconstrained ad-hoc interaction as two important advantages.

A poster arising from work presented in this dissertation will be presented at the 10th International Conference on Human-Computer Interaction in June 2003, with an abstract being published in the conference proceedings.

A paper I co-authored together with my supervisor has been submitted to the Fifth International Conference on Multimodal Interfaces in November 2003. It is reproduced in Appendix A.

5.3 Hindsight and Further Work

If I was to start this project again, I would be rather more optimistic about the the performance of SVM classification. My fears of execution speed of the SVM routines being inadequate especially for video were unconfirmed.

I would have perhaps focussed more on video-based classification from the outset. This would have enabled me to pursue the following particularly interesting further improvements:

- **Better compensation for head motion:** The current implementation adjusts landmark displacements to deal with head motion in video by normalizing with respect to the motion of the nose root. This obviously deals poorly with rotational head motion (pitch, roll, yaw) resulting from nodding or head tilting. A more sophisticated scheme could be implemented. This would very likely boost classification accuracy further.

- **Automatic SVM model selection:** There are methods of automatically choosing the optimal kernel function and SVM parameters for given training data (see for instance [6]). Again, this is likely to improve accuracy.

- **More feature extraction methods for still images:** My original plan to extend the image-based application to use different extraction methods (such as Gabor filtering) was adjusted to focus work on video-based classification. It would be interesting to implement additional feature extraction methods and see how they compare to the displacements approach on images.

5.4 Acknowledgements

I am indebted to Rana El Kaliouby for her help, supervision and feedback throughout this project and while authoring this dissertation and the publications arising from this project.

I would also like to thank the Computer Lab members that took part in the video trials during evaluation.

Bibliography

[1] B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, 1992.

[2] M. Brown, W. Grundy, D. Lin, N. Cristianini, C. Sugnet, T. Furey, M. Ares, and D. Haussler. Knowledge-based analysis of microarray gene expression data using support vector machines. Technical report, University of California in Santa Cruz, 1999.

[3] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[4] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[5] C.-C. Chang and C.-J. Lin. Training ν-support vector classifiers: Theory and algorithms. *Neural Computation*, 13(9):2119–2147, 2001.

[6] O. Chapelle and V. Vapnik. Model selection for support vector machines. In S. Solla, T. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12, pages 230–236. MIT Press, Cambridge, MA, USA, 2000.

[7] J. F. Cohn, A. J. Zlochower, J. J. Lien, and T. Kanade. Automated face analysis by feature point tracking has high concurrent validity with manual faces coding. *Psychophysiology*, 36:35–43, 1999.

[8] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other Kernel-based Learning Methods*. Cambridge University Press, Cambridge, UK, 2000.

[9] M. N. Dailey, G. W. Cottrell, and R. Adolphs. A six-unit network is all you need to discover happiness. In *Proceedings of the Twenty-Second Annual Conference of the Cognitive Science Society*, Mahwah, NJ, USA, 2000. Erlbaum.

[10] C. Darwin. *The Expression of the Emotions in Man and Animals*. John Murray, London, UK, 1872.

[11] M. Dumas. Emotional expression recognition using support vector machines. 2001.

[12] P. Ekman. *Emotion in the Human Face*. Cambridge University Press, Cambridge, UK, 1982.

[13] P. Ekman. Basic emotions. In T. Dalgleish and T. Power, editors, *The Handbook of Cognition and Emotion*. John Wiley & Sons, Ltd., 1999.

[14] P. Ekman and W. Friesen. *Pictures of Facial Affect*. Consulting Psychologists Press, Palo Alto, CA, USA, 1976.

[15] P. Ekman and W. Friesen. *Facial Action Coding System (FACS): Manual*. Consulting Psychologists Press, Palo Alto, CA, USA, 1978.

[16] T. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In M. Kearns, S. Solla, and D. Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11. MIT Press, Cambridge, MA, USA, 1998.

[17] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 137–142, Heidelberg, DE, 1998. Springer Verlag.

[18] J. Jones and L. Palmer. An evaluation of the two-dimensional gabor filter model of simple receptive fields in cat striate cortex. *Journal of Neurophysiology*, 58(6):1233–1258, 1987.

[19] T. Kanade, J. Cohn, and Y. Tian. Comprehensive database for facial expression analysis. In *Proceedings of the 4th IEEE International Conference on Automatic Face and Gesture Recognition (FG'00)*, pages 46–53, 2000.

[20] A. Kapoor. Automatic facial action analysis. Master's thesis, Massachusetts Institute of Technology, 2002.

[21] M. Lades, J. C. Vorbrüggen, J. Buhmann, J. Lange, C. von der Malsburg, R. P. Würtz, and W. Konen. Distortion invariant object recognition in the dynamic link architecture. *IEEE Transactions on Computers*, 42:300–311, 1993.

[22] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Müller, E. Säckinger, P. Simard, and V. Vapnik. Comparison of learning algorithms for handwritten digit recognition. In F. Fogelman-Soulié and P. Gallinari, editors, *Proceedings ICANN'95 - International Conference on Artificial Neural Networks*, volume 2, pages 53–60, EC2, 1995.

[23] G. Littlewort, I. Fasel, M. Stewart Bartlett, and J. R. Movellan. Fully automatic coding of basic expressions from video. Technical Report 2002.03, UCSD INC MPLab, 2002.

[24] M. Loy, R. Eckstein, D. Wood, J. Elliott, and B. Cole. *Java Swing, 2nd Edition*. O'Reilly & Associates, Inc., 2002.

[25] A. Mehrabian. Communication without words. *Psychology Today*, 2(4):53–56, 1968.

[26] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *Proceedings of Computer Vision and Pattern Recognition '97*, pages 130–36, 1997.

[27] M. Pantic and L. J. M. Rothkrantz. Automatic analysis of facial expressions: The state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1424–1445, 2000.

[28] M. Pantic and L. J. M. Rothkrantz. An expert system for recognition of facial actions and their intensity. In *AAAI/IAAI*, pages 1026–1033, 2000.

[29] B. Reeves and C. Nass. *The Media Equation: How People treat Computers, Television, and New Media like Real People and Places*. Cambridge University Press, New York, NY, USA, 1996.

[30] N. Sebe, I. Cohen, A. Garg, M. Lew, and T. S. Huang. Emotion recognition using a cauchy naive bayes classifier. In *Proceedings of the 16th International Conference on Pattern Recognition (ICPR)*, 2002.

[31] B. Stroustrup. *The C++ programming language (2nd ed.)*. Addison-Wesley Longman Publishing Co., Inc., 1991.

[32] Y.-L. Tian, T. Kanade, and J. Cohn. Evaluation of gabor-wavelet-based facial action unit recognition in image sequences of increasing complexity. In *Proceedings of the Fifth IEEE International Conference on Automatic Face and Gesture Recognition*, pages 229 – 234, 2002.

[33] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[34] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[35] V. N. Vapnik. *Statistical Learning Theory*. Wiley, New York, NY, USA, 1998.

Appendix A

Paper arising from Project Work

Starts overleaf.

Real Time Facial Expression Recognition in Video using Support Vector Machines

Philipp Michel
Computer Laboratory
University of Cambridge
Cambridge CB3 0FD, United Kingdom
pmichel@cantab.net

Rana El Kaliouby
Computer Laboratory
University of Cambridge
Cambridge CB3 0FD, United Kingdom
rana.el-kaliouby@cl.cam.ac.uk

ABSTRACT

Enabling computer systems to recognize facial expressions and infer emotions from them in real time presents a challenging research topic. In this paper, we present a real time approach to emotion recognition through facial expression in live video. We employ an automatic facial feature tracker to perform face localization and feature extraction. The facial feature displacements in the video stream are used as input to a Support Vector Machine classifier. We evaluate our method in terms of recognition accuracy for a variety of interaction and classification scenarios. Our person-dependent and person-independent experiments demonstrate the effectiveness of a support vector machine and feature tracking approach to fully automatic, unobtrusive expression recognition in live video. We conclude by discussing the relevance of our work to affective and intelligent man-machine interfaces and exploring further improvements.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces; H.1.2 [**Models and Principles**]: User/Machine Systems—*Human factors*; I.5.4 [**Pattern Recognition**]: Applications; I.4.8 [**Image Processing and Computer Vision**]: Scene Analysis—*Tracking*.

General Terms

Design, Experimentation, Performance, Human Factors.

Keywords

Facial expression analysis, support vector machines, feature tracking, emotion classification, affective user interfaces.

1. INTRODUCTION

The human face possesses superior expressive ability [8] and provides one of the most powerful, versatile and natural means of communicating motivational and affective state [9]. We use facial expressions not only to express our emotions, but also to provide important communicative cues during social interaction, such as our level of interest, our desire to take a speaking turn and continuous feedback signalling understanding of the information conveyed. Facial expression constitutes 55 percent of the effect of a communicated message [18] and is hence a major modality in human communication.

Reeves & Nass [21] posit that human beings are biased to attempt to evoke their highly evolved social skills when confronting a new technology capable of social interaction. The possibility of enabling systems to recognize and make use of the information conferred by facial expressions has hence gained significant research interest over the last few years. This has given rise to a number of automatic methods to recognize facial expressions in images or video [20].

In this paper, we propose a method for automatically inferring emotions by recognizing facial expressions in live video. We base our method on the machine learning system of Support Vector Machines (SVMs). A face feature tracker gathers a set of displacements from feature motion in the video stream. These are subsequently used to train an SVM classifier to recognize previously unseen expressions. The novelty of our approach consists of its performance for real time classification, its unobtrusiveness and its lack of preprocessing. It is particularly suitable for ad-hoc, incrementally trained and person-independent expression recognition.

The remainder of this paper is organized as follows. Section 2 gives an overview of some prior work in the area of facial expression analysis. In Section 3 we outline the overall design of our approach to expression recognition. Section 4 describes how face localization, feature extraction and tracking are accomplished, while Section 5 gives an overview of support vector machines and details how they are used in our approach. We evaluate our system in Section 6 and conclude in Section 7.

2. RELATED WORK

Pantic & Rothkrantz [20] identify three basic problems a facial expression analysis approach needs to deal with: face detection in a facial image or image sequence, facial expression data extraction and facial expression classification.

Most previous systems assume presence of a full frontal face view in the image or the image sequence being analyzed, yielding some knowledge of the global face location. To give the exact location of the face, Viola & Jones [27] use the Adaboost algorithm to exhaustively pass a search sub-window over the image at multiple scales for rapid face detection. Essa & Pentland [13] perform spatial and temporal filtering together with thresholding to extract motion blobs from image sequences. To detect presence of a face, these blobs are then evaluated using the eigenfaces method [24] via principal component analysis (PCA) to calculate the distance of the observed region from a face space of 128 sample images. The PersonSpotter system [22] tracks the bounding box of the head in video using spatio-temporal filtering and stereo disparity in pixels due to motion, thus selecting image regions of interest. It then employs skin color and convex region detectors to check for face presence in these regions.

To perform data extraction, Littlewort et al. [17] use a bank of 40 Gabor wavelet filters at different scales and orientations to perform convolution. They thus extract a "jet" of magnitudes of complex valued responses at different locations in a lattice imposed on an image, as proposed in [16]. Essa & Pentland [13] extend their face detection approach to extract the positions of prominent facial features using eigenfeatures and PCA by calculating the distance of an image from a feature space given a set of sample images via FFT and a local energy computation. Cohn et al. [4] first manually localize feature points in the first frame of an image sequence and then use hierarchical optical flow to track the motion of small windows surrounding these points across frames. The displacement vectors for each landmark between the initial and the peak frame represent the extracted expression information.

In the final step of expression analysis, expressions are classified according to some scheme. The most prevalent approaches are based on the existence of six basic emotions (anger, disgust, fear, joy, sorrow and surprise) as argued by Ekman [10] and the Facial Action Coding System (FACS), developed by Ekman and Friesen [12], which codes expressions as a combination of 44 facial movements called Action Units. While much progress has been made in automatically classifying according to FACS [23], a fully automated FACS-based approach for video has yet to be developed. Dailey et al. [7] use a six unit, single layer neural network to classify into the six basic emotion categories given Gabor jets extracted from static images. Essa & Pentland [13] calculate ideal motion energy templates for each expression category and take the euclidean norm of the difference between the observed motion energy in a sequence of images and each motion energy template as a similarity metric. Littlewort et al. [17] preprocess image sequences image-by-image to train two stages of support vector machines from Gabor filter jets. Cohn et al. [4] apply separate discriminant functions and variance-covariance matrices to different facial regions and use feature displacements as predictors for classification.

3. IMPLEMENTATION OVERVIEW

We use a real time facial feature tracker to deal with the problems of face localization and feature extraction in spontaneous expressions. The tracker extracts the position of 22 facial features from the video stream. We calculate displacements for each feature between a neutral and a representative frame of an expression. These are used together with the label of the expression as input to the training stage of an SVM classifier. The trained SVM model is subsequently used to classify unseen feature displacements in real time, either upon user request or continuously, for every frame in

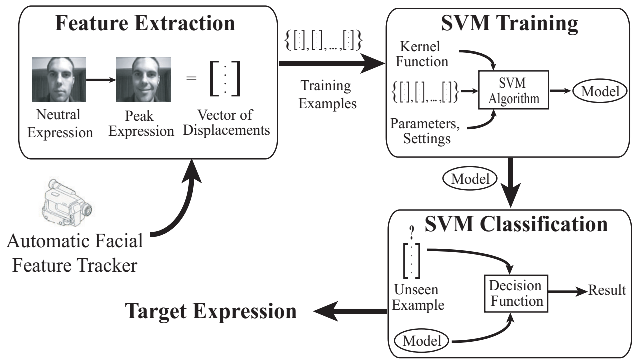


Figure 1: The stages of SVM-based automated expression recognition.

the video stream. Figure 1 illustrates the implementation diagrammatically.

Commodity hardware is used, such as a commercial digital camcorder connected to a standard PC, allowing classification at 30 frames per second. We have found our approach to perform well even at significantly lower frame rates (and hence more intermittent motion) and under a variety of lighting conditions. We assume a full frontal view of the face, but take into account the user-dependent variations in head pose and framing inherent in video-based interaction. Our approach works for arbitrary user-defined emotion classes. For illustration purposes, we use the universal basic emotions throughout the remainder of this paper.

4. FEATURE EXTRACTION & TRACKING

We chose the feature displacement approach due to its suitability for a real time video system, in which motion is inherent and which places a strict upper bound on the computational complexity of methods used in order to meet time constraints. Cohn et al. [5] posit that approaches based on feature point tracking show 92% average agreement with manual FACS coding by professionals and are hence highly applicable to expression analysis. Our earlier experiments using an SVM classifier on displacements of manually defined facial landmarks in image sequences from the Cohn-Kanade facial expression database [15] yielded high classification accuracy. Furthermore, they exhibited short training and classification delays which prompted us to investigate the application of a combined feature displacement / SVM approach for real time video.

The tracker we employ uses a face template to initially locate the position of the 22 facial features of our face model in the video stream and uses a filter to track their position over subsequent frames as shown in Figure 2. For each expression, a vector of feature displacements is calculated by taking the euclidean distance between feature locations in a neutral and a "peak" frame representative of the expression, as illustrated in Figure 3. This allows characteristic feature motion patterns to be established for each expression, as given by Figure 4. Feature locations are automatically captured when the amount of motion is at a minimum, corresponding to either the initial neutral phase or the final phase of a spontaneous expression, when motion has settled around its peak frame.

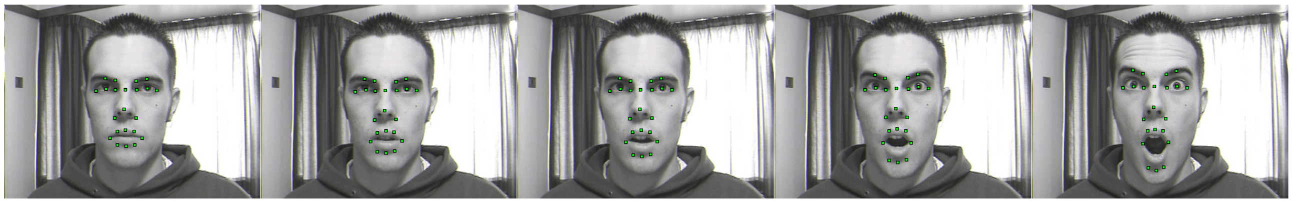


Figure 2: Facial features localized and tracked over a sequence of video frames (taken at intervals in a 30fps stream).

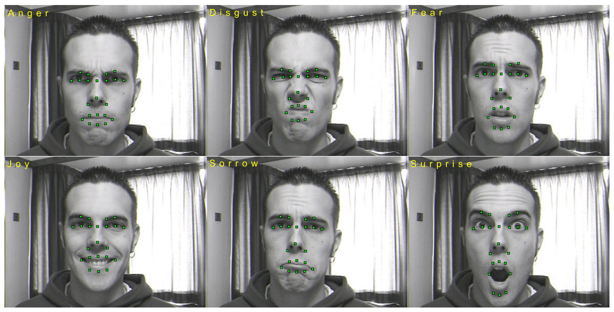


Figure 3: Peak frames for each of the six basic emotions, with features localized.

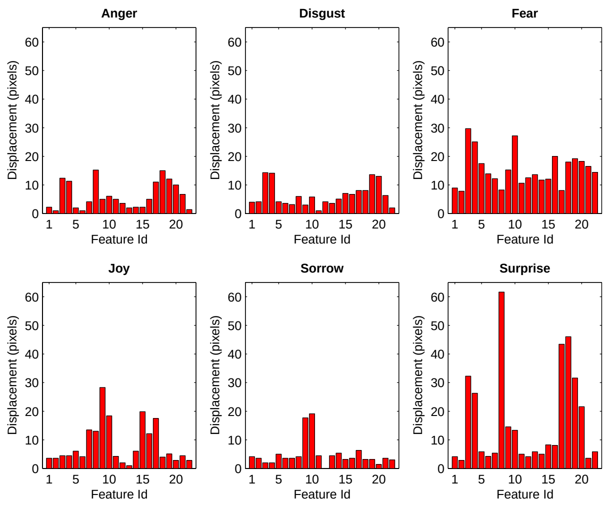


Figure 4: Feature motion patterns for the six basic emotions.

5. CLASSIFICATION

5.1 Support Vector Machine Overview

Machine learning algorithms receive input data during a training phase, build a model of the input and output a hypothesis function that can be used to predict future data. Given a set of labelled training examples

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)), y_i \in \{-1, 1\}$$

learning systems typically try to find a decision function of the form

$$h(\mathbf{x}) = sgn(\langle \mathbf{w} \cdot \mathbf{x} \rangle + b)$$

that yields a label $\in \{-1, 1\}$ (for the basic case of binary classification) for a previously unseen example $\mathbf{x}$.

Support Vector Machines [1, 6] are based on results from statistical learning theory, pioneered by Vapnik [25, 26], instead of heuristics or analogies with natural learning systems. These results establish that the generalization performance of a learned function on future unseen data depends on the complexity of the class of functions it is chosen from rather than the complexity of the function itself. By bounding this class complexity, theoretical guarantees about the generalization performance can be made.

SVMs perform an implicit embedding of data into a high dimensional feature space, where linear algebra and geometry may be used to separate data that is only separable with nonlinear rules in input space. To do so, the learning algorithm is formulated to make use of kernel functions, allowing efficient computation of inner products directly in feature space, without need for explicit embedding. Given a nonlinear mapping $\mathbf{\Phi}$ that embeds input vectors into feature space, kernels have the form

$$K(\mathbf{x}, \mathbf{z}) = \langle \mathbf{\Phi}(\mathbf{x}) \cdot \mathbf{\Phi}(\mathbf{z}) \rangle$$

SVM algorithms separate the training data in feature space by a hyperplane defined by the type of kernel function used. They find the hyperplane of maximal margin, defined as the sum of the distances of the hyperplane from the nearest data point of each of the two classes. The size of the margin bounds the complexity of the hyperplane function and hence determines its generalization performance on unseen data. The SVM methodology learns nonlinear functions of the form:

$$f(\mathbf{x}) = sgn(\sum_{i=1}^{l} \alpha_i y_i K(\mathbf{x}_i \cdot \mathbf{x}) + b)$$

where the α_i are Lagrange multipliers of a dual optimization problem. It is possible to show that only some of the α_i are non-zero in the optimal solution, namely those arising from training points nearest to the hyperplane, called support vectors. These induce sparseness in the solution and give rise to efficient approaches to optimization. Once a decision function is obtained, classification of an unseen example $\mathbf{x}$ amounts to checking on what side of the hyperplane the example lies.

The SVM approach is highly modular, allowing domain-specific selection of the kernel function used. Table 1 gives the kernels used during our evaluation of SVM-based expression classification. In contrast to previous "black box" learning approaches, SVMs allow for some intuition and human understanding. They deal with noisy data and overfitting (where the learned function perfectly explains the training

Kernel	Formula
Linear	$\mathbf{x} \cdot \mathbf{z}$
Polynomial	$(\gamma \mathbf{x} \cdot \mathbf{z} + c)^{degree}$
Radial Basis Function (RBF)	$\exp(-\gamma\lvert\mathbf{x} - \mathbf{z}\rvert^2)$
Sigmoid	$\tanh(\gamma \mathbf{x} \cdot \mathbf{z} + c)$

Table 1: Typically used kernel functions. $c, \gamma, degree$ **are parameters used to define each particular kernel from the family given.**

set but generalizes poorly) by allowing for some misclassifications on the training set. This handles data that is linearly inseparable even in higher space. Multi-class classification is accomplished by a cascade of binary classifiers together with a voting scheme. SVMs have been successfully employed for a number of classification tasks such as text categorization [14], genetic analysis [28] and face detection [19]. They currently outperform artificial neural networks in a variety of applications. Their high classification accuracy for small training sets and their generalization performance on data that is highly variable and difficult to separate make SVMs particularly suitable to a real time approach to expression recognition in video. They perform well on data that is noisy due to pose variation, lighting, etc. and where often minute differences distinguish expressions corresponding to entirely different emotions.

5.2 SVM Classification for Real Time Video

Our implementation uses `libsvm` [2] as the underlying SVM classifier. We encapsulate its stateless functionality in an object-oriented manner to work in an incrementally trained interactive environment. This avoids having to supply the set of training examples in its entirety before any classification can proceed and allows the user to augment the training data on-the-fly. It also enables us to export the entire state of a trained SVM classifier for later use. Hence, data gathered across several training sessions is preserved and can be re-used for classification. In addition, it allows for convenient combination of training data from multiple individuals to accomplish person-independent classification.

The user requests for training examples to be gathered at discrete time intervals and provides a label for each. This is combined with the displacements output by the feature extraction phase and added as a new example to the training set. The SVM is then retrained. Unseen expressions to be classified pass the same feature extraction process and are subsequently assigned the label of the target expression that most closely matches their displacement pattern by the SVM classifier.

Most computational overhead resides in the training phase. However, due to the fact that the training set is interactively created by the user and hence limited in magnitude and that the individual training examples are of constant and small size, overhead is low for typical training runs. This is also aided by the sparseness of the SVM solution, manifested by the fact that the number of support vectors which define the decision surface only increases sub-linearly as more examples are added to the training data. This is illustrated by Figure 5. Because evaluation of an SVM decision function on unseen input essentially amounts to checking which of the two subspaces defined by a separating hyperplane a point lies in, classification overhead is negligible. This allows our approach to perform classification both directly upon user request and continuously in real time for every frame in the video stream, with the current result being constantly reported back to the user.

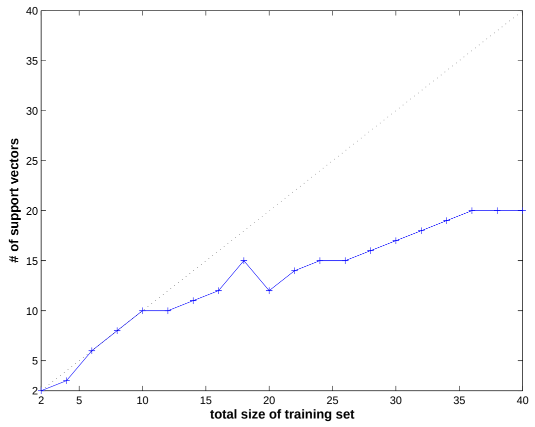


Figure 5: Number of support vectors defining the decision surface as training set size is increased.

6. EVALUATION

We evaluate our system by considering classification performance for the six basic emotions. Our approach makes no assumptions about the specific emotions used for training or classification and works for arbitrary, user-defined emotion categories.

To establish an upper bound on the recognition accuracy achievable for a combined feature displacement / SVM approach to expression recognition, we initially evaluated our method on still images from the Cohn-Kanade facial expression database. Features were manually defined for each image and displacements were subsequently extracted from pairs of images consisting of a neutral and a representative frame for each expression. A set of ten examples for each basic emotion was used for training, followed by classification of 15 unseen examples per emotion. We used the standard SVM classification algorithm together with a linear kernel. Table 2 gives the percentage of correctly classified examples per basic emotion and the overall recognition accuracy. Figure 6 shows the mean and standard deviation of the feature displacements for each basic emotion as extracted from the training data. It can be seen from these results that each emotion, expressed in terms of feature motion, varies widely across subjects. Particular emotions (eg. fear) or indeed particular combinations (eg. disgust vs. fear) are inherently harder to distinguish than others. However, the results also expose certain motion properties of expressions which seem to be universal across subjects (eg. raised mouth corners for 'joy', corresponding to a smile).

Kernel choice is among the most important customizations that can be made when adjusting an SVM classifier to a particular application domain. We experimented with a range of polynomial, gaussian radial basis function (RBF) and sigmoid kernels and found RBF kernels to significantly outperform the others, boosting overall recognition accuracy on the still image data to 87.9%. The human 'ceiling' in correctly classifying facial expressions into the six ba-

Emotion	Percent correct
Anger	82.2%
Disgust	84.6%
Fear	71.7%
Joy	93.0%
Sorrow	85.4%
Surprise	99.3%
Average	**86.0%**

Table 2: Recognition accuracy of SVM classification on displacements extracted from still images.

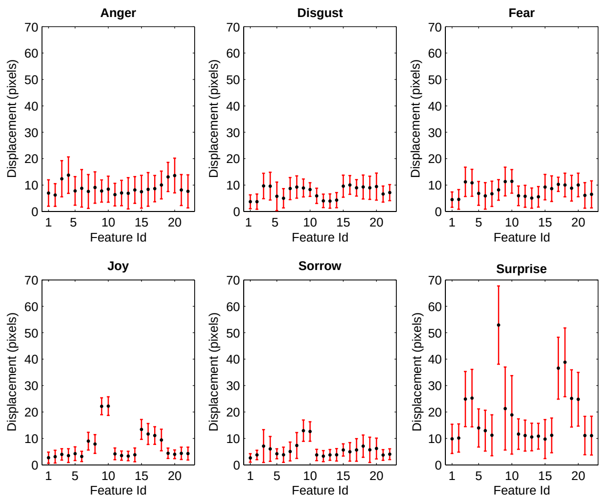


Figure 6: The mean and standard deviation for the characteristic motion of the 22 facial features for each basic emotion.

sic emotions has been established at 91.7% by Ekman & Friesen [11].

We next established the optimum recognition accuracy for video-based spontaneous expressions by having an expert user familiar with our approach and aware of how the basic emotions are typically expressed provide ten examples per basic emotion. A set of 72 test examples (12 per basic emotion) were then SVM classified using an RBF kernel. Table 3 gives the confusion matrix for these trials. It can be seen that for an expert user enthusiastically expressing emotions relatively little penalty in accuracy is incurred when moving from images to video. We are able to customize our tracker to a particular person by calibrating the face template used to initially locate facial features. This further increases recognition performance slightly and allows a tradeoff between generality and precision to be made.

We then evaluated our approach for the most challenging but ultimately desirable scenario of "ad-hoc" interaction, whereby wholly inexperienced users are asked to express emotions naturally in an unconstrained setup in terms of lighting conditions, distance from the camera, pose and so forth. Six individuals were asked to provide one training example per basic emotion and subsequently supply a number of unseen examples for classification. The resulting confusion matrix is shown in Table 4. The significantly lower accuracy achieved in these trials can mainly be attributed to the often significant amount of head motion by

Emotion	Percent correct
Anger	66.7%
Disgust	64.3%
Fear	66.7%
Joy	91.7%
Sorrow	62.5%
Surprise	83.3%
Average	**71.8%**

Table 5: Recognition accuracy for person-independent classification from video.

the subjects during feature extraction, resulting in inaccurate displacement measurements. In addition, since subjects were not given any instructions on how to express each basic emotion, cross-subject agreement on what constituted, for instance, an expression of fear was considerably lower, resulting in much more variable and difficult to separate data.

Finally, we tested the ability to recognize expressions of an expert user given only training data supplied by a different expert user, essentially person-independent classification. One example per basic emotion was supplied for training and 12 unseen examples were classified. This trial was repeated for various suppliers of training and testing data. Table 5 gives the recognition accuracy. Recognition was best for emotions having more uniform expression across subjects (eg. joy or surprise). Trials established that the more expressive the person supplying the training data, the higher the accuracy that can be achieved. Furthermore, we found it to be crucial that training and classification be performed under similar conditions, particularly with regard to the distance from the camera.

7. CONCLUSION AND DISCUSSION

In this paper we presented an approach to expression recognition in live video. Our results indicate that the properties of a Support Vector Machine learning system correlate well with the constraints placed on recognition accuracy and speed by a real time environment. We evaluated our system in terms of accuracy for a variety of interaction scenarios and found the results for controlled experiments to compare favorably to previous approaches to expression recognition. Furthermore, usable results are achieved even for very unconstrained ad-hoc interaction, which we consider a crucial prerequisite for the application of expression recognition in novel multimodal interfaces.

We currently deal with the prevalent problem of inaccuracies introduced by head motion by normalizing all feature displacements with respect to a single fiducial feature (the nose root) which provides an approximation to head motion in the video stream. This does not take into account rotational motion such as nodding or head tilting. We expect an additional increase in recognition accuracy if feature displacements can be properly adjusted in the presence of such motion. Moreover, using automatic SVM model selection [3] to determine optimal parameters of the classifier for displacement-based facial expression data should also increase classification accuracy further.

Incorporating emotive information in computer-human interfaces will allow for much more natural and efficient interaction paradigms to be established. It is our belief that methods for emotion recognition through facial expression

Emotion	Anger	Disgust	Fear	Joy	Sorrow	Surprise	*Overall*
Anger	**10**	0	0	0	0	0	83.3%
Disgust	0	**12**	0	0	0	0	100.0%
Fear	2	0	**10**	0	0	0	83.3%
Joy	0	0	0	**9**	0	3	75.0%
Sorrow	0	0	2	0	**10**	0	83.3%
Surprise	0	0	0	0	0	**12**	100.0%
						Total accuracy:	**87.5%**

Table 3: Person-dependent confusion matrix for training and test data supplied by an expert user.

Emotion	Anger	Disgust	Fear	Joy	Sorrow	Surprise	*Overall*
Anger	**19**	4	3	1	3	2	59.4%
Disgust	2	**18**	1	2	5	3	58.1%
Fear	7	3	**15**	0	3	1	51.7%
Joy	1	5	1	**21**	4	1	63.6%
Sorrow	2	3	7	3	**18**	0	54.4%
Surprise	2	3	1	0	2	**25**	75.8%
						Total accuracy:	**60.7%**

Table 4: Person-dependent confusion matrix for training and test data supplied by six users during ad-hoc interaction.

that work on real time video without preprocessing while remaining cheap in terms of equipment and unobtrusive for the user will play an increasing role in building such affective and intelligent multimodal interfaces.

8. REFERENCES

[1] B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, 1992.

[2] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[3] O. Chapelle and V. Vapnik. Model selection for support vector machines. In S. Solla, T. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12, pages 230–236. MIT Press, Cambridge, MA, USA, 2000.

[4] J. F. Cohn, A. J. Zlochower, J. J. Lien, and T. Kanade. Feature-point tracking by optical flow discriminates subtle differences in facial expression. In *Proceedings International Conference on Automatic Face and Gesture Recognition*, pages 396–401, 1998.

[5] J. F. Cohn, A. J. Zlochower, J. J. Lien, and T. Kanade. Automated face analysis by feature point tracking has high concurrent validity with manual faces coding. *Psychophysiology*, 36:35–43, 1999.

[6] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other Kernel-based Learning Methods*. Cambridge University Press, Cambridge, UK, 2000.

[7] M. N. Dailey, G. W. Cottrell, and R. Adolphs. A six-unit network is all you need to discover happiness. In *Proceedings of the Twenty-Second Annual Conference of the Cognitive Science Society*, Mahwah, NJ, USA, 2000. Erlbaum.

[8] C. Darwin. *The Expression of the Emotions in Man and Animals*. John Murray, London, UK, 1872.

[9] P. Ekman. *Emotion in the Human Face*. Cambridge University Press, Cambridge, UK, 1982.

[10] P. Ekman. Basic emotions. In T. Dalgleish and T. Power, editors, *The Handbook of Cognition and Emotion*. John Wiley & Sons, Ltd., 1999.

[11] P. Ekman and W. Friesen. *Pictures of Facial Affect*. Consulting Psychologists Press, Palo Alto, CA, USA, 1976.

[12] P. Ekman and W. Friesen. *Facial Action Coding System (FACS): Manual*. Consulting Psychologists Press, Palo Alto, CA, USA, 1978.

[13] I. Essa and A. Pentland. Coding, analysis, interpretation and recognition of facial expressions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):757–763, 1997.

[14] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 137–142, Heidelberg, DE, 1998. Springer Verlag.

[15] T. Kanade, J. Cohn, and Y. Tian. Comprehensive database for facial expression analysis. In *Proceedings of the 4th IEEE International Conference on Automatic Face and Gesture Recognition (FG'00)*, pages 46–53, 2000.

[16] M. Lades, J. C. Vorbrüggen, J. Buhmann, J. Lange, C. von der Malsburg, R. P. Würtz, and W. Konen. Distortion invariant object recognition in the dynamic link architecture. *IEEE Transactions on Computers*, 42:300–311, 1993.

[17] G. Littlewort, I. Fasel, M. Stewart Bartlett, and J. R. Movellan. Fully automatic coding of basic expressions from video. Technical Report 2002.03, UCSD INC MPLab, 2002.

[18] A. Mehrabian. Communication without words. *Psychology Today*, 2(4):53–56, 1968.

[19] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In

Proceedings of Computer Vision and Pattern Recognition '97, pages 130–36, 1997.

[20] M. Pantic and L. J. M. Rothkrantz. Automatic analysis of facial expressions: The state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1424–1445, 2000.

[21] B. Reeves and C. Nass. *The Media Equation: How People treat Computers, Television, and New Media like Real People and Places.* Cambridge University Press, New York, NY, USA, 1996.

[22] J. Steffens, E. Elagin, and H. Neven. Personspotter—fast and robust system for human detection, tracking, and recognition. In *Proceedings International Conference on Automatic Face and Gesture Recognition*, pages 516–521, 1998.

[23] Y.-L. Tian, T. Kanade, and J. F. Cohn. Recognizing action units for facial expression analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(2):97–115, 2001.

[24] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[25] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[26] V. N. Vapnik. *Statistical Learning Theory.* Wiley, New York, NY, USA, 1998.

[27] P. Viola and M. Jones. Robust real-time object detection. Technical Report 2001/01, Compaq Cambridge Research Lab, 2001.

[28] A. Zien, G. Rätsch, S. Mika, B. Schölkopf, T. Lengauer, and K. Müller. Engineering support vector machine kernels that recognize translation initiation sites. *Bioinformatics*, 16(9):799–807, 2000.

Appendix B

Sample Source Code

Video-based Application: Capturing an Example for 'Anger'

```
void CSVMTrackDoc::OnCaptureAnger()
{
    // Allocate an array to hold the landmarks
    // Deleted below when we don’t need it anymore
    int size = GetPointCount();
    CPoint* p = new CPoint[size];

    // Copy points from observer into the array of landmarks captured
    for (int i=0; i < size; i++)
    {
        // Adding 0.5 does rounding
        p[i] = CPoint((int)(0.5+m_observerL.GetnodePosArrPtrL()->arr[i].x),
                      (int)(0.5+m_observerL.GetnodePosArrPtrL()->arr[i].y));
    }

    // Array of double displacements. Is deleted after AddExample is called on SVMCore
    // (values will have been copied into SVMCore vectors)
    double* disp = new double[size];

    // Calculate displacements between captured array and last ‘neutral’ array
    for (int i=0; i < size; i++)
    {
        double iX = m_plastNeutralArray[i].x;
        double iY = m_plastNeutralArray[i].y;
        double fX = p[i].x;
        double fY = p[i].y;

        // Euclidean displacement for this landmark
        disp[i] = sqrt((fY-iY)*(fY-iY) + (fX-iX)*(fX-iX));
    }
    delete [] p;

    // Add Example as ‘anger’ into the SVM
    m_svm.AddExample(Anger, disp, size);

    // Retrain the SVM immediately
    m_svm.Train();
    delete [] disp;
}
```

Image-based Application: Part of Conversion to `libsvm` Input Format

```
/**
* One of the internal functions of SVMCore used to convert between
* the dual internal Vectors holding training data (labels & examples)
* and the svm_problem object needed to train the svm provided by libsvm
*/
void createProblem() {
    // Create a new svm_problem object
    prob = new svm_problem();

    // The number of examples used in training
    prob.l = vy.size();

    // prob.x holds the numerical training data
    // (vectors x_i of the SVM equations)
    prob.x = new svm_node[prob.l][];
    for(int i=0; i < prob.l; i++) {
        prob.x[i] = (svm_node[])vx.elementAt(i);
    }

    // prob.y holds the labels of the training examples
    prob.y = new double[prob.l];
    for(int i=0; i < prob.l; i++) {
        prob.y[i] = ((Integer)vy.elementAt(i)).doubleValue();
    }

    // Set gamma to 1/k where k = size of largest example vector
    if (param.gamma == 0) {
        param.gamma = 1.0 / max_index;
    }
}
```

Matlab Code[1] used in Calculating the Eigenfaces for a Set of Images

```
function [Eigvec, Eigval, Psi] = eigenfaces(A, numofvectors)
% Function returns the top numofvectors eigenvectors and eigenvalues
% of the covariance matrix of A, where A is a matrix of image vectors
% (i.e. each image is a column vector in A). The problem is reduced
% from N^2*N^2 (N^2 = size of image) to MxM (M = no. of images) by calculating
% the eigenvectors for A'A and then converting to the eigenvectors of the
% covariance matrix AA' (see Turk & Pentland's paper).

% How many columns there are in the image matrix A
numofimages = size(A,2);

% Compute the ``average'' face Psi. mean works column-wise, so need to
% transpose A to find the mean of each pixel value and then transpose back
```

[1]Adapted from an illustrative example by M. Dailey.

```
Psi = mean(A')';

% Compute the difference vector Phi_i of each face image from the average.
% Store these as column vectors in A
for i=1:numofimages
    A(:,i) = A(:,i) - Psi;
end;

% Compute the eigenvectors for the MxM problem
L = A'*A;
[Eigvec, Eigval]  = eig(L);

% Sort the eigenvectos in order of importance by the corresponding eigenvalues
[Eigvec, Eigval] = sortwrtvals(Eigvec, Eigval);

% Convert the eigenvectors of A'A to the eigenvectors of AA'
Eigvec = A*Eigvec;

% Extract the eigenvalues from the diagonal of Eigval & normalize them
Eigval = diag(Eigval);
Eigval = Eigval / (numofimages-1);

% Normalize all the eigenvectors to unit modulus, set them to null vectors
% if their corresponding eigenvalues are below a certain tolerance. Once an
% eigenvalue ‘underflows’, the remaining ones will as well. At the end of the
% loop, the top numofvectors eigenvectors will be non-zero.
eps = 1e-5;
for i=1:numofimages
    Eigvec(:,i) = Eigvec(:,i)/norm(Eigvec(:,i));

    if Eigval(i) < eps
        Eigval(i) = 0;
        Eigvec(:,i) = zeros(size(Eigvec,1),1);
        numofvectors = numofvectors - 1;
    end;
end;

Eigvec = Eigvec(:,1:numofvectors);
```

Video-based Application: Serializing SVMCore

```
void SVMCore::Serialize(CArchive& ar)
{
    if (ar.IsStoring()) {
        // Store how many elements there were in the m_vx CArray
        ar << (WORD) m_vx.GetSize();

        for (int i=0; i < m_vx.GetSize(); i++)
        {
            // Fetch vector of one example
            svm_node* x = m_vx[i];

            // First figure out the length of the array
            int j = 0; int size = 0;
            while(x[j].index != -1) {
                size++;
                j++;
            }

            // Now store size + 1 in the file, which gives us the size of the
            // array we must initialize to store all the values when loading
            ar << (WORD) (size+1);

            for(j=0; j <= size; j++)
            {
                if (j == size)
                    // Here we're at the last index == -1
                    // Don't try to store x[j].value, since it doesn't exist!
                    ar << x[j].index;
                else {
                    ar << x[j].index;
                    ar << (DOUBLE) x[j].value;
                }
            }
        }

        // Serialize the labels & SVM settings
        m_vy.Serialize(ar);
        ar << (WORD) m_max_index;       ar << (BOOL) m_empty;
        ar << (BOOL) m_trained;         ar << (WORD) m_svm_type;
        ar << (WORD) m_kernel_type;     ar << (DOUBLE) m_degree;
        ar << (DOUBLE) m_gamma;         ar << (DOUBLE) m_coef0;
        ar << (DOUBLE) m_cache_size;    ar << (DOUBLE) m_eps;
        ar << (DOUBLE) m_C;             ar << (WORD) m_nr_weight;
        m_weight_label.Serialize(ar);   // Serialize custom weight labels
        m_weight.Serialize(ar);         // Serialize custom weights
        ar << (DOUBLE) m_nu;            ar << (DOUBLE) m_p;
        ar << (WORD) m_shrinking;
    }
    else {
        // Clear the CArray
        m_vx.RemoveAll();
        m_vx.FreeExtra();

        // Extract size of the vector of examples from the byte stream
        WORD vecsize;
        ar >> vecsize;

        for (int i=0; i < vecsize; i++)
        {
            // Extract the size of the next example
            WORD size;
            ar >> size;

            // Construct a new array for the example's data
            svm_node* x = new svm_node[size];

            for (int j=0; j < size; j++)
            {
                int w;
                DOUBLE d;
                if (j == (size-1)) {
                    // Just read the index (-1), not the value (undefined)
                    ar >> w;
                x[j].index = w;
            }
            else {
                ar >> w;
                ar >> d;
                x[j].index = w;
                x[j].value = d;
            }
        }
        // Add example just read to the internal Vector
        m_vx.Add(x);
    }
    // Read in the labels & SVM settings
    m_vy.Serialize(ar);
    WORD w;
    BOOL b;
    DOUBLE d;
    ar >> w;   m_max_index = w;
    ar >> b;   m_empty = b;
    ar >> b;   m_trained = b;
    ar >> w;   m_svm_type = w;
    ar >> w;   m_kernel_type = w;
    ar >> d;   m_degree = d;
    ar >> d;   m_gamma = d;
    ar >> d;   m_coef0 = d;
    ar >> d;   m_cache_size = d;
    ar >> d;   m_eps = d;
    ar >> d;   m_C = d;
    ar >> w;   m_nr_weight = w;
    m_weight_label.Serialize(ar); // Read in custom weight labels
    m_weight.Serialize(ar);       // Read in custom weights
    ar >> d;   m_nu = d;
    ar >> d;   m_p = d;
    ar >> w;   m_shrinking = w;

    // Train the SVM with the loaded parameters
    Train();
  }
}
```

Appendix C

Image-based Application: Description of Methods & Fields

This appendix lists and describes the most important methods and fields of the component classes implementing the image-based classification application outlined in Section 3.2. It should act as a reference.

Class `Jacha`

Methods

`addFrame, removeFrame`: Methods called whenever a new internal frame needs to be spawned or destroyed. Handle layout of new frames on the desktop. Register/unregister frames with the application.

Fields

`svm`: The application's single SVM instance of class `SVMCore`. Provides an object-oriented interface to the `libsvm` functionality.

`iFrames, tcFrames, dfFrames`: `Vectors` keeping references to all currently existing internal Frames, all `TrainingClassFrames` and all `DefineFeatureFrames` of the application.

`qf`: A reference to the single `QueryFrame` that can be present in the application.

`newTrainingClass, defQueryEx, train, classify`: Actions for defining a new training class, defining the query example, training the SVM and classifying the unseen example. These are subclasses of Java's `AbstractAction`.

`svmSettings`: Single internal frame serving as a dialog to change SVM parameters. Implemented as an inner class within `Jacha`.

Interface `ExampleContainer`

Abstract Methods

`getExamples`: Should return a `Vector` of `Example` objects in the concrete container instance.

`getLabel`: Should return the unique numerical label for the emotion class the examples in the concrete container belong to.

`getName`: Should return the name (as a `String`) of the emotion class the examples in the concrete container instance belong to.

Class `TrainingClassFrame`

Methods

`getExamples`: Returns a `Vector` of `Example` objects for the emotion class corresponding to this frame. The result is converted into training input to the SVM in the main class `Jacha`.

`getLabel`: Returns the unique numerical label for the emotion class of this frame. Associates the SVM input produced from this frame with its emotion class during training.

`getName`: Returns the name (as a `String`) given to this emotion class during its definition by the user (e.g. 'disgust').

`updateStatus`: Called by a `DefineFeaturesFrame` to indicate that feature definition is complete. Updates the status of the associated `Example` to reflect this (background color changed to green).

`addNewlyClassified`: Passed the `Example` object of an unseen example that was just classified into this emotion class. Adds it to the examples in the class.

Fields

`examples`: `Vector` holding the `Example` objects for the particular training class.

`exampleComponents`: `Vector` containing references to the Java `Containers` that sit at the top of the hierarchy of GUI components for each example. It is used to map between `Example` objects and their GUI components[1] during event handling and when examples are removed (which causes their GUI components to be removed as well).

`classLabel, className`: The unique numerical label and the String representing the name of this frame's emotion class.

`addEx`: Action for adding a new example to the class. Two images (neutral and peak expressive frame) are loaded from the data set and a new GUI component representing the example and associated actions is added to the `TrainingClassFrame`.

[1]The `Example` at position *i* in the `examples` vector has corresponding GUI component at position *i* in the vector `exampleComponents`.

`removeEx`: Action to remove an example and its GUI component from the `TrainingClassFrame`.

`defineFeatures`: Action to start feature definition for an example. Causes a new `DefineFeaturesFrame` to be created within `Jacha`.

`loadFeatures, saveFeatures`: Actions to import/export features defined for a particular example to disk as text files.

`imageChooser, featureChooser`: File chooser dialogs to open images and for loading/saving features from/to disk. Derived from `JFileChooser`.

Class `QueryFrame`

Very similar in structure to `TrainingClassFrame`. Only the main differences are given below.

Methods

`getExamples`: Returns the unit `Vector` consisting of the single unseen `Example` object of the frame. The result is converted as SVM input during the classification stage in the main class `Jacha`.

Fields

`targetClassLabel, targetClassName`: Assigned the label and name of the target class after classification of the unseen example.

`addToCandClass`: Action to add the `Example` to its target emotion class after classification. Causes `addNewlyClassified(Example)` to be called on the appropriate `TrainingClassFrame`. This action only becomes active after successful classification.

`unseenExample`: The single `Example` object to be classified.

`exampleComponent`: The single GUI component associated with the `Example` object.

Class `DefineFeaturesFrame`

Fields

`currentExample`: The `ManualExample` object this frame defines landmarks for.

`canvasPanel`: Inner class derived from `JPanel` upon which the face image and the landmarks are drawn using double buffering.

`currentFaceImage`: A Java `BufferedImage` of the face image landmarks are being defined for.

`features`: The `LandmarkSet` used for keeping landmark positions. Stored within `currentExample` upon completion of landmark definition.

Interface `Example`

Abstract Methods

`getFeatures`: Should return a `double` array representing the feature vector corresponding to the example[2].

`getLabel`: Should return the numerical label of the class of the example[3], in case of an example used for training.

Class `ManualExample`

Methods

`getFeatures`: Returns the feature vector of `double` displacements between the neutral and peak images of the example.

`getLabel`: Returns the class label of the example.

Fields

`neutralFrame, peakFrame`: The two `Images` associated with the example.

`neutralLandmarks, peakLandmarks`: `LandmarkSets` representing the landmarks defined for each image.

`displacements`: Internal representation of the landmark displacements using `DisplacementSet` data structure. The `double` vector of displacements returned by `getFeatures` is calculated from this.

Class `DisplacementSet`

Methods

`addDisplacement, addXDisplacement, addYDisplacement`: Add a new landmark displacement to the data structure.

`getDisplacement, getXDisplacement, getYDisplacement`: Return the displacement of a landmark given its id.

[2]Equivalent to the $\mathbf{x}_i$ in the SVM equations in Section 2.4.

[3]Equivalent to the y_i in the SVM equations.

`getAllDisplacements, getAllXDisplacements, getAllYDisplacements`**:** Return a `double` array consisting of all landmark displacements, sorted by landmark id.

Fields

`euclidDisplacements, xDisplacements, yDisplacements`**:** Hashtables mapping landmark id → displacement.

Class `LandmarkSet`

Methods

`addLandmark, removeLandmark`**:** Add/remove a landmark to/from the set.

`getLandmark`**:** Returns the `Landmark` object from the set given its id.

`setCoordinates`**:** Updates the coordinates of a `Landmark` object in the set given its id and the new (x,y) pixel coordinates.

`getId`**:** Returns the id of the landmark that is closest to the (x,y) coordinates passed as parameters.

`getPresentIds`**:** Returns an `int` array of the ids of the landmarks present in the set.

Fields

`landmarks`**:** A `Vector` used internally to hold the `Landmark` objects in the set.

Class `Landmark`

Methods

`getDescription`**:** Returns a `String` describing the `Landmark` object depending on its id. For instance, for a landmark id of 101, "Left Pupil" would be returned.

Fields

`xCoord, yCoord`**:** The (x,y) pixel coordinates of the landmark.

`landmarkId`**:** The id of the landmark.

Interface `Classifier`

Abstract Methods

`addExample`: Should add a labelled example to the training data being kept with the classifier.

`clear`: Should reset the classifier to an untrained state.

`train`: Should train the classifier.

`predict`: Should perform classification given an unseen example and return the predicted target class.

Class `SVMCore`

Methods

`addExample`: Takes a `double` array representing a new training example and the associated class label and adds it to the training data kept within the class.

`train`: Trains the SVM classifier using the training data present so far.

`clear`: Removes any training data kept in the class, deletes any SVM model that may be present from prior training runs.

`predict`: Does classification by taking an unseen example and returning the numerical class label of the predicted target class.

`trainAndPredict`: Combines the previous two steps into one.

Fields

`vx, vy`: `Vectors` containing training examples and their labels, respectively (example at `vx[i]` has label `vy[i]`).

`svm_type, kernel_type, ...`: Various fields used as parameters to `libsvm` to modify SVM behavior.

Appendix D

Original Project Proposal

Introduction

"The question is not whether intelligent machines can have any emotions, but whether machines can be intelligent without emotions"
– Marvin Minsky in *The Society of Mind*, 1985

Humans are extraordinarily good at recognizing emotions expressed via facial articulation, consistently achieving success rates close to 90%. This classification process has long been regarded as being solely in the realm of human capability. Advances in artificial intelligence and machine learning techniques, however, have recently enabled computers to achieve similar classification accuracy.

Support Vector Machines (SVMs), introduced by Vladimir Vapnik in 1995 [34] and described thoroughly in [8], are powerful kernel-based learning techniques based on statistical learning theory that have recently come to play an increasing rôle for classification problems such as optical pattern classification, text classification and medical diagnosis. SVMs treat the classification problem as a quadratic optimization problem, broken down into smaller problems by bounding the margin between different classes. Thus, they perform well with large, high dimensional data sets, which makes them ideal for facial expression analysis.

I propose applying Support Vector Machine to the problem of classifying emotions, by implementing an SVM-based classification application. This entails taking a data set (facial photographs conveying a range of emotions) and processing it in a way to produce suitable input to an implementation of an SVM, providing a way of 'training' the SVM, querying the SVM with data and presenting and evaluating the classification results.

Resources

- I plan to mainly use my own PC (Dell Inspiron 8100 Laptop, Pentium III 1.13Ghz, 512MB RAM, 80GB hard disk, CD-RW). In case the system becomes unavailable, I will switch to one of the Lab's workstations while the problem is being fixed, provided I will have access to both the required Java development tools and enough space for the required corpus files (see *Special Resources*)

- I intend to use Java for this project. Even though Support Vector Machines and the image processing components require a large amount of numerical computation, the performance of a Java implementation should be more than adequate for this project. I plan to use a recent version of Sun's JDK (e. g. 1.4.1) as the compiler
- I shall use CVS version control software for my project, allowing me to retain control of my code as it progresses and offering roll-back mechanisms in case of any problems
- Data will regularly (each day Project data is modified) be backed up to the Pelican archiving service, as well as to CD-R Media. A new CD will be used each week. I will send older CDs to my home to provide off-site storage for disaster planning
- I will require a varied corpus of facial image data and perhaps video expressing various emotions. This can be provided by the Cambridge University Autism Research Centre, The Mindreading Project via my supervisor. There are various freely available facial emotion databases on the Internet. The corpus files tend to be high-resolution and relatively large

Starting Point

- My knowledge of Java stems from the Part IA course 'Programming in Java' and the various other Tripos courses that deal with the language. The largest piece of programming I have done in Java was the Part IB group project. My part, however, dealt mainly with numerical and threading issues. I have thus little experience with coding graphical applications
- Prior to the discussion that gave rise to this project, I had never heard of Support Vector Machines
- My knowledge of image processing algorithms is limited to the theory taught in the Part IB course 'Computer Graphics and Image Processing' and some experience with image editing applications such as Photoshop. I have not coded any such algorithms before
- No work has been done on this project before Michaelmas Term 2002

Project Outline

The overall project can be divided into four major components.

Firstly, the emotion classification application needs to be able to import and pre-process the facial image data to produce suitable input to an SVM. This requires pre-existence of facial landmarks in the images or a way for them to be specified (e. g. by user interaction) and an engine for converting these into appropriate numerical representation. This includes selection of appropriate data structures for each step.

Secondly, a basic Support Vector Machine must be incorporated into the application. Coding an SVM from scratch would be a Part II project in its own right. Fortunately, there are various good and free implementations available, together with source code. Integrating such an implementation into the overall framework will not be a trivial task, however.

Thirdly, there should be a way for the user to train the SVM by selecting various images to form a sample emotion class (or using images from predefined ones, such as 'happy' or 'sad') and to perform queries, presenting it with a new sample image.

Finally, the results of the classification output by the SVM need to be presented to the user in a meaningful manner.

The final project should be able to do simple classification, initially restricted perhaps to binary classification (e. g. only happy/sad), which should be enough to show the SVM approach to classification is feasible.

Possible Extensions

There are various extensions to this project, some of which may be implemented if the main part goes very smoothly and there is slack time in the implementation phase:

1. **Experimenting with different versions of Support Vector Machines** — There are various optimizations of the basic SVM, suited for different applications. Evaluate & Compare the performance of different versions
2. **Automatic facial landmarking** — Instead of having the user select landmarks in the input images or taking pre-existing landmarks, automate the process or implement 'user-aided' definition of landmarks. This requires the use extensive image processing routines and is likely to be rather difficult
3. **Expand possible emotion classes** — Get the application to work with a greater number of and more complex emotion categories. Could for instance train the SVM with 'happy, sad, angry, surprised' and then query it with a new sample belonging to one of these four categories (multi-class classification). This is more difficult than binary classification
4. **Investigate different ways of visualizing results** — There are many ways to improve upon the basic result reporting, for example by bar charts, line graphs, etc.
5. **Real-time classification of video** — Train the machine and then have it classify a video of a person performing various expressions continuously. This requires automatic landmarking to work and a suitable framework for dealing with video. Probably very difficult

Work to be undertaken & Difficulties to overcome

- Familiarizing myself thoroughly with Support Vector Machines, their operation, the necessary mathematical underpinnings and how they are implemented in practice. This will require a considerable amount of time and involve reading scientific papers and other literature
- Expanding my knowledge of Java to cover graphical application programming and the aspects of data conversion required by the project

- Acquiring knowledge & programming skills required to understand and implement the necessary image processing components
- Learning about standard techniques related to computer processing of faces, e. g. landmark placement, absolute feature position & displacement vector analysis, etc.

Plan of Work

The following timetable gives the overall schedule for the project. As suggested, it is divided into work packages, each a *fortnight* long:

10 October – 25 October	**Michaelmas Term** - Project Idea - Contacting Supervisor - Investigate Resources Required - Writing Project Proposal
Milestones:	*- Project Proposal* *- Submission of Project Proposal* *- Approval by Overseers*

26 October – 8 November	- Reading Literature on SVMs - Investigate available SVM implementations & choose one - Investigate possible image processing libraries to be used and their extensibility (e. g. for their use in automated landmarking) - Setting up development environment / backup provisions - Acquiring face image data - Write 'toy' programs that perform simple interaction with an underlying SVM
Milestones:	*- Good understanding of SVMs* *- Reasonably good idea of the interaction of chosen SVM module with other software components and SVM input format*

9 November – 22 November	- Devise first draft of architectural design of application - Brush up on Java GUI programming knowledge - Decide on appropriate data structures for each component & establish how data should be converted - Get to know image handling libraries to be used (loading, etc.) - Write rudimentary landmark placement code - Finalize overall and component design
Milestones:	*- Overall & component design diagrams and spec. Major interface classes between components* *- Understanding and feel for image processing part of project, some sample programs*

23 November – 6 December	- Implement a rudimentary draft of the GUI - Write image processing (loading, landmark placement, etc.) and data acquisition code (e. g. how to get numerical values from landmarks) - Write engine to convert from landmarks or displacement vectors to SVM input - Write foundation (‘glue’) that links major components together
Milestones:	*- Working user interface for landmarking* *- Landmark-to-SVM-input conversion engine* *- Integrated SVM, accepting converted data from engine*

7 December – 15 January	**Christmas Vacation** - Girlfriend & family time - Eat, sleep, relax, revise - If time available, do some code tweaking and reflection on project
Milestones:	*- Get something good from Santa*

16 January – 31 January	**Lent Term** - Code training and querying functionality into application - Implement rudimentary reporting of results from SVM to application - Improve user interface - Test simple output from SVM for correctness (components work together so far) - Write Progress Report & prepare presentation on status of project
Milestones:	*- Submission of Progress Report* *- Core functionality components integrated in application*

31 January	**Progress Report Submission Deadline**

1 February – 15 February	- Implement classification result reporting & visualization - Implement user-definable classification classes for training - Improve integration of major components of application, cleanup
Milestones:	*- Overall application integrated, working*

15 February – 28 February	- Debugging & extensive system testing - Experimentation & gathering of performance data - Implement extension if time allows - Start work on Introduction & Preparation Chapters of dissertation
Milestones:	*- Final implementation ready* *- Some evaluation data from application gathered for inclusion in dissertation* *- Extension(s) possibly implemented* *- First part of dissertation written*

1 March – 14 March	- Writing Implementation & Evaluation Chapters
Milestones:	*- Second Part of dissertation written* *- First draft of dissertation complete*

15 March – 25 April	**Easter Vacation** - Revision - Perhaps work on dissertation

26 April – 9 May	**Easter Term** - Submission of dissertation to supervisor - Feedback from supervisor - Preparation of diagrams & appendices - Proofreading of dissertation - Final review & submission of dissertation
Milestones:	*Final version of dissertation handed in*

10 May – 16 May	- Slack time in case of printer failure, illness or other delays
Milestones:	- *Final version of dissertation handed in in case of unforeseen delay*

16 May	**Dissertation Submission Deadline**

This schedule leaves enough slack time to accommodate illness and other unforeseen circumstances.