Radboud University

The Performance Of Automated Facial Expression Analysis

Thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Artificial Intelligence

Author: M. van der Stam

Supervisors: Dr. F. Grootjen
Dr. P.N. Barkhuysen

June 9, 2017
Abstract

This research is about the performance of automated facial expression recognition. One of the goals was to examine whether an open source system can compete with a state of the art (and expensive) system. In order to do this a combination of open source systems DRMF2013 and WEKA was compared to FaceReader, by running them on several images of different data sets. The used pictures were divided in two categories; the first with frontal images only and the second with images where persons had their heads in positions between 45 degrees to the left and the same degrees to the other side. The results show that the developed open source system can compete with FaceReader, as it beats FaceReader when it comes to the group of frontal images and the images with a greater angel. However, when it comes to frontal images only, FaceReader beats the open source system. These findings are significant, as money could be saved in research projects that for example involve automated facial expression recognition.
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 Conclusion</td>
<td>26</td>
</tr>
<tr>
<td>5.1 All Images</td>
<td>26</td>
</tr>
<tr>
<td>5.1.1 Different expressions</td>
<td>26</td>
</tr>
<tr>
<td>5.2 FaceReader: Logarithmic Loss</td>
<td>27</td>
</tr>
<tr>
<td>5.3 Data Sets</td>
<td>28</td>
</tr>
<tr>
<td>6 Discussion</td>
<td>29</td>
</tr>
<tr>
<td>7 Future Work</td>
<td>30</td>
</tr>
<tr>
<td>A Database</td>
<td>34</td>
</tr>
<tr>
<td>A.1 Main</td>
<td>34</td>
</tr>
<tr>
<td>A.2 Database</td>
<td>35</td>
</tr>
<tr>
<td>B Matlab Files</td>
<td>39</td>
</tr>
<tr>
<td>B.1 Extractor Script</td>
<td>39</td>
</tr>
<tr>
<td>B.2 Extractor Function</td>
<td>41</td>
</tr>
<tr>
<td>B.3 Rdir</td>
<td>44</td>
</tr>
<tr>
<td>B.4 Convert gray scale to RGB</td>
<td>49</td>
</tr>
<tr>
<td>C Writing the arff file</td>
<td>50</td>
</tr>
<tr>
<td>C.1 Main</td>
<td>50</td>
</tr>
<tr>
<td>C.2 KeyPoints</td>
<td>53</td>
</tr>
<tr>
<td>C.3 Tuple</td>
<td>56</td>
</tr>
<tr>
<td>C.4 Emotion</td>
<td>58</td>
</tr>
<tr>
<td>C.5 Face</td>
<td>59</td>
</tr>
<tr>
<td>C.5.1 FaceKDEF</td>
<td>64</td>
</tr>
<tr>
<td>C.5.2 FaceMSFDE</td>
<td>66</td>
</tr>
<tr>
<td>C.5.3 FaceRaFD</td>
<td>68</td>
</tr>
<tr>
<td>C.6 WriteArff</td>
<td>70</td>
</tr>
<tr>
<td>C.7 Main</td>
<td>72</td>
</tr>
<tr>
<td>C.8 Emotion</td>
<td>74</td>
</tr>
<tr>
<td>C.9 FaceReader</td>
<td>75</td>
</tr>
<tr>
<td>C.10 PerformanceCalculator</td>
<td>83</td>
</tr>
<tr>
<td>C.11 FaceReaderState</td>
<td>90</td>
</tr>
<tr>
<td>C.12 PerformanceCalculatorState</td>
<td>97</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Facial expression is one of the non-verbal ways humans use to communicate with each other. Partly, those expressions are universal for people all over the world, even when they are isolated from the rest of the world[9]. The relation between someone’s facial expression and his or her emotions is still being questioned. However, systems that are able to perform an automatic analysis of facial expression can be of great usage. This type of software is already used to analyze how people respond to advertisements[17], mapping the displayed expressions to certain events in the advertisement. It could also be used as a tool that automatically adjusts the level of a training game, designed for example to help children diagnosed with autism. That way they wouldn’t get frustrated if it becomes too difficult or get bored if it is too easy.

There are multiple (expensive) systems on the market that can perform automated facial expression analysis such as Emotient, FaceReader[17] and Affectiva. Besides the expensive programs there are also open source landmark detection software systems available such as IntraFace, Chehra and Robust Discriminative Response Map Fitting (DRMF2013)[1]. As those software systems are landmark detection systems, they only detect certain points in the face but are not capable of telling the expression that comes with it. Besides that there are also online systems available such as clmtrackr. Apart from software that detects expressions or landmark points based on visual input, systems exists that can analyze emotions based on sound such as the ampath API.

In this bachelor thesis the performance of a cutting edge system will be compared with one of the open source systems. In order to do this, a reliable performance measurement had to be found, as well as trusted test data.
The cutting edge system that has been picked is called FaceReader and the open source landmark detection software is named Robust Discriminative Response Map Fitting (DRMF2013). The latter had to be combined with a classification tool, for which WEKA was chosen. Beforehand, the expectation was that FaceReader would outperform the combination of DRMF2013 and WEKA, because it is an established system. FaceReader is also used to validate one of the data sets used in this study. The overall agreement rate that was found in this study is 82\%\cite{3} However, if this would happen not to be the case, a free system that outperforms cutting edge software would have been found. This would be of great advantage for the scientific world, as research to facial expressions using an automated system would now have been made affordable.

In order to automatically analyze facial expressions, the face needs to be detected first\cite{15}. For humans it is easy to classify whether someone is a male or a female, of what age or race someone is, and even the expected professionalism\cite{15}. For a computer, the estimated number of features needed to identify someone grows logarithmically with the number of faces; in order to distinguish 1000 faces, 10 features are needed\cite{15}. Bledsoe, Chan and Bisson did the first work on facial recognition during 1964 and 1965\cite{16}. Bledsoe\cite{4} listed some difficulties for this problem such as the position of the head, aging, the lightning intensity, the angle and the facial expression.

To summarize it all; In this bachelor thesis the performance rate of a cutting-edge system called FaceReader and an open source system called Robust Discriminative Response Map Fitting (DRMF2013) along with a classification tool called WEKA are compared. In order to do this, data sets and a performance measurement were found and additional software is written. In this thesis among other things the most important considerations, some background information, a description of the methods used and of course the results are stated. Furthermore, in the appendix the code written for this study can be found.
Chapter 2

Facial Expression

2.1 Facial Expressions

The first research on facial expressions was done by Darwin, he wrote 'The expression of the Emotion in Man and Animals' (1872-1898) \[8\]. Darwin expected facial expressions to be universal and tried to find evidence for this hypothesis by analyzing observers’ responses to a variety of expressions in different countries using informants. Tomkins and McCarter\[20\] were the first to provide evidence that high agreement could be achieved by observers in judgement. This inspired Izard and Ekman to perform more cross-cultural research. They found evidence strongly suggesting that facial expressions are universal. For example, Ekman went to New Guinea to do research on the preliterate, isolated South Fore and found that they interpret facial expressions of emotions in a similar way. Furthermore there is unambiguous evidence that anger, disgust, happiness, sadness, fear and surprise are universal expressions. However, there are also cultural differences. Research has shown that when Americans and Japanese are alone watching a film, they display the same facial expressions. Yet when a person in authority was present, there was a difference; the Japanese smiled more and controlled their expressions more than the Americans\[9\]. This suggests that display rules are a source of the found cultural variation.

A number of studies have indicated that facial expressions don’t correlate with the experience of emotions\[8\], on the other hand they do correlate with different patterns of central nervous system activity. An important obser-
vation for this research is that people can differentiate between spontaneous and posed expressions. Studies have shown that this is the case for among others embarrassment, amusement, love and desire. Despite the focus on facial expression in this study, the face is not the only source of information when it comes to decoding emotions. Other factors are for example: speech, body cues, the given task and the conditions in which the behavior occurred. Research has shown that other channels are often used and that the used channel differs along the task. Furthermore, it has shown that what has been said matters more than the facial expressions itself.

2.1.1 FACS

Dr. P. Ekman did a lot of research to muscle contractions in different expressions. Together with W. Friesen they developed the Facial Action Coding System (FACS), which was first published in 1978. The system was built on work of the anatomist Hjorstjo (1970), identifying action units based on muscle groups in the face. Based on this system, companies like Emotient (in which Ekman is involved), started implementing the analysis of facial expressions. In 2002 a new manual was published via which people can learn FACS and can even get FACS certified by passing a test.

Ekman and Friesen created FACS from the philosophy of anatomical basis of facial movement, not per se being the result of an emotion. Before the development of this system, there were others that included anatomy in their systems but never exclusively. FACS, Facial Action Coding System is purely based on measurement of the face, excluding everything else such as crying, sweating but also visible changes in muscle tonus that do not entail movement. Where muscle tonus is the continuous passive partial contraction of the muscles, that could be picked up by Electromyography or EMG. The advantage of this method over using the information that observers were able to infer from a face, is that this method is more reliable. This is true as observers can be influenced by for example contextual knowledge or differences in culture, in other words their judgement is subjective whereas FACS tries to be objective.

2.1.2 Basic Emotions

One outstanding difference between various facial expression recognition systems, and also between several data sets, is the number of different emotions
that systems can classify or that are included in data sets. This is probably
due to the fact that there is no consensus among scientists about what basic
emotions are. For example Ekman, Friesen and Ellsworth think of anger,
disgust, fear, joy, sadness and surprise to be the basic emotions, where for
example Weiner and Graham think only happiness and sadness are basic
emotions[13].
FaceReader is able to classify expressions that resemble the basic emotions
as listed by Ekman, Friesen and Ellsworth[10] above. Be that as it may, Fac-
eReader does not list their work on their website so it is unclear whether this
decision is based on their work. As for this study, the used facial expressions
are taken from FaceReader together with the statement of Ekman, Friesen
and Ellsworth that those are the basic emotions; happy, neural, sad, angry,
surprised, scared and disgusted.
Chapter 3

Method

In this research a comparison is made between a cutting-edge system called FaceReader and a pipeline developed in this research. That pipeline consists of an open source system that detects landmark points in the face (DRMF2013), which is then classified into different categories of expressions using a classification tool named WEKA. In order to do this, three different data sets with pictures of faces that were already annotated by people were used. The golden standard is set to be those annotated pictures. That means that the two software systems are compared to each other using this golden standard to determine the score and a performance measurement.

The expressions FaceReader is able to distinguish are used as a standard, meaning that the developed pipeline is also able to classify those specific expressions. This way the comparison regarding the performance between the two systems can be made. For this reason images where people have expressions not listed in chapter 2 are not taken into account when it comes to calculating the score. After the output of the two systems was calculated, a performance measurement (Logloss and percentages) of those results was computed and compared. See 3.1 for a graphical representation of the research.

The remainder of this chapter is structured according to the order of the study. Starting by discussing the different data sets, moving on to the two different systems and finishing with the performance rate.
3.1 Data Sets

In this study three different data sets are used: DRMF\textsuperscript{[1]}, KDEF\textsuperscript{[11]} and Radboud Faces\textsuperscript{[3]}. They are all created by researchers. In this section information about each of them is listed, together with a summarizing table with the most important characteristics.

An essential remark is that not all the images in the data sets are used. This is due to the angles of the heads of the people in the pictures, as they are sometimes too big for the software to be able to analyze. Because it is known beforehand that the angles are relevant for analyzing expressions in pictures, two pools of images are created with some overlapping images. The pictures where the person is looking straight in the camera are copied into a special folder. After that, a second folder is created filled with pictures where the head positions are within an angle of 45 degrees in both directions. In total 7986 different images are used, 2812 of those pictures are frontal images. The implementation of the creation of those two sub data sets can be found in appendix \textsuperscript{A}.

For research purposes, it is important to have an annotated data set with
images of people expressing different facial expressions. There should be variation in facial characteristics such as the expressions itself, gaze direction, head orientation, different sexes and ages (children/adults). Ideally this data set consists of pictures with controlled technical factors including for instance lighting conditions and facial landmarks. The participants of whom the pictures are taken displayed a variety of characteristics, amongst them are men, woman, children, people of different ethical backgrounds etcetera.

3.1.1 Radboud Faces

Radboud Faces (RaFD) \[3\] is a free data set consisting of 49 Caucasian models who vary along the characteristics mentioned above. Amongst the models there were 39 adults (19 females) and 10 children (6 females). The pictures were taken in a highly controlled environment, meaning that the technical factors were regulated. The models presented eight different expressions (sadness, neutral, angry, contempt, disgust, surprise, fear and happiness) with three different gaze directions (front, left and right), from five different camera angels. The expressions were based on prototypes of FACS\[7\]. After the pictures were taken, they were first rated on attractiveness in order to familiarize the participants with the models. Then they were rated on facial expression, intensity of the expression, clarity, valence and genuineness of the expression. For each model two images were present in the database for each combination of gaze direction and facial expression. Only the ones with the highest agreement on the expressed and intended emotion were included in the final database. Each image was rated by at least 20 participants.

The agreement between the chosen and the target expressions was on average 82% (SD = 19%). The standard deviation can be explained by amongst other things the difference between happiness (M = 98%, SD = 3%) and contempt (M = 50%, SD = 15%). The lower mean of contempt could be explained by the fact that this emotion is less universally recognized across cultures\[6\]. Finally, the most frequent alternative for surprise - disgust and fear - had morphological overlaps\[3\]. In fig. 3.2 some example pictures from the Radboud Faces data set are presented.

3.1.2 KDEF

The Karolinska Directed Emotional Faces database\[11\] consists of 4900 pictures of 70 different people. The models were all adults between 20 and 30
3.1. DATA SETS

CHAPTER 3. METHOD

(a) Disgusted, frontal  (b) Surprised, frontal  (c) Sad, left  (d) Disgusted, left

Figure 3.2: A sample of images from the Radboud Faces database

years old. Amongst the models were 35 females and 35 males. The participants were asked to display 7 different expressions, which they had practiced at home: neutral, happy, angry, afraid, disgusted, sad and surprised. The expressions were photographed from 5 different angles. All participants wore gray T-shirts and did not have a beard, earrings or eyeglasses etc. They were all seated at approximately three meters from the camera and the light was taken care of as well. More specific information can be found in an ”about” file on their website[11].

3.1.3 MSFDE

The Montreal Set of Facial Displays of Emotion[2] is a database that consists of 224 images of 32 different people. There are pictures of 4 different ethical backgrounds: African, Asian, Caucasian and Hispanic. For every ethnic background there were four males and four females expressing six different emotions: anger, joy, sadness, fear, disgust and shame. The expressions were created using a directed facial action task and were FACS coded, so that identical expressions across actors were the same. MSFDE consists of gray scaled images. This causes a problem for DRMF2013 as this color scheme converts to a matrix with other dimensions than images with a RGB color scheme. For this reason all the images are converted to RGB scaled images, not changing anything about the pictures itself. The implementation for changing the color schemes of the pictures can be found in appendix[3].
3.2. SOFTWARE TOOLS

3.2.1 FaceReader

FaceReader is a commercial facial expression analysis software package, created by Noldus[17]. In this study version 5 is used. This analyzes seven different expressions: neutral, happiness, sadness, angry, surprise, fear and disgust. The analysis is based on 500 key points in the face. FaceReader also takes into account gaze direction, head orientation and person characteristics. The software is trained on 10,000 manually annotated images. It scores
95.9% agreement on happiness[^3], comparing the scores to manual scores from professional annotators. It is possible to analyze images, videos and a web-cam stream. The software is used both in research and the commercial field, for example to study the effect of commercials.

In this study the pictures are loaded into FaceReader and analyzed using the "batch analysis" option. An example of this analysis can be found in fig. 3.3. The results are then exported into text files. This yields into two types of files: one with detailed information (a score per picture for every expression) and one called "state" (with only the 'winning' expression per picture). To further process these detailed results, software is written that reads those scores, normalizes them and stores them. In order to compare different aspects to each other, several averages are calculated. For each data set its total score and the score per expression are calculated. Besides that, the average score of all the pictures is calculated, as well as the total average score per expression.

The first step FaceReader takes, is to "fit" a face, in other words find for example the landmark points. However, when this is not possible, the detection of the expression for that picture fails. This happens for instance when the quality of a picture is not right or when the angle of the head of the person in the picture is too great. In case of a "fit fail", there is no score from FaceReader, thus $1/7e$ (chance-level) is added. In order to keep track of how often this occurs, the number of times is again calculated per database and overall. The exact implementation can be found in appendix C.6.
3.2. SOFTWARE TOOLS

3.2.2 DRMF2013

DRMF2013 is a software package in MATLAB that detects 66 landmark points on the face and estimates the rough 3D head pose. This is a new regression based approach for Constrained Local Models called Discriminative Response Map Fitting. The method outperforms state-of-the-art algorithms for face fitting\[1\]. It is computationally efficient (1 second per image on an Intel Xeon 3.80 GHz processor) and the system is real-time capable. The basic idea behind the method is that it is possible to learn robust functions from response maps to shape parameter updates.

First a dictionary is trained for response map approximations, that can be used for extracting the relevant features for learning the fitting update model. Then the fitting model is iteratively learned by modified boosting procedure. The method is a performance gain over other methods. For the algorithm itself and the mathematics behind it, see Asthana et al.\[1\].

DRMF only takes images as input (although it is real-time capable). It works for a range in head position of thirty degrees in both directions. It is also tested that the system handles challenging uncontrolled natural variations.

In this study the images processed by FaceReader are also processed by DRMF2013. As might be expected from free open source software, the functionality is such that more (technical) skills are required to use it. For exam-
Additional software is written that recursively processes all the files with a certain extension from a given location. The output, which among other things consists of 66 landmark points in the face, is written to a specified location. As this is needed for all the used data sets, a script is written to do previous for all data sets. The implementation can be found in appendix B.

As DRMF2013 only detects 66 landmark points and the position of the face, an additional tool needs to be used in order to classify the pictures to expressions. In fig. 3.4 the output of DRMF2013 is showed by the red dots.

![Image of Mona Lisa with facial landmarks](image.png)

Figure 3.4

### 3.2.3 WEKA

WEKA\[12] is an open source software package for data mining tasks. It was developed by the machine learning group of the university of Waikato. Multiple different machine learning algorithms are incorporated in the tool. There are two different ways to use WEKA, the first one is in an application and the second one is by calling it from Java code, as the tool is Java based. Before the tool can be used an ARFF-file has to be written.
ARFF-file

An ARFF-file (Attribute-Relation File Format file), is an ASCII text file that describes a list of instances sharing a set of attributes. In this case the attributes are the different landmark points, where every coordinate has two landmark points: one for the x-coordinate and one for y-coordinate. The instances are listed in the output of DRMF2013 as at are the specific coordinates for every landmark point on every specific face. As the data sets are split into two categories, one with only frontal images and one with all images, two different ARFF-files are constructed. However, before the ARFF-files are constructed the landmark points are normalized, as the size of a face differs between the participants. The normalization is performed by calculating a factor for every face. This is done by taking the average of two mirrored lengths in the face. The first one is the distance between highest jawline point next to the left ear to the tip of the nose (a) and the second one is mirrored to the previous one (b). For clarification of these two lengths that are averaged to become the factor see the green lines in fig. 3.4. Using this particular factor, the normalization is performed by dividing all landmark points by this factor. After the normalization the ARFF-file is created, for the implementation of constructing the ARFF-files see appendix C.

Next the ARFF-files are loaded into WEKA. All the classifiers WEKA supports are used on the data, and the classifier with the highest performance is chosen. It turns out that the Multilayer Perceptron with the back propagation algorithm, gives the best result for this classification problem. The GUI is used with the default options for the Multilayer Perceptron, along with a cross-validation of ten folds. In fig. 3.5 a simplistic visual representation of a multilayer perceptron network with back propagation is displayed. The input of the network consists of the data, the images accompanied by their displayed expression, as can be found in WEKA. There is at least one hidden layer in the network and an output layer which consists of the possible expressions. An activation function is used to determine the output of the nodes. The error is calculated and then propagated backwards into the network (the red arrow in fig. 3.5), with the objective to minimize this error.
3.3 Performance Rate

FaceReader displays a distribution over the expressions for a score. As the images are annotated to be one specific expression, a comparison has to be made between a distribution on one hand, and a 'binary distribution' on the other hand. In order to do so, the Logarithmic Loss\cite{14} function is used:

\[
\text{logloss} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} \log_2(p_{ij})
\]

The formula is used the following way; first there is an iteration over all the images (1 to N). According to the formula, there has to be an iteration over all the expressions (1 to M) after that, however, the correct expression is known beforehand as the images are annotated. Thus there is no need to implement that. Instead, the logarithm of the score belonging to the correct expression is taken, this way \( y_{ij} \) would always be one. All those scores are summed and in the end divided by the number of images. Because the logarithm of the score is taken, the best possible score is zero (as the logarithm of 1 is zero). The worst possible score would be chance level, in this case there were seven possible expressions meaning that the worst possible score would become \(-\log_2(1/7)\), which is about 2.8.
3.3. PERFORMANCE RATE

CHAPTER 3. METHOD

3.3.1 Entropy

Cross-Entropy

Logarithmic Loss is based on a technique coming from machine learning called Cross-Entropy. This technique is used in for example optimization problem solving, rare event stimulation, performance analysis of telecommunication systems and artificial neural networks[18]. The measurement is used as an alternative error function to squared error in neural networks. In terms of the network, the formula can be used when the output of the nodes represent independent hypotheses. In this study, it could be thought of granting the expressions to different output nodes in the 'network' (the software systems). The activations of the nodes can be thought of as representing the probability that each hypothesis might be true. In this study, the output of the systems represents a probability distribution. The cross-entropy error measurement measures the distance between what the system believes the distribution should be and what the annotated images say it should be.

Information Theory

Information theory is the topic of measures that are related to how unexpected an event is, used in for example complexity measurements. In the information theory lays the mathematical foundation of the logarithmic loss formula. A usable measure for the information obtained by observing the occurrence of an event with probability p is developed, in other words the information in terms of probability p. There are four properties that have to be satisfied:

1. \( I(p) \geq 0 \)
   Information can not be negative

2. \( I(p_1 * p_2) = I(p_1) + I(p_2) \)
   For two independent events whose joint probability is the product of their individual probabilities

3. \( I(1) = 0 \)
   If the probability of the event is 1, no information is obtained from occurrence
3.3 PERFORMANCE RATE

4. $I(p)$ is monotonic and continuous in $p$

This way small changes in the probability result in small changes in the information.

Using the previous properties the following can be derived:

5. $I(p^2) = I(p \ast p) = I(p) + I(p) = 2 \ast I(p)\{r2\}$

6. $I(p^n) = n \ast I(p)\{r1, induction\}$

7. $I(p) = I((p^{1/m})^m) = m \ast I(p^{1/m})$

   $I(p^{1/m}) = \frac{1}{m} \ast I(p)$ in general:

   $I(p^{n/m}) = \frac{n}{m} \ast I(p)\{r6\}$

8. for $0 < p \leq 1$ and $a > 0$:

   $I(p^a) = a \ast I(p)\{r4\}$

9. $I(p) = -\log_b(p)$

For base $b$, two was chosen as that is common in informatics. As the score of all images is required, the average is taken. The formula then ends up looking like this: $I(p) = \frac{1}{N} \sum_{i=1}^{N} -\log(p)$, which is what is used in this study.

3.3.2 Percentages

As stated above, the logarithmic loss function is a proven method for measuring performance. However, due to the nature of the way WEKA classifies the features using the multilayer perceptron, it is not possible to use this performance measure. The results of this classifier are discrete as it is a unary classification system, where the results of FaceReader are stochastic of nature as this system performs probability classification. Because the results of those discrete values (WEKA) in the Logloss formula would yield 0 or infinite, it can’t be used to compare both systems. For this reason, the main comparison is done using percentages. However, for FaceReader the performances are also calculated using the Logloss function, they can be found in section 4.1.2.
Chapter 4

Results

In this section the performances of the two systems can be found. As mentioned before they are created by running the software on three different data sets.

In table 4.1 an overview of the results can be found. The difference between the open source software system and FaceReader is stated, as well as the difference between processing only the frontal images versus processing all the images in the database. In fig. 4.1 a graphical overview of these results is presented.

<table>
<thead>
<tr>
<th>Emotion</th>
<th>Open Source Frontal</th>
<th>FaceReader Frontal</th>
<th>Open Source All</th>
<th>FaceReader All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neutral</td>
<td>64%</td>
<td>98%</td>
<td>53%</td>
<td>72%</td>
</tr>
<tr>
<td>Happiness</td>
<td>94%</td>
<td>94%</td>
<td>88%</td>
<td>61%</td>
</tr>
<tr>
<td>Sadness</td>
<td>43%</td>
<td>88%</td>
<td>39%</td>
<td>43%</td>
</tr>
<tr>
<td>Anger</td>
<td>69%</td>
<td>84%</td>
<td>60%</td>
<td>46%</td>
</tr>
<tr>
<td>Surprise</td>
<td>89%</td>
<td>96%</td>
<td>82%</td>
<td>36%</td>
</tr>
<tr>
<td>Fear</td>
<td>62%</td>
<td>80%</td>
<td>55%</td>
<td>30%</td>
</tr>
<tr>
<td>Disgust</td>
<td>80%</td>
<td>87%</td>
<td>73%</td>
<td>37%</td>
</tr>
<tr>
<td>Total</td>
<td>71%</td>
<td>89%</td>
<td>64%</td>
<td>47%</td>
</tr>
</tbody>
</table>

Table 4.1
4.1 FaceReader

In this section the results that came from FaceReader and are analyzed using the logarithmic loss function can be found. In line with the rest of the study, these results have been split as well into two different cases; frontal images only and all images selected from the data sets. Furthermore, the results have been split per data set and per emotion.

4.1.1 Frontal Faces

In table 4.2 the results for the frontal images can be found. Besides a performance rate per emotion per data set, total performance rates are also calculated. Additionally the total number of images used and an important
4.1. FACEREADER

CHAPTER 4. RESULTS

component namely the number of “fit failed” is stated. In fig. 4.2 a graphical overview has been created.

<table>
<thead>
<tr>
<th></th>
<th>Happy</th>
<th>Neutral</th>
<th>Sad</th>
<th>Angry</th>
<th>Surprised</th>
<th>Scared</th>
<th>Disgusted</th>
<th>Total</th>
<th>N</th>
<th>#Fit Failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>KDEF</td>
<td>0.331</td>
<td>0.100</td>
<td>0.726</td>
<td>0.874</td>
<td>0.227</td>
<td>1.329</td>
<td>0.760</td>
<td>0.621</td>
<td>980</td>
<td>14</td>
</tr>
<tr>
<td>MSFDE</td>
<td>0.855</td>
<td>0.575</td>
<td>2.061</td>
<td>1.759</td>
<td>-</td>
<td>1.402</td>
<td>1.375</td>
<td>1.338</td>
<td>224</td>
<td>27</td>
</tr>
<tr>
<td>RaFD</td>
<td>0.341</td>
<td>0.038</td>
<td>0.229</td>
<td>0.470</td>
<td>0.179</td>
<td>0.532</td>
<td>0.450</td>
<td>0.323</td>
<td>1608</td>
<td>9</td>
</tr>
<tr>
<td>Total</td>
<td>0.381</td>
<td>0.107</td>
<td>0.732</td>
<td>0.732</td>
<td>0.199</td>
<td>0.917</td>
<td>0.646</td>
<td>0.512</td>
<td>2812</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 4.2: Results FaceReader: frontal images

Figure 4.2: FaceReader: overall performance of frontal images

4.1.2 All Images

In this section the results for FaceReader have been displayed. In table 4.3 the results are displayed per data set and per emotion. Additionally the number of images that is tested and the number of fit failed per data set is shown. In fig. 4.3 a graphical overview of table 4.3 is visible.
### 4.1. Facereader

#### 4.1.3 Fit Failed

As the number of cases where Facereader displayed "Fit Failed" is important for the performance of Facereader, a graphical overview is created. In fig. 4.4, the percentage is displayed per data set, split for the two image categories in this study.

<table>
<thead>
<tr>
<th></th>
<th>Happy</th>
<th>Neutral</th>
<th>Sad</th>
<th>Angry</th>
<th>Surprised</th>
<th>Scared</th>
<th>Disgusted</th>
<th>Total</th>
<th>N</th>
<th>#Fit Failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>KDEF</td>
<td>1,156</td>
<td>0.995</td>
<td>2,103</td>
<td>2,151</td>
<td>1,993</td>
<td>3,595</td>
<td>2,204</td>
<td>2938</td>
<td>893</td>
<td></td>
</tr>
<tr>
<td>MSFDE</td>
<td>0.855</td>
<td>0.575</td>
<td>2,061</td>
<td>1,759</td>
<td>-</td>
<td>1,492</td>
<td>1,375</td>
<td>1,338</td>
<td>224</td>
<td>27</td>
</tr>
<tr>
<td>RaFD</td>
<td>1,405</td>
<td>0.923</td>
<td>1,876</td>
<td>1,640</td>
<td>1,895</td>
<td>2,268</td>
<td>2,676</td>
<td>1,863</td>
<td>4824</td>
<td>1631</td>
</tr>
<tr>
<td>Total</td>
<td>1,289</td>
<td>0.941</td>
<td>1,972</td>
<td>1,847</td>
<td>1,936</td>
<td>2,976</td>
<td>2,939</td>
<td>1,986</td>
<td>7986</td>
<td>2551</td>
</tr>
</tbody>
</table>

Table 4.3: Results Facereader: all images

![Performance All Images](image)

Figure 4.3: Facereader: overall performance all images
4.1.4 Differences between the data sets

In fig. 4.5 the averages per data set are shown, again split for the two categories of pictures that are made in this study.
Chapter 5

Conclusion

5.1 All Images

Taking all images into account, without distinguishing the different expressions, the open source system performs better than FaceReader (62% correctly classified to 47%). However, only taking the subgroup of frontal images into account, FaceReader performs better than the open source system (89% to 71%). Concluding from the results, it depends on the angle of the face in the picture which software system performs best. A possible explanation for the difference in performance, comes from how FaceReader works. FaceReader starts by 'fitting' the face, finding the important spots in the face. When the quality of the picture is not good enough or when the angle of the head is too great this fails and that will cause FaceReader not to give an expression as output at all. However, the influence of the previous should not be magnified, as in this study in those case chance level was added instead of zero. Furthermore, in fig. 4.4 it is shown that for frontal images specifically the total percentage of fit failed is only 1%, however considering all images this percentage rises to 26%.

5.1.1 Different expressions

Again considering fig. 4.1 there are great differences amongst the percentage correctly classified images between different expressions. This variety is better visible in fig. 5.1 where the standard deviations per class (system
Figure 5.1: Percentages of Correctly Classified Images with standard deviation combined with images) are depicted. The standard deviation for the Open Source System for all images is 17% and for frontal images only is 18%. The standard deviation for FaceReader for all images is 15% and for frontal images only is 7%. This shows that FaceReader is a more stable system when it comes to frontal images only.

An explanation for the differences in percentages between the expressions, might be found in the fact that for example anger is for humans also less easy recognizable than happiness.

### 5.2 FaceReader: Logarithmic Loss

In fig. 4.2 and fig. 4.3 the performance of FaceReader is shown using the Logarithmic Loss algorithm where the worst case scenario is 2.8 and best case scenario is 0. These figures show again that the performance of FaceReader is worse when it comes to all images compared to only frontal images.

These graphs show the differences between the expressions similar to what
is discussed previously.

5.3 Data Sets

Interestingly fig. 4.5 shows that for data sets KDEF and Radboud Faces there is a difference between frontal pictures and the set including also the angled pictures. As expected the performance of FaceReader and those data sets is better for the frontal images. However, FaceReader shows no difference in performance comparing those classes of images for MSFDE, the reason for this is that there is no difference whatsoever between the two classes, as MSFDE only consists of frontal images.
Chapter 6

Discussion

In this research three different data sets were used to create two pools of annotated images: one with only frontal images and one with images with a rotation of the head of minus 45 degrees to plus 45 degrees. On these images two software systems were tested; one commercial system called FaceReader and one open source system that was supplemented by a classification tool called WEKA and completed with software written by the authors. In advance of the research the hypothesis was that the commercial system would outperform the open source system. In the end, this was the case for frontal images, but taking all the images into account the open source system outperformed FaceReader.

However, looking back at the performed research, the comparison might not have been completely fair. Partly the created pipeline is solely trained and tested on the data sets used in this research. This is true for WEKA, but not for DRMF2013 and might have given this system an advantage over FaceReader. On the other hand, it is impossible to know on what pictures FaceReader or DRMF or trained, so that a strictly fair competition is virtually impossible.

On the subject of fair comparison another matter arises, FaceReader is able to take the position of someone’s face into account. The in this research developed pipeline doesn’t take this into account, that is, DRMF2013 is calculating some parameters but these are not taken into account when an image is classified. This might give FaceReader an advantage.
Chapter 7

Future Work

A suggestion for future work is taking into account the head positions that DRMF2013 is already calculating. This could be done by either normalizing this factor, as is done with the sizes of the heads, or it could be made an important attribute in WEKA.

One of the applications mentioned in the introduction is that the automatic analysis of expressions could be used to adjust the level of a training program based on this expression. If this should be made possible on portable devices such as a smart-phone or tablet, the software needs to be adjusted to these devices and therefore probably have to become more efficient because reducing the processing times is necessary. Also, in this study the tested systems are applied to static pictures, for this application they would need to be able to analyze faces real-time.

Furthermore, the systems that are used in this thesis, FaceReader and DRMF2013, are not capable of analyzing expressions when angle of head is too great. In order for this to be solved, an expression would have to be recognized by analyzing for example only half of the face. Although you could wonder whether this is necessary for applications.

Using WEKA the results of DRMF2013 are not classified as a distribution over the faces, but as one class only. It would be a great improvement if an extension would be programmed that classifies the expressions in a probabilistic way. Additionally the network architecture and parameters can be adjusted towards this specific problem, instead of using the default options. Building on the previous suggestion, applying the algorithm called Logarithmic Loss to the open source system as well would ensure a fairer comparison. Additionally, in order to gain more understanding of the figures
resulting from calculating the performance using the Logarithmic Loss algorithm, there could be thought of a way to present these numbers in a better way.


Appendix A

Database

A.1 Main

```java
package database;

/**
 * @author Margo van der Stam
 */

public class Main {

    public static void main(String[] args) {
        new Database();
    }
}
```
A.2 Database

```java
package database;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Collection;
import static java.nio.file.StandardCopyOption.*;
import org.apache.commons.io.FileUtils;

/**
 * Class for selecting images of people within a certain angle from the data
 * sets Radboud Faces and KDEF
 *
 * @author Margo van der Stam
 *
 */

public class Database {
    static Collection<File> files;
    static File path;

    public Database() {
        path = new File("C:\Users\Margo\Documents\Universiteit\Bachelor_ThesisNG\Databases\");
        files = getFiles();
        loopFiles();
    }

    private static Collection<File> getFiles() {
        String[] filetypes = { "png", "jpg" };
        Collection<File> files = FileUtil.listFiles(path, filetypes, true);
        return files;
    }
}
```
return files;
}

/**
 * Loops over all files and checks to which database
 * a file belongs
 */
private static void loopFiles() {
    for (File file : files) {
        String path =
            file.getAbsolutePath().split("\\\")[7];
        switch (path) {
            case "KDEF":
                checkFileKDEF(file);
                break;
            case "Database_RaFD":
                checkFileRaFD(file);
                break;
        }
    }
}

/**
 * @param file
 * of the Radboud Faces database Checks
 * whether the image is a
 * frontal image, or -45 or 45 degrees,
 * and if so copies it to
 * another folder
 */
private static void checkFileRaFD(File file) {
    StringBuilder name = new
        StringBuilder(file.getName().split("_")[0]);
    Path newDir =
        Paths.get("C:\Users\Margo\Documents\Universiteit\Bachelor\DatabasesFrontal\RaFD");
    Path source =
        Paths.get(file.getAbsolutePath());
if (name.charAt(5) == '9') {// //
    name.charAt(5) == '4' //
  //
  // name.charAt(5) == '3')
  //
  // The used
  // images
  // in the
database
  are
coded
  with
  //
  // 045/090/

try {
    try {
        Files.copy(source,
            newDir.resolve(source.getFileName()),
            REPLACE_EXISTING);
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

/**
   * @param file of the KDEF database Checks whether the
   * image is a frontal
   * image and if so copies it to another
   * folder
   */
private static void checkFileKDEF(File file) {
    String name = file.getName();
    Path newDir = Paths.get("C:\\Users\\Margo\\Documents\\Universiteit\\Bachelor\\DatabasesFrontal\\KDEF");
    Path source = Paths.get(file.getAbsolutePath());
    if (name.charAt(6) == 'S') {
        try {
            Files.copy(source, newDir.resolve(source.getFileName()), REPLACE_EXISTING);
        } catch (IOException e) {
            e.printStackTrace();
        }
    } else if (name.charAt(6) == 'H') {
        try {
            Files.copy(source, newDir.resolve(source.getFileName()), REPLACE_EXISTING);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
Appendix B

Matlab Files

B.1 Extractor Script

%% Script for running different databases, using the Extractor Function and saving the processing time. %
% ----------------------------------------------- %
% Script_Extractor
% by Margo van der Stam
% 20 - 5 - 2016
% ----------------------------------------------- %

time = zeros(3);

%Radboud Faces
time (1) = Extractor_Function( 0, 0, '*.jpg',
    'C:\Users\Margo\Documents\bachelor-thesis\Output\Matlab_Frontal\RaFD',
    'C:\Users\Margo\Documents\MATLAB\DatabasesFrontal\RaFD');

%KDEF
time (2) = Extractor_Function( 0, 0, '*.jpg',
    'C:\Users\Margo\Documents\bachelor-thesis\Output\Matlab_Frontal\KDEF',
    'C:\Users\Margo\Documents\MATLAB\DatabasesFrontal\KDEF');

%MSFDE
time (3) = Extractor_Function( 0, 0,
    '*.png','C:\Users\Margo\Documents\bachelor-thesis\Output\UsedImages\MSFDE\',
    'C:\Users\Margo\Documents\MATLAB\DatabasesUsed\MSFDE\RGBvalues\RGBvalues\');
B.2 Extractor Function

function [ mean_time ] = Extractor_Function( bbox_method, visualize, file_type, export_path, image_path )
%Extractor_function extracts and processes all the files with
% a certain
% fileType from all the nested maps from a certain path.
% bbox_method: see down below
% visualize: 0 if image isn't displayed
% file_type: The filetype of the images in the map
% export_path: Path where the processed images are stored
% image_path: Path of the 'highest' map your images are in
%
% Uses the function rdir to find all files in the deepest
% maps

% ----------------------------------------------- %
% Extractor_Function.m (modded Demo.m, modded Extractor.m)
% by Margo van der Stam
% 20 - 5 - 2016
% ----------------------------------------------- %
%
clear; close all;
addpath(genpath('.'));

% ----------------------------------------------- %
% % bbox_method:
% % Choose Face Detector
% % 0: Tree-Based Face Detector (p204);
% % 1: Matlab Face Detector (or External Face Detector);
% % 2: Use Pre-computed Bounding Boxes
% %
% NOTES:
% % [a] Option '0' is very accurate and suited for faces
% in the 'wild';
% % But it is EXTREMELY slow!!!
% % [b] Option '1' supports the functionality for
% incorporating
% % % YOUR OWN FACE DETECTOR WITH DRMF FITTING;
% % % Simply modify the function
% External_Face_Detector.m
% % %
%-------------------------------------------------%
%-------------------------------------------------%
% % % Load DRMF Model
clm_model='model/DRMF_Model.mat';
load(clm_model);

% % % Start time
time = 0;
tic

% % % Iterate over images
path = strcat(image_path, '**', file_type);
img_list = rdir(path);

for i=1:length(img_list)
clear data

[path,name,ext] = fileparts(img_list(i).name); %split name file
targetfile = strcat(export_path, name, '.txt');

disp(strcat('targetfile:', targetfile));
disp(sprintf('%d/%d', [i, size(img_list,1)]));
if ~exist(targetfile, 'file') == 2
    disp(sprintf('Calculating DRMF for %s',
        img_list(i).name));
data(1).name = img_list(i).name;
data(1).img = im2double(imread([img_list(i).name]));

    % % % Required Only for bbox_method = 2;
data(1).bbox = [] % Face Detection Bounding Box
    [x;y;w;h]
% Initialization to store the results
data(1).points = [];
% MAT containing 66 Landmark Locations

data(1).pose = [];
% POSE information [Pitch;Yaw;Roll]

data = DRMF(clm_model, data, bbox_method, visualize);

time = time + toc;

%----------write results to targetfile----------
if (~isempty(data(1).points))
    disp(sprintf('Write %s', targetfile));
    fileID = fopen(targetfile, 'w');
    fprintf(fileID, '%s
', 'markers (66x2)');
    fprintf(fileID, '%4.6f %4.6f
', data(1).points);
    % transpose is extremely important
    fprintf(fileID, '%s
', 'pose (pitch yaw roll)');
    fprintf(fileID, '%4.6f %4.6f %4.6f
', data(1).pose);
    fclose(fileID);
else
    disp(sprintf('%s exists, skipping...', targetfile));
end

mean_time = time / size(img_list, 1);

end
B.3  Rdir

function [varargout] = rdir(rootdir,varargin)
% Lists the files in a directory and its sub directories.
% % ----------------------------------------------- %
% rdir
% by Gus Brown
%  (https://nl.mathworks.com/matlabcentral/fileexchange/19550-recursive-directory-listing/content/rdir.m)
% 11 - 4 - 2008
% ----------------------------------------------- %

% function [D] = rdir(ROOT,TEST)
%
% Recursive directory listing.
%
% ROOT is the directory starting point and includes the
% wildcard specification.
% The function returns a structure D similar to the one
% returned by the built-in dir command.
% There is one exception, the name field will include
% the relative path as well as the name to the file that
% was found.
% Pathnames and wildcards may be used. Wild cards can exist
% in the pathname too. A special case is the double * that
% will match multiple directory levels, e.g. c:\**\*.m.
% Otherwise a single * will only match one directory level.
% e.g. C:\Program Files\Windows *
%
% TEST is an optional test that can be performed on the
% files. Two variables are supported, datenum & bytes.
% Tests are strings similar to what one would use in a
% if statement. e.g. 'bytes>1024 & datenum>now-7'
%
% If not output variables are specified then the output is
% sent to the screen.
%
% See also DIR
%

44
% examples:
%  D = rdir('*.m');
%  for ii=1:length(D), disp(D(ii).name); end;
%
%  % to find all files in the current directory and sub
directories
%  D = rdir('**\')
%
%  % If no output is specified then the files are sent to
%  % the screen.
%  rdir('c:\program files\windows \*.exe');
%  rdir('c:\program files\windows \**\*.dll');
%
%  % Using the test function to find files modified today
%  rdir('c:\win\*','datenum>floor(now)');
%  % Using the test function to find files of a certain size
%  rdir('c:\program files\win\*.exe','bytes>1024 &
           bytes<1048576');
%
% use the current directory if nothing is specified
if ~exist('rootdir', 'var'),
  rootdir = '*';
end;

% split the file path around the wild card specifiers
prepath = '';       % the path before the wild card
wildpath = '';      % the path wild card
postpath = rootdir; % the path after the wild card
I = find(rootdir==filesep,1,'last');
if ~isempty(I),
  prepath = rootdir(1:I);
  postpath = rootdir(I+1:end);
  I = find(prepath=='*',1,'first');
  if ~isempty(I),
    postpath = [prepath(I:end) postpath];
    prepath = prepath(1:I-1);
    I = find(prepath==filesep,1,'last');
end;

45
if ~isempty(I),
    wildpath = prepath(I+1:end);
    prepath = prepath(1:I);
end;
I = find(postpath==filesep,1,'first');
if ~isempty(I),
    wildpath = [wildpath postpath(1:I-1)];
    postpath = postpath(I:end);
end;
end;
end;

% disp([' "' prepath '" ~ "' wildpath '" ~ "' postpath '"
        ']);

if isempty(wildpath),
    % if no directory wildcards then just get file list
    D = dir([prepath postpath]);
    D([D.isdir]==1) = [];
    for ii = 1:length(D),
        if (~D(ii).isdir),
            D(ii).name = [prepath D(ii).name];
        end;
    end;

    % disp(sprintf('Scanning "%s" %g files found',[prepath
        postpath],length(D)));
elseif strcmp(wildpath,'**'),
    % a double wild directory means
    % recurs down into sub directories

    % first look for files in the current directory (remove
    % extra filesep)
    D = rdir([prepath postpath(2:end)]);

    % then look for sub directories
    Dt = dir('');
    tmp = dir([prepath '*']);
% process each directory
for ii = 1:length(tmp),
    if (tmp(ii).isdir && ~strcmpi(tmp(ii).name, '.') &&
        ~strcmpi(tmp(ii).name, '..'),
        Dt = [Dt; rdir([prepath tmp(ii).name filesep wildpath postpath])];
    end;
end;
D = [D; Dt];

else
    % Process directory wild card looking for sub directories that match
    tmp = dir([prepath wildpath]);
    D = dir('');
    % process each directory found
    for ii = 1:length(tmp),
        if (tmp(ii).isdir && ~strcmpi(tmp(ii).name, '.') &&
            ~strcmpi(tmp(ii).name, '..'),
            D = [D; rdir([prepath tmp(ii).name postpath])];
        end;
    end;
end;

% Apply filter
if (nargin>=2 && ~isempty(varargin{1})),
    date = [D.date];
    datenum = [D.datenum];
    bytes = [D.bytes];

    try
        eval(sprintf('D((%s)==0) = []', varargin{1}));
    catch
        warning('Error: Invalid TEST "%s", varargin{1}');
    end;
end;
% display listing if no output variables are specified
if nargout==0,
    pp = {'' 'k' 'M' 'G' 'T'};
    for ii=1:length(D),
        sz = D(ii).bytes;
        if sz<=0,
            disp(sprintf(' %31s %-64s','',D(ii).name));
        else
            ss = min(4,floor(log2(sz)/10));
            disp(sprintf('%4.0f %1sb %20s %-64s
',sz/1024^ss,pp{ss+1},D(ii).date,D(ii).name));
        end;
    end;
else
    % send list out
    varargout{1} = D;
end;
B.4 Convert gray scale to RGB

% Script for converting gray scaled images to RGB scaled images
% Uses rdir to find the 'deepest' files as well
% ----------------------------------------------- %
% Convert_GrayScale_toRGB
% by Margo van der Stam
% 20 - 5 - 2016
% ----------------------------------------------- %

img_list = rdir('C:\Users\Margo\Documents\MATLAB\MSFDE\MSFDE\**\*.tif');
size(img_list)
originalFolder = pwd; % Saves current folder
for i = 1 : size(img_list, 1)
    [pathstr,filename,ext] = fileparts(img_list(i).name);
    cd(pathstr); %Go to the directory of the image
    filename = strtrim(filename); %Strip filename from whitespaces etc.
    image = imread(strcat(filename, ext));

    % Convert images to RGB and write as png to certain location
    rgbImage = repmat(double(image) ./ 255, [1 1 3]);
    cd('C:\Users\Margo\Documents\MATLAB\MSFDE\RGBvalues\');
    imwrite(rgbImage, strcat(filename, '.png'));
end
Appendix C

Writing the arff file

C.1 Main

import java.io.File;
import java.io.IOException;
//import java.net.URI;
//import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.Collection;
import org.apache.commons.io.FileUtils;

/**
 * @author Margo van der Stam Class that writes an arff file
 * based on images and
 * preprocesses them as well.
 */

public class Main {
    private static File path = new File("C:/Users/Margo/Documents/Universiteit/Bachelor Thesis/Output/Matlab_Frontal");
    private static Collection<File> files;
    private static String data;
}
private static String arff_file;

public static void main(String[] args) throws IOException {
    arff_file = "C:/Users/Margo/Documents/Universiteit/Bachelor Thesis/FrontalImages.arff";
    files = getFiles();
    data = preprocess();
    WriteArff arff = new WriteArff(data, arff_file);
}

/**
* Based on the file path of the image, determines the class and appends the
* information of that image into one string.
*
* @return One string with information about the processed images
* @throws IOException throws an exception in case the file couldn't be opened
*/
private static String preprocess() throws IOException {
    Face face = null;
    StringBuilder sb = new StringBuilder();
    for (File file : files) {
        String path = file.getAbsolutePath().split("\\\")[8];
        switch (path) {
            case "KDEF":
                face = new FaceKDEF(file);
                break;
            case "MSFDE":
                face = new FaceMSFDE(file);
                break;
            case "RaFD":
                break;
        }
        sb.append(face.toString());
    }
    return sb.toString();
}
face = new FaceRaFD(file);
break;

default:
    face = null;
}

Emotion emotion = face.getEmotion();
if (emotion != null && emotion != Emotion.CONTEMPT) {
    sb.append(face.toString());
} else {
    System.out.println(file.getAbsolutePath());
    System.out.println(face.getEmotion());
}

return sb.toString();

/**
 * Finds all the files with the specified filetype
 * and prints the size of
 * the collection
 *
 * @return A Collection of the files with the
 * specified type
 */
private static Collection<File> getFiles() {
    String[] filetypes = {"png", "jpg"};
    Collection<File> files =
        FileUtils.listFiles(path, filetypes, true);
    System.out.println(files.size());
    return files;
}
C.2 KeyPoints

/**
 * Enum for all the keypoints in the face according to DRMF2013
 * @author Margo van der Stam
 */
public enum keyPoints {
    JawlineLeft(1), JawlineLeft1(2), JawlineLeft2(3),
    JawlineLeft3(4), JawlineLeft4(5), JawlineLeft5(6),
    JawlineLeft6(7), JawlineLeft7(8), JawlineMiddle(9),
    JawlineRight1(10),
    JawlineRight2(11), JawlineRight3(12), JawlineRight4(13),
    JawlineRight5(14),
    JawlineRight6(15),
    JawlineRight7(16),
    JawlineRight8(17),

    // Eyebrows from left to right
    EyebrowLeftL(18), EyebrowLeft1(19), EyebrowLeft2(20),
    EyebrowLeft3(21), EyebrowLeft4(22), EyebrowLeft5(23), EyebrowLeft6(24),
    EyebrowLeft7(25),
    EyebrowLeftR(26),
    EyebrowRight1(27),

    // Nose from between the eye to the tip
    NoseUp(28), Nose1(29), Nose2(30), NoseTip(31),

    // Below the nose from left to right
    NoseBLeft(32), NoseBLeft1(33), NoseBMiddle(34),
    NoseBRight1(35), NoseBRight(36),

    // Left eye

}
C.2. KEYPOINTS  APPENDIX C. WRITING THE ARFF FILE

EyeLeftIn(37), EyeLeftUp1(38), EyeLeftUp2(39),
→ EyeLeftOuter(40), EyeLeftDown1(41),
→ EyeLeftDown2(42),

// Right eye
EyeRightIn(43), EyeRightUp1(44), EyeRightUp2(45),
→ EyeRightOuter(46), EyeRightDown1(47),
→ EyeRightDown2(48),

// Mouth
MouthLeft(49), MouthLeftUp1(50), MoutLeftUp2(51),
→ MouthMiddeleUp(52), MouthRightUp2(53),
→ MouthRightUp1(54), MouthRight(55),
→ MouthRightDown1(56),
→ MouthRightDown2(57),
→ MouthMiddleDown(58),
→ MouthLeftDown2(59),
→ MouthLeftDown1(60),
→ TeethLeft(61),
→ TeethMiddle(62),
→ TeethRight(63),
→ Teeth1(64),
→ Teeth2(65),
→ Teeth3(66);

private int i;
private static keyPoints[] list = keyPoints.values();

private keyPoints(int i) {
    this.i = i;
}

/**
   * @return Function that creates a string for the arff file with the
   */
C.2. KEYPONITS  Appendix C. Writing the ARFF File

`@ATTRIBUTE attribute name x NUMERIC
@ATTRIBUTE attribute name y NUMERIC
`
C.3 Tuple

/**
 * Generic class for creating a Tuple
 * 
 * @author Margo van der Stam
 * 
 * @param <T>
 * @param <U>
 */
public class Tuple<T, U> {

    public T x;
    public U y;

    public Tuple() {
    }

    public Tuple(T x, U y) {
        this.x = x;
        this.y = y;
    }

    @Override
    public boolean equals(Object o) {
        if (o == null || o.getClass() !=
            this.getClass()) {
            return false;
        } else {
            @SuppressWarnings("unchecked")
            Tuple<T, U> t = (Tuple<T, U>) o;
            return t.x.equals(this.x) &&
            t.y.equals(this.y);
        }
    }

    @Override
    public String toString() {

    }

    @Override
    public String toString() {

    }
}

56
StringBuilder sb = new StringBuilder();
    sb.append(x.toString() + ',');
    sb.append(y.toString());
    return sb.toString();
}
C.4 Emotion

public enum Emotion {
    NEUTRAL, HAPPINESS, SADNESS, ANGER, SURPRISE, FEAR,
    DISGUST, CONTEMPT;
}

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.HashMap;
import java.util.Map;

/**
 * @author Margo van der Stam Class for handling the
 * preprocessing for the
 * generic part of the data sets
 */
public abstract class Face {

    private Map<Integer, Tuple<Double, Double>> keyPointsMap = new HashMap<Integer, Tuple<Double, Double>>();
    private Map<Integer, Tuple<Double, Double>> preprocessedKeyPoints = new HashMap<Integer, Tuple<Double, Double>>();
    protected File file;

    public Face(File file) throws IOException {
        createFace(file);
        normalize(calculateFactor());
        this.file = file;
    }

    /**
     * Function that reads all the coordinates of the
     * keypoints and stores them
     * in a map
     */
private void createFace(File file) throws IOException {
    Tuple<Double, Double> coordinates;
    FileInputStream stream = null;
    stream = new FileInputStream(file);
    BufferedReader reader = new BufferedReader(new InputStreamReader(stream));
    String line;
    line = reader.readLine(); // Skip first line
    // Read File Line By Line
    for (int i = 0; i < keyPoints.values().length; i++) {
        line = reader.readLine();
        String[] splitted = line.split(" ");
        Double x = Double.parseDouble(splitted[0]);
        Double y = Double.parseDouble(splitted[1]);
        coordinates = keyPointsMap.put(i, new Tuple<Double, Double>(x, y));
    }
    // Close the input stream
    stream.close();
}

/**
 * Function that determines the corresponding emotion according to
 * information in the filename
 *
 * @return the emotion
 */
abstract Emotion getEmotion();

/**
 * Calculates the factor: the average of the distance
 * between the tip of the nose and the highest point of the jaw near the ear
 * @return the factor
 */
private double calculateFactor() {
    int nose = keypoints.NoseTip.ordinal();
    int jawLeft = keypoints.JawlineLeft.ordinal();
    int jawRight = keypoints.JawlineRight.ordinal();
    double distanceLeft =
        calculateDistance(keypointsMap.get(nose),
                         keypointsMap.get(jawLeft));
    double distanceRight =
        calculateDistance(keypointsMap.get(nose),
                         keypointsMap.get(jawRight));
    double distance = (distanceLeft +
                        distanceRight) / 2;
    return distance;
}

/**
 * Function that calculates the distance between two coordinates
 * @param x first coordinate
 * @param y second coordinate
 * @return distance between two coordinates
 */
private double calculateDistance(Tuple<Double, Double> x, Tuple<Double, Double> y) {

return Math.sqrt(Math.abs(x.x - y.x) + 
   Math.abs(x.y - y.y));
}

/**
 * Function for normalizing the coordinates by
deviding all coordinates by
 * the calculated factor
 *
 * @param factor
 * calculated by @see calculateFactor
 */
private void normalize(double factor) {
    for (int i = 0; i < keyPoints.values().length; i++) {
        Tuple coordinates =
            keyPointsMap.get(i);
        double x = (double) coordinates.x /
            factor;
        double y = (double) coordinates.y /
            factor;
        preprocessedKeyPoints.put(i, new
            Tuple<Double, Double>(x, y));
    }
}

public Map<Integer, Tuple<Double, Double>>
    getKeyPoints() {
    return preprocessedKeyPoints;
}

/**
 * Creates a string of all preprocessed data and
 * append the right emotion to
 * it
 */
public String toString() {
    StringBuilder data = new StringBuilder();

    // Code continues...
for (Tuple coordinates : preprocessedKeyPoints.values()) {
    data.append(coordinates.toString() + ',');
}
data.append(this.getEmotion() + "\n");
return data.toString();
C.5.1 FaceKDEF

```java
import java.io.File;
import java.io.IOException;

/**
 * Class for data of the KDEF data set
 * @author Margo van der Stam
 */
public class FaceKDEF extends Face {

    public FaceKDEF(File file) throws IOException {
        super(file);
    }

    @Override
    Emotion getEmotion() {
        String name = file.getName();
        String emotion = name.substring(4, 6);
        switch (emotion) {
            case "AF":
                return Emotion.FEAR;
            case "AN":
                return Emotion.ANGER;
            case "DI":
                return Emotion.DISGUST;
            case "HA":
                return Emotion.HAPPINESS;
            case "NE":
                return Emotion.NEUTRAL;
            case "SA":
                return Emotion.SADNESS;
            case "SU":
                return Emotion.SURPRISE;
        }
        return null;
    }
}
```
C.5. FACE

APPENDIX C. WRITING THE ARFF FILE

```
C.5.2 FaceMSFDE

```java
import java.io.File;
import java.io.IOException;

/**
 * Class for data of the MSFDE data set
 *
 * @author Margo van der Stam
 */

public class FaceMSFDE extends Face {

    public FaceMSFDE(File file) throws IOException {
        super(file);
    }

    @Override
    Emotion getEmotion() {
        String name = file.getName();
        String emotion = name.substring(2, 3);
        switch (emotion) {
            case "1":
                return Emotion.ANGER;
            case "2":
                return Emotion.HAPPINESS;
            case "3":
                return Emotion.SADNESS;
            case "4":
                return Emotion.FEAR;
            case "5":
                return Emotion.DISGUST;
            case "6":
                return Emotion.CONTEMPT;
            default:
                return Emotion.NEUTRAL;
        }
    }
}
```
C.5.3 FaceRaFD

```java
import java.io.File;
import java.io.IOException;

/**
 * Class for data of the Radboud Faces data set
 *
 * @author Margo van der Stam
 *
 */
public class FaceRaFD extends Face {

    public FaceRaFD(File file) throws IOException {
        super(file);
    }

    @Override
    Emotion getEmotion() {
        String name = file.getName();
        String emotion = name.split("_")[4];
        switch (emotion) {
            case "surprised":
                return Emotion.SURPRISE;
            case "sad":
                return Emotion.SADNESS;
            case "neutral":
                return Emotion.NEUTRAL;
            case "happy":
                return Emotion.HAPPINESS;
            case "fearful":
                return Emotion.FEAR;
            case "disgusted":
                return Emotion.DISGUST;
            case "contemptuous":
                return Emotion.CONTEMPT;
            case "angry":
                return Emotion.ANGER;
        }
    }
}
```
C.5. FACE APPENDIX C. WRITING THE ARFF FILE

} 

return null;

} 

}
C.6 WriteArff

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;

/**
 * Class for creating an Arff file depending on the data
 * @author Margo van der Stam
 */

public class WriteArff {
    FileWriter writer;
    File file;
    String data;

    /**
     * Creates a new empty arff file
     * @param preprocessed data from one of the images
     * @param arff_file Name of the arff file
     */

    public WriteArff(String data, String arff_file) {
        this.data = data;
        boolean bool = false;
        try {
            file = new File(arff_file);
            bool = file.createNewFile();
            System.out.println("File created: ");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public WriteArff(String data, String arff_file) {
        this.data = data;
        boolean bool = false;
        try {
            file = new File(arff_file);
            bool = file.createNewFile();
            System.out.println("File created: " + bool);

            } catch (IOException e) {
                e.printStackTrace();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public WriteArff(String data, String arff_file) {
        this.data = data;
        boolean bool = false;
        try {
            file = new File(arff_file);
            bool = file.createNewFile();
            System.out.println("File created: " + bool);

            } catch (IOException e) {
                e.printStackTrace();
            }
    }

70
C.6. WRITEARFF  APPENDIX C. WRITING THE ARFF FILE

```java
write();
}

/**
 * Writes the heading of the arff file and continues
 * with the data from the
 * images
 */
public void write() {
    try (Writer writer = (new FileWriter(file))) {
        writer.write("% 1. Title: emotion
             keypoints
             database\n %n 2.
             Sources:\n	(a) Creater: M. van der
	(b)Date:
	 June, 2016\n %n");
        writer.write("@RELATION emotion\n\n`");
        writer.write(keyPoints.toArff());
        writer.write("@ATTRIBUTE class
             {NEUTRAL, HAPPINESS, SADNESS,
             ANGER, SURPRISE, FEAR,
             DISGUST}\n\n`");
        writer.write("@DATA\n\n`");
        writer.write(data);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```
FaceReader performance calculator

C.7  Main

package faceReader;
import java.io.File;
/**
 * @author Margo van der Stam
 *
 */
public class Main {
    /**
     * The path to the folder where the images/databases are
     */
    private static File path = new File("C:/Users/Margo/Documents/Universiteit/Bachelor Thesis/Output/FaceReader_Frontal/detailed");
    private static File pathState = new File("C:/Users/Margo/Documents/Universiteit/Bachelor Thesis/Output/FaceReader/state");

    public static void main(String[] args) {
        // faceReader();
        faceReaderState();
    }
}
public static void faceReader() {
    PerformanceCalculator calculator = new
            PerformanceCalculator();
    new FaceReader(path, calculator);
    System.out.println(calculator.getResultsKDEF());
    System.out.println(calculator.getResultsMSFDE());
    System.out.println(calculator.getResultsRaFD());
    System.out.println(calculator.getResultsTotal());
}

public static void faceReaderState() {
    PerformanceCalculatorState calculator = new
            PerformanceCalculatorState();
    new FaceReaderState(pathState, calculator);
    System.out.println(calculator.getResultsKDEF());
    System.out.println(calculator.getResultsMSFDE());
    System.out.println(calculator.getResultsRaFD());
    System.out.println(calculator.getResultsTotal());
}
C.8 Emotion

package faceReader;

/**
 * Class for all the emotions displayed in the images
 * @author Margo van der Stam june 2016
 */
enum Emotion {
    HAPPY, NEUTRAL, SAD, ANGRY, SURPRISED, SCARED,
    DISGUSTED;

    /**
     * @param index integer
     * @return Converted integer to emotion
     */
    public static Emotion getEmotion(int index) {
        for (Emotion e : Emotion.values()) {
            if ((e.ordinal() + 1) == index)
                return e;
        }
        return null;
    }
}
C.9 FaceReader

package faceReader;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Collection;
import org.apache.commons.io.FileUtils;

/**
 * @author Margo van der Stam june 2016
 */

class FaceReader {
    private static File path;
    static Collection<File> files;
    static PerformanceCalculator calculator;

    public FaceReader(File path, PerformanceCalculator calculator) {
        FaceReader.calculator = calculator;
        FaceReader.path = path;
        files = getFiles();
        loopFiles();
    }

    private static Collection<File> getFiles() {
        String[] filetypes = { "txt" };
        Collection<File> files = FileUtils.listFiles(path, filetypes, true);
        return files;
    }
}
private static void loopFiles() {
    double performanceRate = 0;
    double counter = 0;
    for (File file : files) {
        double result[] = readFile(file);
        performanceRate += result[0];
        counter += result[1];
    }
    // System.out.println("Performancerate: "+
    // performanceRate/counter +
    // "Counter: "+counter);
}

private static double[] readFile(File file) {
    FileInputStream stream = null;
    String line;
    double performanceRate = 0;
    double counter = 0;
    try {
        stream = new FileInputStream(file);
        BufferedReader reader = new
        // BufferedReader(new
        // InputStreamReader(stream));
for (int i = 0; i <= 6; i++) {
    line = reader.readLine();
}

line = reader.readLine();
while (line != null) {
    readLine(line);
    line = reader.readLine();
}
reader.close();
} catch (FileNotFoundException e) {
    System.out.println("File could not be found: " + file);
} catch (IOException e) {
    System.out.println("File could not be read: " + file);
}
performanceRate = performanceRate / counter;
return new double[]{performanceRate, counter};
}

/**
 * @param line
 * One line of the facereader file,
 * normalizes the score and saves it to the correct variable
 */
private static void readLine(String line) {
    boolean fit_failed = false;
    int scoreNr = 0;
    int db = 0;
    double score = 0;
    String[] splitted = line.split("\t");
    System.out.println("Filename" + splitted[0]);
    String[] splittedname = splitted[0].split("\\\\");
```java
switch (splittedname[7]) {
    case "KDEF":
        scoreNr =
        ->  getScoreKDEF(splittedname[8]);
        db = 3;
        break;
    case "RaFD":
        scoreNr =
        ->  getScoreRaFD(splittedname[8]);
        db = 1;
        break;
    case "MSFDE":
        scoreNr =
        ->  getScoreMSFDE(splittedname[10]);
        db = 2;
}
if (scoreNr != -1) {
    if
    ->  (!splitted[scoreNr].equals("FIT_FAILED")
    ->      &&
    ->  !splitted[scoreNr].equals("FIND_FAILED"))
    ->  {
        double sum =
        ->  Double.parseDouble(splitted[1])
        ->      +
        ->  Double.parseDouble(splitted[2])
        ->      +
        ->  Double.parseDouble(splitted[3])
        ->      +
        ->  Double.parseDouble(splitted[4])
        ->      +
        ->  Double.parseDouble(splitted[5])
        ->      +
        ->  Double.parseDouble(splitted[6])
        ->      +
        ->  Double.parseDouble(splitted[7])
```
C.9. FACEREADER APPENDIX C. WRITING THE ARFF FILE

```java
//
// System.out.println((Double.parseDouble(splitted[scoreNr])
// / sum));
 score = Double.parseDouble(splitted[scoreNr]) / sum;
// score =
// -Math.log(Double.parseDouble(splitted[scoreNr]) / sum) / Math.log(2);
}
else {
 System.out.println("Fit Failed:", splitted[0]);
 score = 1.0 / 7;
 fit_failed = true;
// score =
// -Math.log(1/7)/Math.log(2);
}
 calculator.addPerformance(Emotion.getEmotion(scoreNr), score, db, fit_failed);
}
else {
 // System.out.println("This emotion could not be mapped");
 score = 1;
}

/**
 * @param name, the name of the file
 * @return The score of facereader for the emotion it should be according to the database Radboud Faces
 */
 private static int getScoreRaFD(String name) {
 String[] emotion = name.split("_");
```
switch (emotion[4]) {
    case "neutral":
        return 1;
    case "happy":
        return 2;
    case "sad":
        return 3;
    case "angry":
        return 4;
    case "surprised":
        return 5;
    case "fearful":
        return 6;
    case "disgusted":
        return 7;
    case "contemptuous":
        return -1;
    default:
        System.out.println(name);
        return 0;
}

/**
 * @param name, the name of the file
 * @return The score of facereader for the emotion it should be according to
 * the database MSFDE
 */
private static int getScoreMSFDE(String name) {
    char emotion = name.charAt(2);
    switch (emotion) {
        case ' ':
            return 1;
        case 'M':
            return 1;
        case 'n':
            return 1;
        case 'n':
            return 1;
    }
}
```java
private static int getScoreKDEF(String name) {
    String emotion = name.substring(4, 6);
    switch (emotion) {
    case "NE":
        return 1;
    case "HA":
        return 2;
    case "SA":
        return 3;
    case "AN":
        return 4;
    case "SU":
        return 6;
    case "5":
        return 7;
    case "6":
        return -1;
    default:
        System.out.println(name);
        return -1;
    }
    return -1;
}
```

/**
 * @param name, the name of the file
 * @return The score of facereader for the emotion it should be according to the database KDEF
 */
return 5;
case "AF":
    return 6;
case "DI":
    return 7;
default:
    return -1;
}
C.10 PerformanceCalculator

package faceReader;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * @author Margo van der Stam june 2016
 */

/**
 * @author Margo
 *
 */

public class PerformanceCalculator {
    private static Map<Emotion, List<Double>> resultsRaFD = new HashMap();
    private static Map<Emotion, List<Double>> resultsMSFDE = new HashMap();
    private static Map<Emotion, List<Double>> resultsKDEF = new HashMap();
    private static int failed_RaFD;
    private static int failed_MSFDE;
    private static int failed_KDEF;

    public PerformanceCalculator() {
        initialize();
    }

    private void initialize() {
        for (Emotion e: Emotion.values()){
            resultsRaFD.put(e, new ArrayList<Double>();
            resultsMSFDE.put(e, new ArrayList<Double>();
            resultsKDEF.put(e, new ArrayList<Double>();
        }
    }
}
C.10. PERFORMANCECALCULATOR APPENDIX C. WRITING THE ARFF FILE

```java
resultsKDEF.put(e, new ArrayList<Double>();
}
failed_RaFD = 0;
failed_MSFDE = 0;
failed_KDEF = 0;
}

/**
 * Function for adding a certain score to the right "storing space"
 * @param emotion Emotion that belongs to the score
 * @param score Score from FaceReader for a particular Emotion
 * @param database An integer coding for the data set
 * @param fit_failed boolean stating whether the image was marked as a fit failed
 */
public void addPerformance(Emotion emotion, double score, int database, boolean fit_failed) {
    if(score == 0.0)
        score = 1 * 10^(-16);
    score = -Math.log(score) / Math.log(2);
    switch (database) {
    case 1:
        resultsRaFD.get(emotion).add(score);
        if(fit_failed)
            failed_RaFD++;
        break;
    case 2:
        if(fit_failed)
            failed_MSFDE++;
        resultsMSFDE.get(emotion).add(score);
        break;
    case 3:
        if(fit_failed)
            failed_KDEF++;
        resultsKDEF.get(emotion).add(score);
```
public String getResultsRaFD() {
    return "Results Radboud Faces: " +
    calculateAverageDataSet(resultsRaFD) + "\n"
    + toString(calculateResults(resultsRaFD)) +
    "Number of fit failed images: " +
    failed_RaFD + "\n";
}

public String getResultsKDEF() {
    return "Results KDEF: " +
    calculateAverageDataSet(resultsKDEF) + "\n"
    + toString(calculateResults(resultsKDEF)) +
    "Number of fit failed images: " +
    failed_KDEF + "\n";
}

public String getResultsMSFDE() {
    return "Results MSFDE: " +
    calculateAverageDataSet(resultsMSFDE) +
    "\n" +
    toString(calculateResults(resultsMSFDE)) +
    "Number of fit failed images: " +
    failed_MSFDE + "\n";
}

public String getResultsTotal() {
    int total = failed_KDEF + failed_MSFDE +
    failed_RaFD;
}
return "Results Total: " +
   calculateResultsOverall() + "\n" +
   toString(calculateResultsTotal()) + "Total
   Number of fit failed images: " + total +
   "\n";
}

/**
 * @param results a list of scores for one emotion
 * @return the average of the list
 */
private double calculateAverage(List<Double> results) {
    double sum = 0;
    for (Double result : results) {
        if(Double.isNaN(result)){
            System.out.println(result);
            result = 0.0;
        }
        sum += result;
    }
    return results.isEmpty() ? 0 : sum / results.size();
}

/**
 * @param map with per emotion the average score for
 * that data set
 * @return a string with the emotion followed by the
 * score, for all emotions
 */
private String toString(Map<Emotion, Double> map) {
    StringBuilder results = new StringBuilder();
    for(Emotion e : Emotion.values()){
        if(map.get(e) == -100.0)
            results.append(e + " : -100.0\n");
        else
            results.append(e + " : " + map.get(e) + "\n");
    }
    return results.toString();
}
C.10. PERFORMANCECALCULATOR APPENDIX C. WRITING THE ARFF FILE

```java
results.append(e.toString() + 
   "": This emotion wasn't in 
   the data set
"");
else
results.append(e.toString() + 
   "": " + 
   (map.get(e)).toString() + 
   "\n");
}
return results.toString();
```

```java
/**
 * @param dataSet, one list containing information on 
 * the images of one data set
 * @return a Map with for every emotion the average 
 * score (if none the score is -100)
 */
private Map<Emotion, Double> 
   calculateResults(Map<Emotion, List<Double>> 
   dataSet){
   HashMap<Emotion, Double> results = new 
   HashMap<Emotion, Double>();
   for(Emotion e : Emotion.values()){
      if(dataSet.get(e).isEmpty())
         results.put(e, -100.0);
      else results.put(e, 
         calculateAverage(dataSet.get(e)));
   }
   return results;
}
```

```java
/**
 * @param results, results of one data set
 * @return the total average score of that data set
 */
private double calculateAverageDataSet(Map<Emotion, 
   List<Double>> results){
```
List<Double> combined = new ArrayList<Double>();
for(Emotion e: Emotion.values()){
    combined.addAll(results.get(e));
}
return calculateAverage(combined);

/**
 * @return calculates the total average score of all
 * data sets per emotion
 */
private Map<Emotion, Double> calculateResultsTotal() {
    HashMap<Emotion, Double> results = new HashMap<Emotion, Double>();
    List<Double> combined = new ArrayList<Double>();
    for(Emotion e: Emotion.values()){
        combined.addAll(resultsRaFD.get(e));
        combined.addAll(resultsMSFDE.get(e));
        combined.addAll(resultsKDEF.get(e));
        results.put(e, calculateAverage(combined));
        combined.clear();
    }
    return results;
}

/**
 * @return Calculates the total average of all data
 * sets over all emotion
 */
private double calculateResultsOverall() {
    List<Double> combined = new ArrayList<Double>();
    for(Emotion e: Emotion.values()){
        combined.addAll(resultsRaFD.get(e));
        combined.addAll(resultsMSFDE.get(e));
    }
    return calculateAverage(combined);
}
combined.addAll(resultsKDEF.get(e));
}
return calculateAverage(combined);
}
C.11. FACEREADERSTATE APPENDIX C. WRITING THE ARFF FILE

C.11 FaceReaderState

package faceReader;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Collection;
import org.apache.commons.io.FileUtils;

/**
 * @author Margo van der Stam june 2016
 */

public class FaceReaderState {

    private static File path;
    static Collection<File> files;
    static PerformanceCalculatorState calculator;

    public FaceReaderState(File path,
                            PerformanceCalculatorState calculator) {
        FaceReaderState.calculator = calculator;
        FaceReaderState.path = path;
        files = getFiles();
        loopFiles();
    }

    private static Collection<File> getFiles() {
        String[] filetypes = {"txt"};
        Collection<File> files =
            FileUtils.listFiles(path, filetypes, true);
        return files;
    }

}
/**
 * Runs over all the files, and keeps track of the
 * performance rate and
 * amount of images per file
 */
 private static void loopFiles() {
    double performanceRate = 0;
    double counter = 0;
    for (File file : files) {
        double result[] = readFile(file);
        performanceRate += result[0];
        counter += result[1];
    }
    // System.out.println("Performance rate: " +
    // performanceRate/counter +
    // "Counter: " + counter);
}

/**
 * @param file
 * an output file of Facereader
 * @return returns an array with the performance rate
 * and the amount of
 * images rated for a file Using a
 * BufferedReader this function
 * passes a line to a function that can read
 * that line
 */
 private static double[] readFile(File file) {
    FileInputStream stream = null;
    String line;
    double performanceRate = 0;
    double counter = 0;
    try {
        stream = new FileInputStream(file);
        BufferedReader reader = new
        BufferedReader(new
        InputStreamReader(stream));
    } catch (IOException e) {
        e.printStackTrace();
    }
    return new double[]{performanceRate, counter};
}
for (int i = 0; i <= 6; i++) { // Skip first lines
    line = reader.readLine();
}

line = reader.readLine();
while (line != null) {
    readLine(line);
    line = reader.readLine();
}
reader.close();

} catch (FileNotFoundException e) {
    System.out.println("File could not be found: " + file);
}
} catch (IOException e) {
    System.out.println("File could not be read: " + file);
}

performanceRate = performanceRate / counter;
return new double[] { performanceRate, counter };

/**
 * @param line One line of the facereader file
 */
private static void readLine(String line) {
    int scoreNr = 0;
    int db = 0;
    double score = 0;
    String[] splitted = line.split("\t");
    String[] splittedname = splitted[0].split("\\\\");
    String emotion = splitted[1];
    switch (splittedname[7]) {
    case "KDEF":
C.11. FACEREADERSTATE APPENDIX C. WRITING THE ARFF FILE

```java
scoreNr =
    getScoreKDEF(splittedname[8]);
db = 3;
break;

case "RaFD":
    scoreNr =
    getScoreRaFD(splittedname[8]);
db = 1;
break;

case "MSFDE":
    scoreNr =
    getScoreMSFDE(splittedname[10]);
db = 2;
}
if (scoreNr != -1) {
    calculator.addPerformance(Emotion.getEmotion(scoreNr),
    convertEmotion(emotion), scoreNr,
    db);
} else {
    // System.out.println("This emotion could not be mapped");
    score = 1;
}
}

private static int convertEmotion(String emotion) {
    switch (emotion) {
    case "Neutral":
        return 1;
    case "Happy":
        return 2;
    case "Sad":
        return 3;
    case "Angry":
        return 4;
    case "Surprised":
        return 5;
    }
}
case "Scared":
    return 6;
case "Disgusted":
    return 7;
case "Unknown":
    return -1;
}
return 0;
}

/**
 * @param name, the name of the file
 * @return The score of facereader for the emotion it should be according to the database Radboud Faces
 */
private static int getScoreRaFD(String name) {
    String[] emotion = name.split("_");
    switch (emotion[4]) {
    case "neutral":
        return 1;
    case "happy":
        return 2;
    case "sad":
        return 3;
    case "angry":
        return 4;
    case "surprised":
        return 5;
    case "fearful":
        return 6;
    case "disgusted":
        return 7;
    case "contemptuous":
        return -1;
    default:
/**
 * @param name, the name of the file
 * @return The score of facereader for the emotion it should be according to the database MSFDE
 */
private static int getScoreMSFDE(String name) {
    char emotion = name.charAt(2);
    switch (emotion) {
        case ' ': return 1;
        case 'M': return 1;
        case 'n': return 1;
        case '2': return 2;
        case '3': return 3;
        case '1': return 4;
        case '4': return 6;
        case '5': return 7;
        case '6': return -1;
        default: System.out.println(name);
                return -1;
    }
}
/**
 * @param name, the name of the file
 * @return The score of facereader for the emotion it should be according to
 * the database KDEF
 */

private static int getScoreKDEF(String name) {
    String emotion = name.substring(4, 6);
    switch (emotion) {
    case "NE":
        return 1;
    case "HA":
        return 2;
    case "SA":
        return 3;
    case "AN":
        return 4;
    case "SU":
        return 5;
    case "AF":
        return 6;
    case "DI":
        return 7;
    default:
        return -1;
    }
}
}
package faceReader;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * @author Margo van der Stam june 2016
 */
public class PerformanceCalculatorState {

private static Map<Emotion, List<Double>> resultsRaFD =
    new HashMap<Emotion, List<Double>>();
private static Map<Emotion, List<Double>> resultsMSFDE =
    new HashMap<Emotion, List<Double>>();
private static Map<Emotion, List<Double>> resultsKDEF =
    new HashMap<Emotion, List<Double>>();
private static int failed_RaFD;
private static int failed_MSFDE;
private static int failed_KDEF;

public PerformanceCalculatorState() {
    initialize();
}

private void initialize() {
    for(Emotion e: Emotion.values()){
        resultsRaFD.put(e, new ArrayList<Double>());
        resultsMSFDE.put(e, new ArrayList<Double>());
        resultsKDEF.put(e, new ArrayList<Double>());
    }
    failed_RaFD = 0;
    failed_MSFDE = 0;
    failed_KDEF = 0;
}
C.12. PERFORMANCECALCULATORSTATE APPENDIX C. WRITING THE ARFF FILE

```java
/**
   * Function for adding a certain score to the right
   * "storing space"
   * @param emotion Emotion that belongs to the score
   * @param scoreFr Score from FaceReader for a
   * particular Emotion
   * @param scoreDb Score according to the annotated
data set
   * @param database An integer coding for the data set
   */
public void addPerformance(Emotion emotion, int
scoreFr, int scoreDb, int database) {
    double add = 0.0;
    boolean fit_failed = scoreFr == -1;
    if(scoreFr == scoreDb){
        add = 1.0;
    }
    switch (database) {
    case 1:
        resultsRaFD.get(emotion).add(add);
        if(fit_failed)
            failed_RaFD++;
        break;
    case 2:
        if(fit_failed)
            failed_MSFDE++;
        resultsMSFDE.get(emotion).add(add);
        break;
    case 3:
        if(fit_failed)
            failed_KDEF++;
        resultsKDEF.get(emotion).add(add);
        break;
    }
}
```

98
public String getResultsRaFD() {
    return "Results Radboud Faces: " + 
            calculateAverageDataSet(resultsRaFD) + "\n" 
            + toString(calculateResults(resultsRaFD)) + 
            "Number of fit failed images: " + 
            failed_RaFD + "\n";
}

public String getResultsKDEF() {
    return "Results KDEF: " + 
            calculateAverageDataSet(resultsKDEF) + "\n" 
            + toString(calculateResults(resultsKDEF)) + 
            "Number of fit failed images: " + 
            failed_KDEF + "\n";
}

public String getResultsMSFDE() {
    return "Results MSFDE: " + 
            calculateAverageDataSet(resultsMSFDE) + 
            "\n" + 
            toString(calculateResults(resultsMSFDE)) + 
            "Number of fit failed images: " + 
            failed_MSFDE + "\n";
}

public String getResultsTotal() {
    int total = failed_KDEF + failed_MSFDE + 
            failed_RaFD;
    return "Results Total: " + 
            calculateResultsOverall() + "\n" + 
            toString(calculateResultsTotal())+ "Total 
            Number of fit failed images: " + total + 
            "\n";
}
/**
 * @param results a list of scores for one emotion
 * @return the average of the list
 */

private double calculateAverage(List<Double> results) {
    double sum = 0;
    for (Double result : results) {
        if (Double.isNaN(result)) {
            System.out.println(result);
            result = 0.0;
        }
        sum += result;
    }
    return results.isEmpty() ? 0 : sum / results.size();
}

/**
 * @param map with per emotion the average score for that data set
 * @return a string with the emotion followed by the percentage, for all emotions
 */

private String toString(Map<Emotion, Double> map) {
    StringBuilder results = new StringBuilder();
    for (Emotion e : Emotion.values()) {
        if (map.get(e) == -100.0)
            results.append(e.toString() + ": This emotion wasn't in the data set\n");
        else
            results.append(e.toString() + "\:" + (map.get(e)).toString() + ":");
    }
    return results.toString();
}
C.12. PERFORMANCECALCULATORSTATE APPENDIX C. WRITING THE ARFF FILE

    }

    /**
     * @param dataSet, one list containing information on
     * the images of one data set
     * @return a Map with for every emotion the average
     * score (if none the score is -100)
     */
    private Map<Emotion, Double> calculateResults(Map<Emotion, List<Double>> dataSet){
        HashMap<Emotion, Double> results = new HashMap<Emotion, Double>();
        for(Emotion e : Emotion.values()){
            if(dataSet.get(e).isEmpty())
                results.put(e, -100.0);
            else results.put(e, calculateAverage(dataSet.get(e))*100);
        }
        return results;
    }

    /**
     * @param results, results of one data set
     * @return the total percentage score of that data
     */
    private double calculateAverageDataSet(Map<Emotion, List<Double>> results){
        List<Double> combined = new ArrayList<Double>();
        for(Emotion e : Emotion.values()){
            combined.addAll(results.get(e));
        }
        return calculateAverage(combined)*100;
    }

    /**
     * @param results, results of one data set
     * @return the total percentage score of that data
     */
    private double calculateAverageDataSet(Map<Emotion, List<Double>> results){
        List<Double> combined = new ArrayList<Double>();
        for(Emotion e : Emotion.values()){
            combined.addAll(results.get(e));
        }
        return calculateAverage(combined)*100;
    }

    /**
     */

101
* @return calculates the total average score of all data sets per emotion
 */
private Map<Emotion, Double> calculateResultsTotal() {
    HashMap<Emotion, Double> results = new
            HashMap<Emotion, Double>();
    List<Double> combined = new
            ArrayList<Double>();
    for(Emotion e: Emotion.values()){
        combined.addAll(resultsRaFD.get(e));
        combined.addAll(resultsMSFDE.get(e));
        combined.addAll(resultsKDEF.get(e));
        results.put(e,
                calculateAverage(combined)*100);
        combined.clear();
    }
    return results;
}

/**
 * @return Calculates the total percentage of all data sets over all emotion
 */
private double calculateResultsOverall() {
    List<Double> combined = new
            ArrayList<Double>();
    for(Emotion e: Emotion.values()){
        combined.addAll(resultsRaFD.get(e));
        combined.addAll(resultsMSFDE.get(e));
        combined.addAll(resultsKDEF.get(e));
    }
    return calculateAverage(combined)*100;
}