Bachelor Thesis
Realtime Action Unit detection

Freie Universität Berlin

Simón Auch

18.10.2016

Abstract

The automatic detection of emotions in faces plays an increasing role in todays life. The goal of this work is to provide a simple and flexible Framework for Action Unit detection with Support Vector Machines. Therefore I present the project for which this framework was made, a short overview of the theories used, the database used for training, the created framework, including landmark estimation and normalization, feature extraction, training Support Vector Machines and predicting Action Units with them.

Contents

List of Figures

Chapter 1

Project

The Project "Camera Facialis" [1] at Zuse Institue Berlin works on the automation of measuring and analysing the individual morphology of faces, especially in regards to emotions. Aim of the project is the creation of a comprehensive 3D morphology database, which can be used to develop new digital applications. Using a stereophotogrammatic setup the faces are captured and then reconstructed as a 3D model. Until now, one of the critical parts of the setup is the manual control of the moment of taking the images by a supervisor. A automatic detection of emotions in realtime would permit to automate this step.

Chapter 2

Action Unit Theory

The Facial Action Coding System (FACS) provides a theory about the correlation of facial movements based on muscle groups, the so called Action Units (AU), and the shown emotions on the face. The FACS is one of the most used systems to encode facial expressions, providing many databases that can be used for machine learning and allowing comparisons with other approaches. The Action Units therefore can be used to encode facial expressions in a anatomically motivated way and is often used as a feature space for later analysis like for example prediction of one the 6 basic emotions.

Chapter 3

Support Vector Machine Theory

A support vector machine (SVM) is a machine learning algorithm used for classification. In the simplest case a SVM is a large-margin classifier for a 2-Class problem. The training of a SVM therefore tries to find the plane with the maximal margin between the classes (see fig. 3.1). As most data used for training a SVM might contain outliers, a soft margin is introduced. The training then tries to maximize the margin while minimizing the sum of distances of points that are behind the margin of there class (see fig. 3.2). In cases where the input space is not linearly separable such a plane with good classification accuracy will not exist. Therefore a SVM transforms the input space into a kernel space using a kernel function, where the data gets separable by a hyperplane (see fig. 3.3). To do this efficiently the kernel function must fulfill certain requirements. When using such a kernel function, the hyperplane is fully described by all values used in training that lie behind their margin, these are then called support vectors.

The advantage of a SVM using a kernel function is clearly the possibly better classification accuracy, as not linearly separable data can be classified. The disadvantage is that a linear SVM can be evaluated much faster for prediction as all support vectors can be joined together and the hyperplane can then be described by their normal vector.

In case of a linear SVM with

- w - normal vector of hyperplane
- b - bias (shift of the hyperplane)

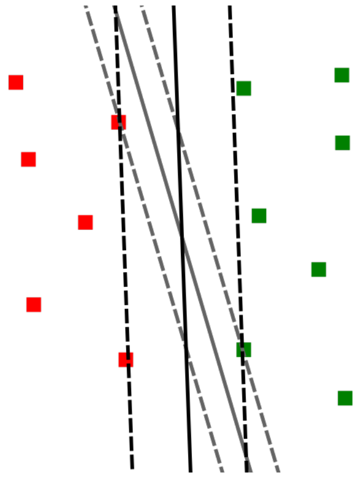


Figure 3.1: A margin (gray) and a maximum margin (black)

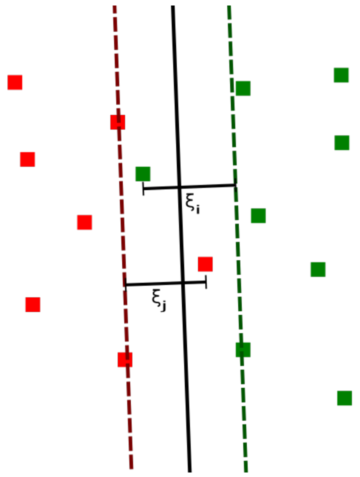


Figure 3.2: The error used with a soft margin SVM

- C - penalty for misclassifications
- x_i - input vector i
- $\xi_i > 0$ - slack variable of x_i
- $y_i \in \{-1, 1\}$ - correct class of x_i

the minimization problem can then be formalized as:

$$\frac{1}{2}\|w\|_2^2 \cdot C \sum \xi_i \tag{3.1}$$

While the additional conditions

$$y_i(\langle w, x_i \rangle + b) \geq 1 - \xi_i \tag{3.2}$$

must hold. The first part $\frac{1}{2}\|w\|$ of equation 3.1 is to maximize the margin between the classes, the second part $C \sum \xi_i$ is to account for misclassifications. On the left equation 3.2 contains the distance of a sample to the hyperplane, on the right it describes the soft margin 1 and the misclassification ξ_i.

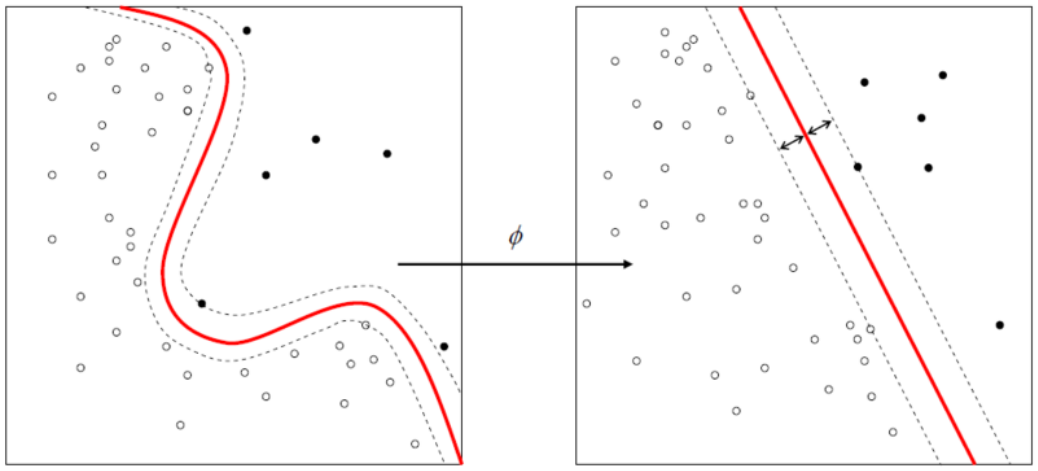


Figure 3.3: Transformation of input space into kernel space, where the classes become separable by a plane [11]

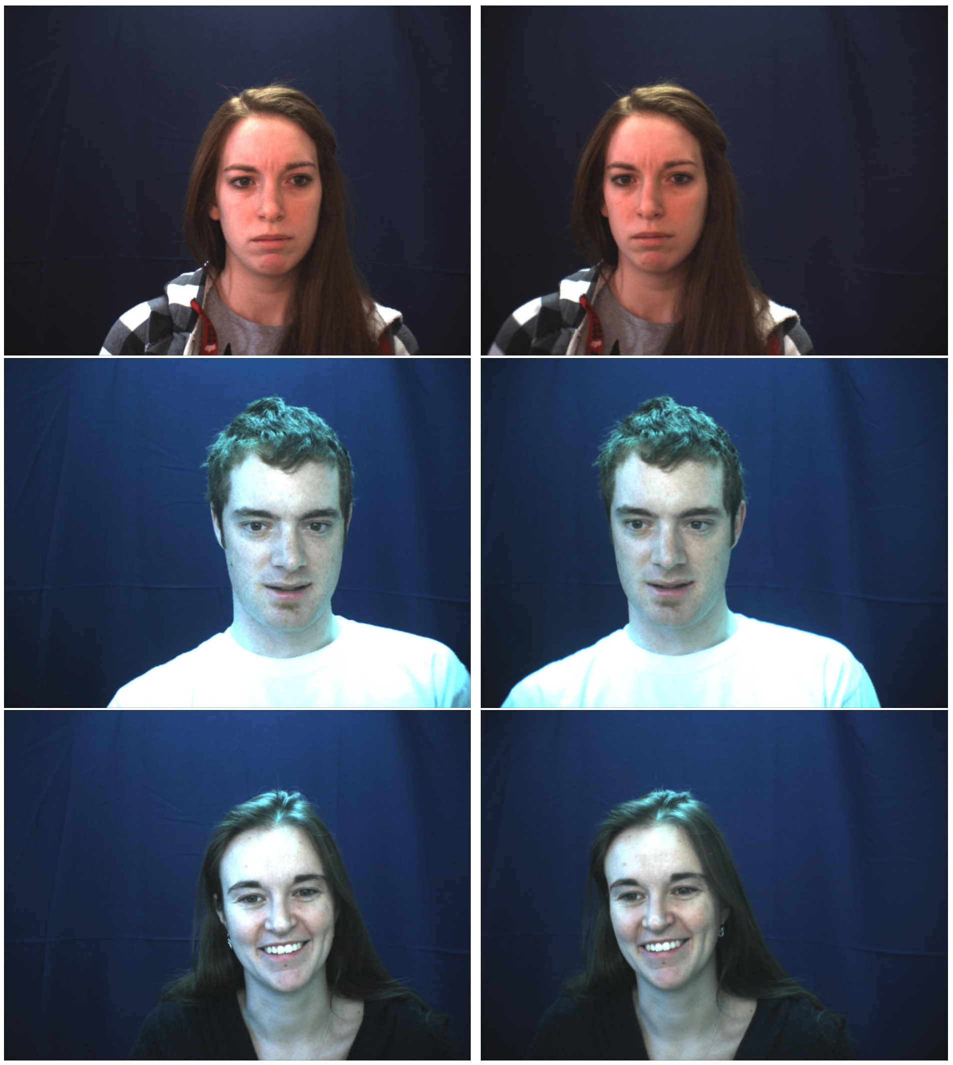
Chapter 4

Disfa Database

The Denver Intensity of Spontaneous Facial Action (DISFA) database consists of 27 stereo recordings (see figure 4.1). Each subject was recorded for 4 minutes with approximated 20 frames per second. All frames were then manually coded for the Action Units AU1, AU2, AU4, AU5, AU6, AU9, AU12, AU15, AU17, AU20, AU25, AU26 by one of two FACS coders (for examples see figure 4.2). The strength of activation of a Action Unit is encoded into the integers 0 to 5, where 0 stands for no activation and 5 stands for maximal activation. Also contained are calibration images for the cameras which can be used to rectify the frames obtained from the videos.

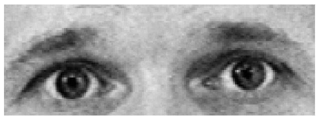
Unfortunately the database does not contain any information about wether or not the FACS coders would encode frames differently as well as there is no information about differences in the encoding from the coders over time.

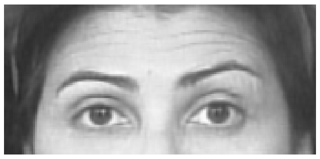
For this work I will only use features from the left video of the stereo pair, as all features are based on 2D images, therfore we have a total of about 130.000 frames for training.

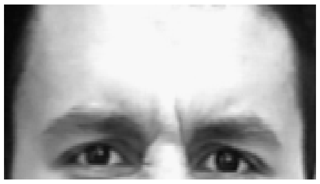


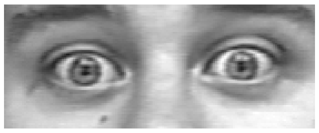
(a) left camera (b) right camera

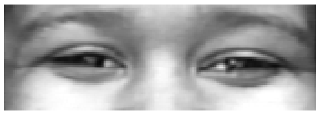
Figure 4.1: Example images from the DISFA database

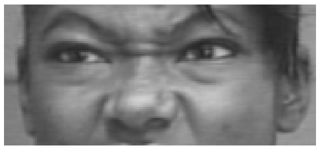

(a) AU1: Inner Brow Raiser

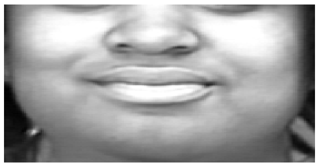

(b) AU2: Outer Brow Raiser

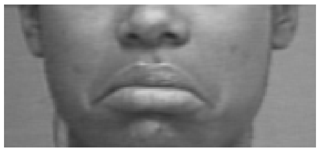

(c) AU4: Brow Lowerer

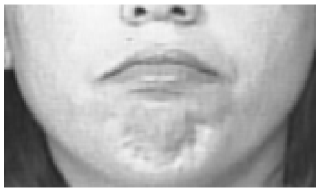

(d) AU5: Upper Lid Raiser

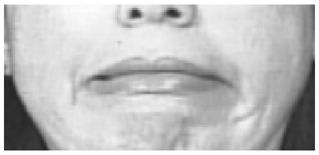

(e) AU6: Cheek Raiser

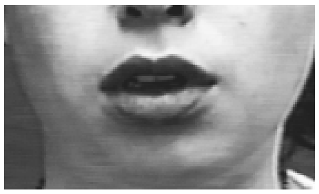

(f) AU9: Nose Wrinkler


(g) AU12: Lip Corner Puller


(h) AU15: Lip Corner Depressor


(i) AU17: Chin Raiser


(j) AU20: Lip Stretcher


(k) AU25: Lips part

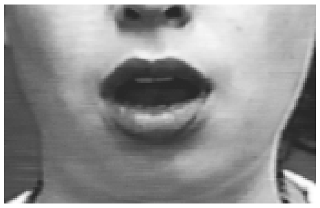

(l) AU26: Jaw Drop

Figure 4.2: Examples for all Action Units encoded in the DISFA database [3]

Chapter 5

Framework

In this chapter I will explain the different parts of the framework, ordered by the logical use of them in a program. The explanations will cover the basics of the algorithms used, comments on the decisions for specific algorithms and design decisions as well as some implementation specific details. An overview of the framework and its components can be seen in figure 5.1.

The framework is completely written in c++11. Libraries used are OpenCV [5], Qt [10], DLib [7], libsvm [2] and liblinear [4].

Connecting all the different parts of the framework we have the *data* class, which contains all variables regarding a frame. For each such variable it also has a boolean, indicating that the variable was set or not by a previous stage. This way, for example, the *landmark estimator* can test if the *face finder* found a face. A application can then be created by simply calling the appropriate functions of the classes in the framework and providing a *data* object, or by using the *signal and slot* semantics from Qt.

Despite the fact that the part *normalization* comes after *grouping* in the framework, we first will discuss *normalization*, as it is one of the reasons to use *grouping*.

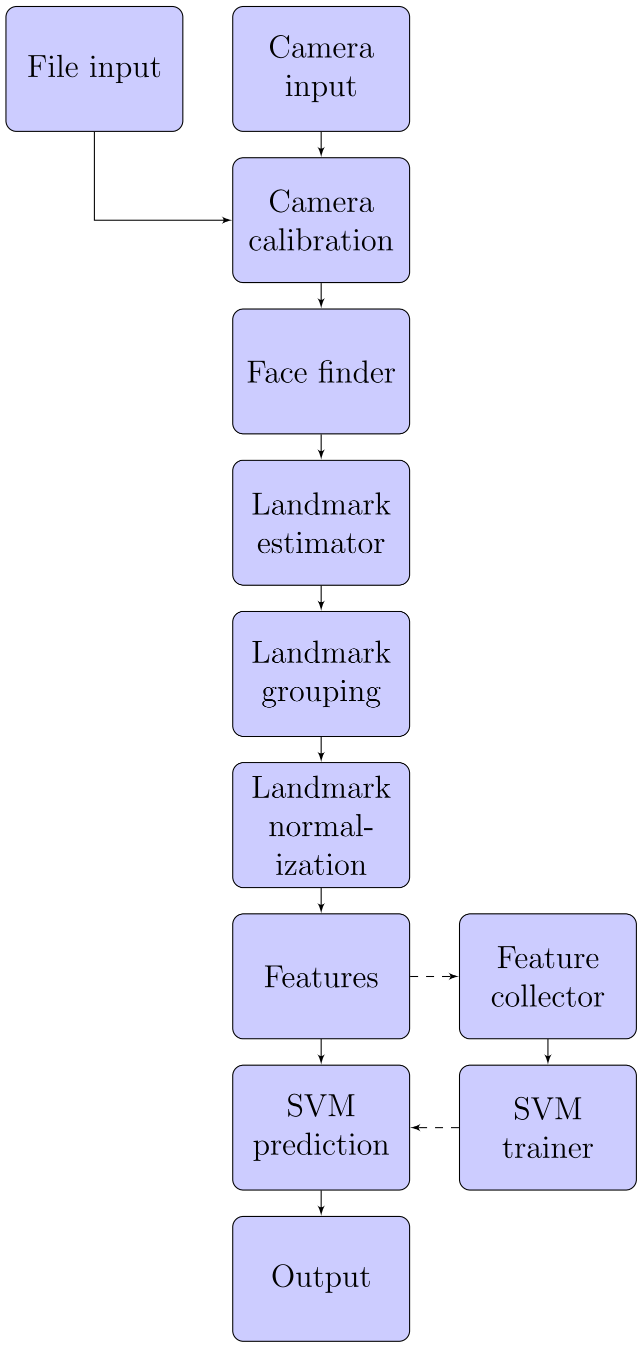


Figure 5.1: Basic framework setup

5.1 Input

The framework provides the two classes *camera_input* and *file_input* for input. The *camera* is the simplest of inputs, providing a thread that reads from an OpenCV *VideoCapture* object. This input is mainly intended for use in online detection, but can as well be used for in testing.

The second method of input, *file_input*, provides more options like ordered reading of files matching a regular expression and extracting meta information from the path like subject and frame number.

Data objects created by either of the two methods can and should then be passed to a instance of the *camera_calibration* class, which removes distortion effects of the camera. Intended for future work with the framework it also contains the *camera_stereo_calibration* class, which can be used to calculate the matrices needed for 3D reconstruction.

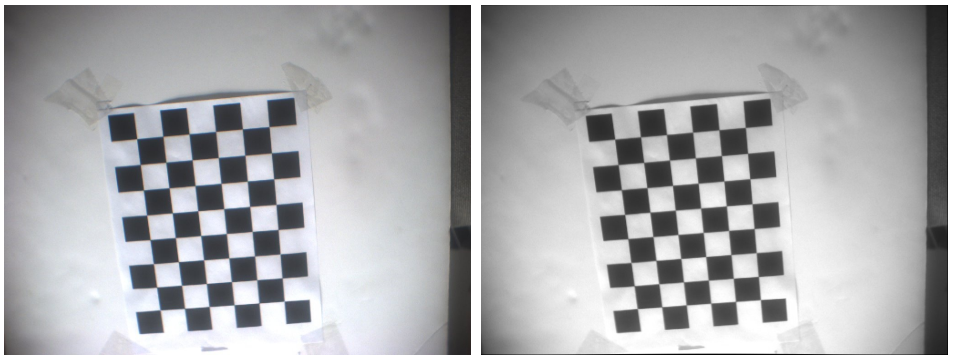


Figure 5.2: Before (left) and after (right) calibration. Notice the small black borders at the top and bottom.

5.2 Landmark estimation

The main task of this step is to reduce the dimensionality of the input data, without loosing information needed for Action Unit classification. The input in our case are images, the dimensionality being the amount of pixels multiplied by the amount of color channels. One way to achieve this goal is by using landmarks, describing the positions of selected points in an image. An example for such a landmark would be the outer left lip corner (see figure 5.3 for an example outcome).

Most algorithms for landmark estimation can be split into two steps:

- Find region of interest
- Estimate landmarks

For this work I will use the implementation for facial landmark estimation from the DLib library, which provides us with 68 landmarks. The rest of this chapter will give a short overview of the algorithms used in the implementation and some notes when using it.

Region of interest

For estimating facial landmarks the region of interest (ROI) is the region containing the face, typically described by an bounding box. The algorithm used by DLib for this consists of:

- Image pyramid
  Creates multiple scaled versions of the image to search for faces of different sizes.
- Histogram of oriented gradients (HOG)
  Calculates the histogram of the gradients for each 8x8 cell. The histograms are then normalized using the histograms of adjacent cells. In a sliding window manner the histograms of 10x10 cells are concatenated into a feature vector.
- Linear classifier
  A previously trained linear SVM is used to decide if the window contains a face or not.

Estimate landmarks

The landmark estimation is done using the DLib implementation of the paper "One Millisecond Face Alignment with an Ensemble of Regression Trees by Vahid Kazemi and Josephine Sullivan" (ERT) [6]. The DLib library ships with an model for this, trained on the iBUG 300-W face landmark dataset. For this work I will use this model, which provides us with 68 landmarks as can be seen in figure 5.3.

The algorithm uses a iterative approach, with the following steps:

- Start with average face
- Sample pixels around the landmarks
- Calculate features
- Update the shape using regression trees to calculate a $\delta x, \delta y$ for each landmark

For more information about how the algorithm works, I strongly recommend reading the paper, as explaining all the details is out of scope for this work. In the rest of this work L_i will denote the i-th landmark, x_i and y_i will denote the x and y component of the i-th landmark. Furthermore a landmark might be explicitly written as (x, y).

Notes

Despite not being big problems for this work I noticed three things that are noteworthy for future work:

- Speed
  HOG features are not especially fast to compute, for more than 30 frames per second it might be interesting to look at other solutions for this step. This however is no problem in this work, as the setup at ZiB only provides 30 frames per second and the faces cover most of the image, allowing us to subsample the image, strongly reducing the computation time needed.
- Trained head poses
  The linear classifier used in finding faces is only trained for faces being frontal, frontal left, frontal right and with a maximum tilt of $\pm 45°$.
- Landmark accuracy
  In some situations the landmarks around the lips fail to correctly fit these. These situations typically are either a wide open mouth or the corners of the lips being pulled down strongly.

5.3 Landmark normalization

Looking at a small example makes clear that the input for a SVM, during training and prediction, has to be normalized. Let's assume we have trained a

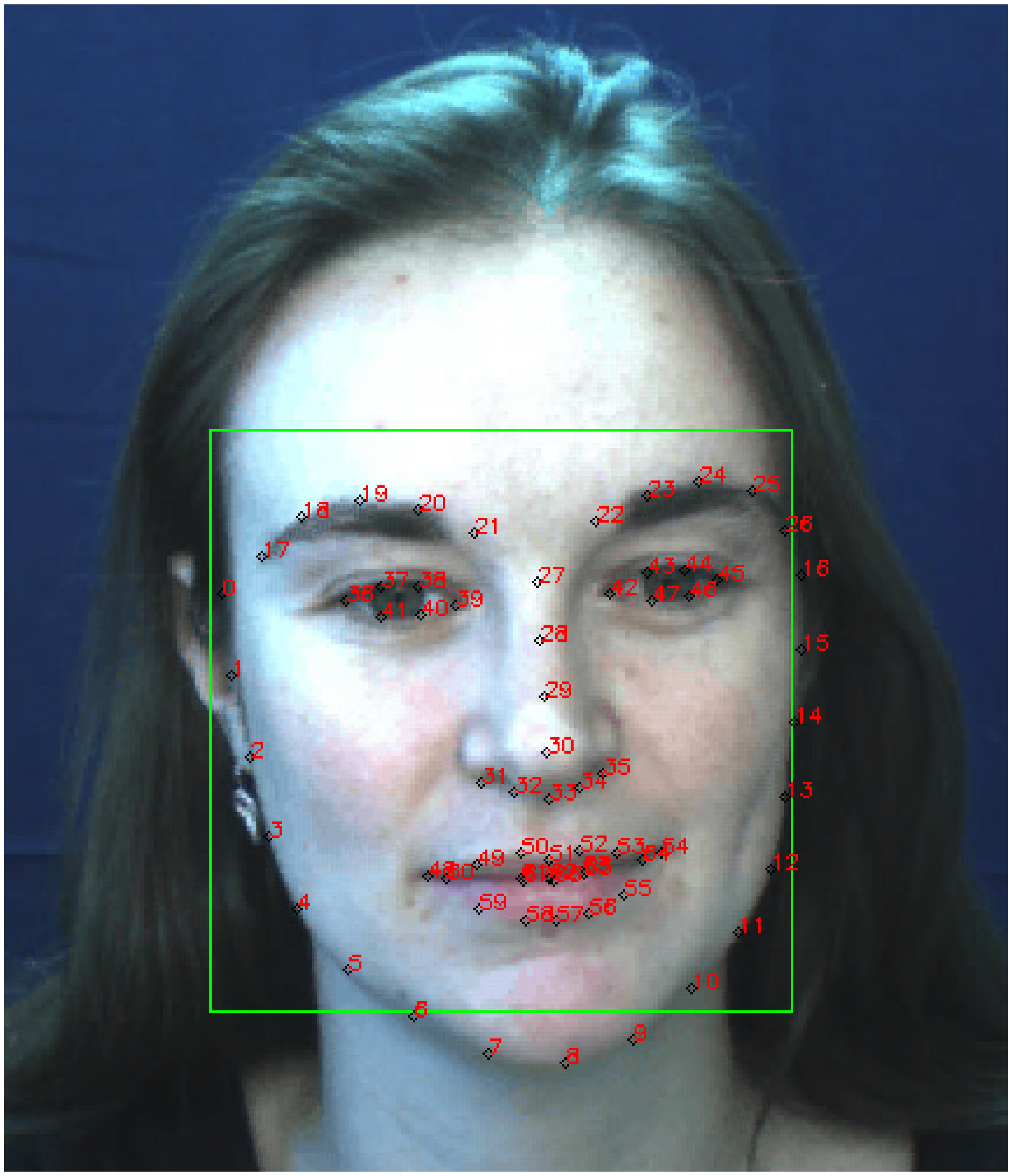


Figure 5.3: Example of the bounding box and the 68 landmarks estimated

SVM with the distances between our landmarks in pixels. If the distance between the test subject and the camera would change, maybe because the test subject moves or the camera setup was changed, the distances between the landmarks would change and the SVM would not classify correct anymore.

There are at least three properties that should be normalized:

- Translation
- Scale
- Rotation

The next sections will cover the different possibilities to normalize these properties. Therefore N is the amount of landmarks provided and $\{(u_i, v_i) | 0 \leq i < N\}$ will describe the set of points we want to use as reference.

Translation

Undoubtedly the most basic property to normalize, there still are plenty of options, for example:

- translate all points so that one point becomes $(0, 0)$
  $(x_i - x_t, y_i - y_t)$ for $t \in \{0, .., N - 1\}$
- translate the center of the points to $(0, 0)$
  $(x_i - x', y_i - y')$ with $x' = N^{-1} \cdot \sum_{i=0}^{N-1} x_i$ and $y' = N^{-1} \cdot \sum_{i=0}^{N-1} y_i$

The second option gives us the benefit of not needing to specify any point to use for normalization. The downside is, that when a group of points move, all other points will move too (e.g opening the mouth will move the upper half of the landmarks up). For now we will stick with second option, as the disadvantage should be solved by the grouping of landmarks discussed later.

Scale

Having removed the translation the scale is next. Again we have multiple options to choose from:

- scale all points so that the distance between a specific pair i, j is 1

$$
\begin{aligned}
1 &= \sqrt{(x_i * s - x_j * s)^2 + (y_i * s - y_j * s)^2} \\
1 &= \sqrt{s^2 * ((x_i - x_j)^2 + (y_i - y_j)^2} \\
1 &= s^2 * ((x_i - x_j)^2 + (y_i - y_j)^2 \\
s &= \sqrt{((x_i - x_j)^2 + (y_i - y_j)^2)^{-1}}
\end{aligned}
$$

- scale all points so that the root mean square distance to the center becomes 1

$$
\begin{aligned}
1 &= \sqrt{\frac{\sum_{i=0}^{N-1} (x_i * s)^2 + (y_i * s)^2}{N}} \\
1 &= \sqrt{\frac{s^2 * \sum_{i=0}^{N-1} x_i^2 + y_i^2}{N}} \\
1 &= s * \sqrt{\frac{\sum_{i=0}^{N-1} x_i^2 + y_i^2}{N}} \\
s^{-1} &= \sqrt{\frac{\sum_{i=0}^{N-1} x_i^2 + y_i^2}{N}}
\end{aligned}
$$

Again, the disadvantage of the of the first solution is that it needs information which points to use for normalization. But more important than that is the fact that if the distance of these points changes, the whole scaling does too. The second option, inspired from statistics, has a similar problem as option two from translation: When a group of points moves away from the center, all other points will be drawn to the center. But just as with the translation this problem should be solved by the grouping of landmarks.

Rotation

Out of the three, rotation is probably the most difficult to normalize. The rotation performed always is calculated as

$$(x_i * cos(w) - y_i * sin(w), x_i * sin(w) + y_i * cos(w))$$

Options for finding w include rotating so that:

- $x_i = x_j$ holds for the points i, j

$$\begin{aligned}
x_i * cos(w) - y_i * sin(w) &= x_j * cos(w) - y_j * sin(w) \\
cos(w) * (x_i - x_j) &= sin(w) * (y_i - y_j) \\
tan(w) = \frac{sin(w)}{cos(w)} &= \frac{x_i - x_j}{y_i - y_j}
\end{aligned}$$

- $y_i = y_j$ hold for the points i, j

$$tan(w) = \frac{sin(w)}{cos(w)} = \frac{y_i - y_j}{x_j - x_i}$$

- the sum of square distances (SSD) to a reference shape is minimal

$$\begin{aligned}
0 &= \frac{\delta}{\delta w} \sum_{i=0}^{N} + ((x_i * cos(w) - y_i * sin(w) - u_i)^2 \\
&\qquad + ((x_i * sin(w) + y_i * cos(w) - v_i))^2 \\
&= \frac{\delta}{\delta w} \sum_{i=0}^{N} + (x_i c)^2 + (y_i s)^2 - 2x_i y_i cs - 2U_i(x_i c - y_i s) \\
&\qquad + (x_i s)^2 + (y_i c)^2 + 2x_i y_i sc - 2V_i(x_i s + y_i c) \\
&= \frac{\delta}{\delta w} \sum_{i=0}^{N} - 2U_i(x_i c - y_i s) - 2V_i(x_i s + y_i c) \\
&= \sum_{i=0}^{N} + 2U_i(x_i s + y_i c) - 2V_i(x_i c - y_i s) \\
&= \sum_{i=0}^{N} + s(2U_i x_i + 2V_i y_i) - c(-2U_i y_i + 2V_i x_i) \\
tan(w) = \frac{sin(w)}{cos(w)} &= \frac{\sum_{i=0}^{N} 2V_i x_i - 2U_i y_i}{\sum_{i=0}^{N} 2U_i x_i + 2V_i y_i}
\end{aligned}$$

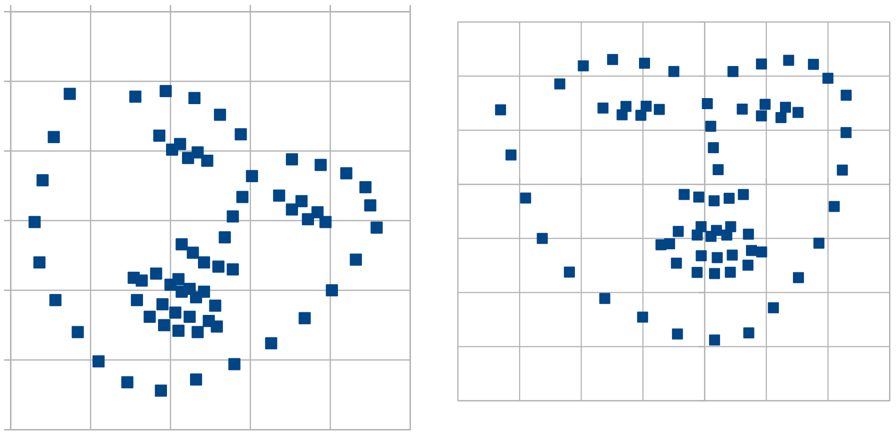


Figure 5.4: Before (left) and after (right) landmark normalization

The disadvantages of the first two options are obviously that if the reference points i, j move, all other points move. Similar to the problem while scaling the third option reducing the SSD has the disadvantage that if a group of points move, all other points move too. The reference points needed for the third option can be obtained for example by taking the average positions of points in multiple frames or using a generalized Procrustes analysis (GPA). In case of this work the reference points were created using a GPA on the CK+ dataset [9] and was provided by the work group at ZiB.

Given the special case that the center of the reference points is at $(0, 0)$ the selected steps equal a Procrustes analysis (see figure 5.4).

5.4 Landmark grouping

Having explained the problems with landmark normalization, I will now present a novel method to further improve classification accuracy (see table 6.2, 6.3 and 6.4). Taking a look at many of the AU's encoded in the DISFA database it is clear that many of them are "local", in the sense that they are defined by only a part of the morphology of the face. The Action Units AU12, AU15, AU20 and AU25 (lip corner puller, lip corner depressor, lip stretcher and lips part) should be defined by the morphology surrounding the lips and not take into account e.g. the morphology around the eyes. Having these "local" Action Units it makes sense to normalize corresponding landmarks independently from other landmarks.

Based on this idea we introduce the so called "landmark groups". These groups contain specific landmarks, each group describing one important region of the human face. The proposed groups are:

- Left eye
  $G_{e_L} = \{L_{42}, L_{43}, L_{44}, L_{45}, L_{46}, L_{47}\}$
- right eye
  $G_{e_R} = \{L_{36}, L_{37}, L_{38}, L_{39}, L_{40}, L_{41}\}$
- Both eyes
  $G_e = G_{e_L} \cup G_{e_R}$
- Left eyebrow
  $G_{b_L} = \{L_{22}, L_{23}, L_{24}, L_{25}, L_{26}\}$
- Right eyebrow
  $G_{b_R} = \{L_{17}, L_{18}, L_{19}, L_{20}, L_{21}\}$
- Both eyebrows
  $G_b = G_{b_L} \cup G_{b_R}$
- Eyes and Eyebrows
  $G_{eb} = G_e \cup G_b$
- Nose
  $G_n = \{L_{27}, L_{28}, L_{29}, L_{30}, L_{31}, L_{32}, L_{33}, L_{34}, L_{35}\}$
- Mouth
  $G_m = \{L_{48}, L_{49}, \ldots, L_{66}, L_{67}\}$

Using these groups, the SVM trained later should be able to achieve higher accuracy due to less influence of unimportant landmarks to a specific Action Unit.

5.5 Features

After normalizing the groups of landmarks we now need to extract the features we want to use for training our SVM. The features used can be partitioned into following groups:

- positions
  The position features contain all the normalized positions of the landmarks, once for all and once for each of the groups defined earlier.

- distances
  The distance features contain all distances between the normalized positions of landmarks, once for all landmarks and once for each group.

- geometric features
  The geometric features were implemented following the paper "Real-time Emotion Recognition - Novel Method for Geometrical Facial Features Extraction." [8]. In cases were landmarks used in the paper were not present in the landmarks dlib provides, the closest available landmarks were used instead. The implemented features include:

  - Linear features
    The idea behind the linear features proposed is to give a measure of the "relative movements between facial landmarks while expressing emotions" [8, Section 2.2]. The features F can then be calculated as:

$$\begin{aligned} DEN &= \overline{L33_y L37_y} \\ F_1 &= \overline{L19_y L37_y}/DEN \\ F_2 &= \overline{L33_y L51_y}/DEN \\ F_3 &= \overline{L33_y L57_y}/DEN \end{aligned}$$

  - Eccentricity features
    The idea behind the eccentricity features proposed describe the similarity of a ellipse described by three points and a circle. The ellipses used for this are:
    - $(L48, L54, L51)$ upper mouth
    - $(L28, L54, L57)$ lower mouth
    - $(L36, L39, L38)$ upper left eye
    - $(L36, L39, L40)$ lower left eye
    - $(L42, L45, L43)$ upper right eye
    - $(L42, L45, L47)$ lower right eye

- $*$ $(L17, L21, L19)$ left eyebrow
- $*$ $(L22, L26, L24)$ right eyebrow

The eccentricity e of a triple (A, B, C) can then be calculated as:

$$e = \frac{\sqrt{a^2 - b^2}}{a}$$
$$a = \frac{B_x - A_x}{2}$$
$$b = A_y - U_y$$

Assuming that $b^2 \leq a^2$ and $A_y = B_y$. The implementation however does not assume this, in contrast to the paper.

5.6 Training

Finally, now that we have around 130.000 frames, features for each frame and the results we want to predict, we can start to think about training our SVMs. As we want to train one SVM per Action Unit and the training will be equal for all SVMs the rest of this section will just refer to "the SVM". Furthermore we will call the set of a frame, the corresponding features, prediction and result a *sample*. The result is extracted from the DISFA database for the corresponding frame and Action Unit. In this section we will cover the choices made for

- SVM type
- Classes
- Samples used for training
- Training and Test sets
- Parameter search
- Feature reduction

SVM type

First of all, we need to decide wether or not we want to use a linear SVM or not. Given the fact that linear SVMs can be trained and evaluated much faster than other SVMs, we will use linear SVMs as our prediction step should be as fast as possible for the setup. Later we will also see, that this has a benefit for the Feature reduction.

Classes

As we remember, the labels in the DISFA database range from 0 to 5. The most common approach would now be a *multiclass SVM*, which (in our case) would actually consist of 6 SVMs. Each of these SVMs would be a *one against all SVM*, meaning that only one class was used for positive samples and all other classes where negative samples.

This however does not work well in our case. Take for example the AU25 (Lips part), which will most probably be described by the distance between landmarks on the lower and upper lip. This distance would then be partitioned into ranges for classifying 0,1,...,5. A SVM from a multiclass SVM would now try to separate these ranges, in case of a linear SVM it could only use 1 value for this. This would obviously only work for the first and last range, but not for the ranges in the middle, as these are described by 2 values, a minimum and a maximum. Regardless of being a very reduced example, it still gives an idea of the problem using a linear multiclass SVM in our training.

To overcome this we will reduce our samples to two classes. Therefore we introduce a *border* B, which can be used to determine the positive and negative samples:

$$result' = result \geq B$$

where *result* is the strength encoded in the DISFA database for the current sample, which for the later steps is replaced with $result'$.

Samples used for training

Looking at the 130.000 frames it can be noticed that the amount of negative samples N_- (Action Unit in question not present) is much higher than the amount of positive samples N_+. If we would now just train our SVM it most probably would just "learn" saying "No", as this would achieve a accuracy similar to $\frac{N_- + N_+}{N_-}$ which when $N_- \gg N_+$ is nearly 1. This behavior is obviously not wanted. Therefore we randomly take samples out of both classes until we have all samples from at least one class. We will refer to this as *balanced set* SET_B.

Training and Test sets

To be able to compare the accuracy of different SVMs (for the same Action Unit) we need a so called *test set* (SET_E) which will only be used to measure the accuracy of the SVMs. The SET_E will therefore be a random subset of SET_B:

$$SET_E \subset SET_B$$
$$|SET_E| = |SET_B| \cdot 20\%$$

The *training set* (SET_T) contains all other samples:

$$SET_T = SET_B \setminus SET_E$$

Parameter search

Looking at the term a SVM tries to minimize

$$\frac{1}{2}||w||_2^2 + C\sum_{i=1}^{m} \varepsilon_i$$

we see the constant C which describes how strong samples on the wrong side of their margin are penalized. This value can be tuned in order to find the SVM with the best accuracy. To do this we use a method called *k-fold cross validation.*

In k-fold cross validation the training samples are split into k parts. For each value tested as parameter, every of the k parts is once used as test set while the rest is used as training set. The resulting accuracy on the k parts used as test set is then averaged. The value with the best average is then used for actual training.

The values tested with cross validation for this work were $\{2^0, 2^1, \dots 2^{10}\}$.

Feature reduction

Having many features is not always good, especially if these do not contribute to the problem with more information than is already contained in other features. This is well known under the name of "curse of dimensionality" and can be explained by the amount of variables a model has and therefore must be "found" during training. In case of a linear SVM with N features we have $N + 1$ variables that must be trained (the extra one being the bias).

The actual need of this becomes more evident when looking at the total amount of 3233 features, when using all features described earlier. Now first, many of these features will not be of any use for specific Action Units, as they concern other regions of the face. Second many of the features contain similar information, of which most probably only one is needed.

A well known algorithm for this is the "recursive feature reduction". This algorithm works by training the model, removing the features with lowest impact on classification. This procedure is then repeated until no features are left or a significant decrease in classification accuracy is noticed.

In our case we remove the lowest 20% of features until the amount of features does not change anymore, which is 4. For this we train a linear SVM on SET_T with the optimal parameter C found during parameter optimization. To later select the SVM with the best accuracy we also measure the accuracy of this SVM on SET_E. After removing features the optimal parameter C might have changed, therefore parameter optimization is repeated.

At the end the SVM with the best accuracy on SET_E is used.

5.7 Prediction

When using a linear SVM prediction becomes as simple as

$$\langle w, x \rangle + b > 0$$

where w is the normal vector of the hyper plain, b is the bias and x is the feature vector. Taking into account the feature reduction from the training step, the normal vector w does not have the same dimensionality for all action units. This problem can be solved simply by expanding w into w', w_i (w'_i) denoting the element at position i. To do this we need two mappings, the first one tells us if a feature was used $f(i) : \mathbb{N} \to \mathbb{B}$, and the second mapping tells us what index the feature has after feature reduction $g(i) : \mathbb{N} \to \mathbb{N}$.

$$w'_i = \begin{cases} w_{g(i)} & \text{if } f(i) \\ 0 & \text{else} \end{cases}$$

By inserting zeros at features that are not used by the SVM, the same feature vector can be used for all SVM's. This step must only be done once for a SVM and is in fact already done during the training step, removing the need to save f and g.

Chapter 6

Results

In this chapter I present the results obtained with different sets of features F, different values for the border for classification B and with or without feature reduction R. The possible sets of features are:

- F_P - The positions of the landmarks
- F_D - The distances of the landmarks
- F_E - The geometric features as described in section 5.5
- F_G - The positions and distances of the landmarks from the groups defined in section 5.5

Possible values for B are $\{1, 2, 3, 4, 5\}$. Higher values for B mean less positive samples for training and testing. Due to this, the values 4 and 5 should not be used, as some Action Units miss positive samples. For improved comparability between results for the same AU with the same value for B, the training and test partitions were always the same. Table 6.2, 6.3 and 6.4 show the accuracy of the trained SVM on the test data, defined by $\frac{N_C}{N} \cdot 100$, where N_C is the amount of correct classifications and N is the amount of samples (see table 6.1).

Looking at the results, one can see:

- the additional features in F_G improve overall accuracy (column 6)
- the feature sets $F_P \cup F_D \cup F_E$ and F_G are disjunct (column 4,5)
- higher values of B have better accuracy

The last point might be due to less accurate labeling of the data, as already explained in chapter 4.

Table 6.1: Amount of samples used for training and testing ($|SET_B|$)

B	1	2	3
AU 1	17428	12992	9498
AU 2	13900	10460	8592
AU 4	48486	39172	23942
AU 5	5444	2290	856
AU 6	38968	20654	8682
AU 9	14264	10946	6876
AU 12	61552	33666	19964
AU 15	15724	5364	2128
AU 17	25860	13176	4808
AU 20	9064	5882	2666
AU 25	92104	72494	44624
AU 26	49952	23066	8120

Table 6.2: SVM accuracy with $B = 1$

B	1				
F_P	y	y	y	n	y
F_D	n	n	y	n	y
F_E	n	y	y	n	y
F_G	n	n	n	y	y
R	y	y	y	y	y
AU 1	84.85	85.03	87.01	85.54	**87.87**
AU 2	88.71	89.14	91.01	89.35	**91.04**
AU 4	87.74	87.86	89.44	87.45	**89.72**
AU 5	88.71	89.44	90.45	89.16	**90.82**
AU 6	89.45	89.93	90.79	89.26	**91.17**
AU 9	90.57	90.57	92.46	90.61	**92.95**
AU 12	89.20	89.28	90.34	88.59	**90.64**
AU 15	84.10	84.32	88.39	84.48	**89.35**
AU 17	81.83	82.21	85.54	81.90	**85.98**
AU 20	82.74	83.07	87.15	84.94	**88.47**
AU 25	92.32	92.51	93.39	92.51	**93.45**
AU 26	85.50	85.58	87.53	86.39	**88.21**
Average	87.14	87.41	89.46	87.52	**89.97**

Table 6.3: SVM accuracy with $B = 2$

B	2				
F_P	y	y	y	n	y
F_D	n	n	y	n	y
F_E	n	y	y	n	y
F_G	n	n	n	y	y
R	y	y	y	y	y
AU 1	88.80	89.15	91.27	89.84	**92.19**
AU 2	92.45	93.83	94.17	92.64	**94.65**
AU 4	88.81	89.15	90.41	88.73	**90.76**
AU 5	96.72	96.72	96.94	96.72	**97.16**
AU 6	92.23	92.59	93.61	92.42	**94.05**
AU 9	91.42	91.14	93.33	90.73	**93.70**
AU 12	94.10	94.06	95.13	93.61	**95.22**
AU 15	90.21	89.93	92.73	90.40	**93.38**
AU 17	87.52	87.75	89.87	87.94	**90.14**
AU 20	89.97	90.74	92.95	90.74	**93.54**
AU 25	94.54	94.62	95.37	95.07	**95.79**
AU 26	90.25	90.49	91.76	90.38	**92.18**
Average	91.42	91.68	93.13	91.60	**93.56**

Table 6.4: SVM accuracy with $B = 3$

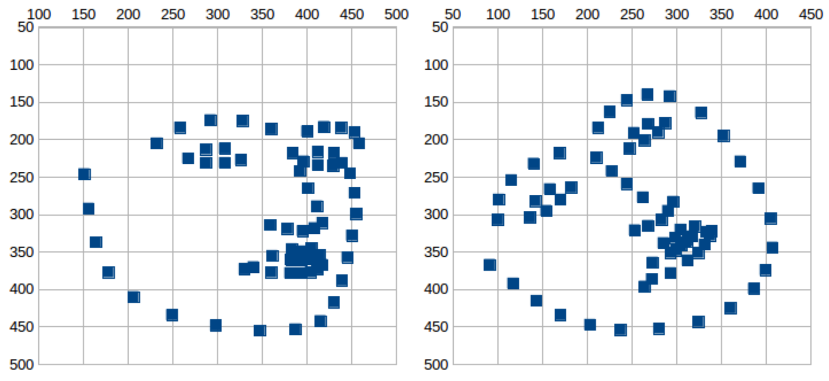
B	3				
F_P	y	y	y	n	y
F_D	n	n	y	n	y
F_E	n	y	y	n	y
F_G	n	n	n	y	y
R	y	y	y	y	y
AU 1	90.68	91.89	93.74	93.58	**94.11**
AU 2	95.00	96.57	97.15	95.99	**97.38**
AU 4	89.31	90.02	91.86	90.27	**92.40**
AU 5	**98.26**	**98.26**	**98.26**	97.67	**98.26**
AU 6	94.36	94.36	**94.93**	93.44	**94.93**
AU 9	93.82	93.75	94.84	93.90	**95.06**
AU 12	96.52	96.49	96.72	95.54	**96.77**
AU 15	96.24	**96.48**	**96.48**	94.13	**96.48**
AU 17	93.35	93.66	94.39	91.48	**94.49**
AU 20	90.45	91.01	**94.19**	91.95	**94.19**
AU 25	97.39	97.46	**97.88**	97.28	97.87
AU 26	94.15	94.03	**94.95**	91.63	**94.95**
Average	94.13	94.50	95.45	93.90	**95.57**

Chapter 7

Future work

3D landmarks

3D landmarks, obtained by triangulation of 2D landmarks from a stereo image, might contain more information and thus could lead to better classification results. This becomes especially clear, when looking at the task of removing head rotation from 2D landmarks. This head rotation and the rotation of landmarks should not be confused, see figure 7.1 for an example. The effects of tilting can be removed by rotating the landmarks as proposed in chapter 5.3, but the effects of rotating the head can not be solved with 2D landmarks, without assuming some additional information. When using 3D landmarks this problem could again be solved by minimizing the SSD between the 3D landmarks and a reference face.



(a) Rotation of head (b) Tilt of head

Figure 7.1: Head rotation vs. tilt

Landmarks over time

Another possibility for more accurate classifications might be the use of features over time. This might be of special interest when trying to forecast the activation of a certain Action Unit. This could then be used to reduce the time between the frame from the webcam stream and the taking of a photo.

Landmark accuracy

When all features are based on landmarks, it is only natural, that these should be as accurate as possible. For this there are mainly two possibilities, without using a different algorithm. The first option is to train a new model that has more landmarks, resulting in more features that might contain additional information. The second option is to train the model on more data.

Faster face finder

Being by far the slowest step in the framework, it should be replaced if webcams with more than 30 frames per second are available. This should also be beneficial when controlling cameras, as the latency between the input of a frame from a webcam stream and the taking of a photo would be reduced.

Multiclass SVM

Despite the good results of the SVM's trained, it would be optimal to have a multiclass classification. As already explained in section 5.6 linear SVM's are not suitable for this task when using *one-against-all* multiclass SVM. A different approach, the rank SVM, includes the "rank" of the data, which in our case would be the strength of activation. Using this method it should be possible to use a linear rank SVM, to obtain a multiclass classification as well as a fast prediction step.

Another different approach might be to just use binary search with the trained linear SVM's to obtain a multiclass classification (see figure 7.2).

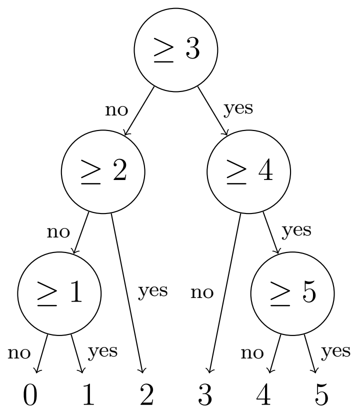


Figure 7.2: Possible solution for multiclass classification

Bibliography

[1] Camera Facialis, 2015. `http://www.zib.de/projects/camera-facialis`[Online; accessed 10-October-2016].

[2] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[3] Ekman and Friesen. FACS - Facial Action Coding System, 1987. `https://www.cs.cmu.edu/~face/facs.htm` [Online; accessed 10-October-2016].

[4] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[5] Itseez. Open Source Computer Vision Library. `https://github.com/itseez/opencv`, 2015.

[6] Vahid Kazemi and Josephine Sullivan. One Millisecond Face Alignment with an Ensemble of Regression Trees. In *CVPR*, pages 1867–1874. IEEE Computer Society, 2014.

[7] Davis E. King. Dlib-ml: A Machine Learning Toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009.

[8] Claudio Loconsole, Catarina Runa Miranda, Gustavo Augusto, Antonio Frisoli, and Verónica Costa Orvalho. Real-time Emotion Recognition - Novel Method for Geometrical Facial Features Extraction. In Sebastiano Battiato and José Braz, editors, *VISAPP (1)*, pages 378–385. SciTePress, 2014.

[9] Patrick Lucey, Jeffrey F. Cohn, Takeo Kanade, Jason M. Saragih, Zara Ambadar, and Iain A. Matthews. The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression. In *CVPR Workshops*, pages 94–101. IEEE Computer Society, 2010.

[10] The Qt Company. Qt. `https://www.qt.io/`[Online; accessed 10-October-2016].

[11] Wikipedia. Support vector machine — Wikipedia, The Free Encyclopedia, 2016. `https://en.wikipedia.org/w/index.php?title=Support\_vector\_machine\&oldid=743684334` [Online; accessed 10-October-2016].