Automatic Human Emotion Detection

Retshidisitswe Lehata

Thesis presented in fulfilment
of the requirements for the degree of
Bachelor of Science Honours
at the University of the Western Cape

Supervisor: Mehrdad Ghaziasgar
Co-supervisor: Reg Dodds

This version November 22, 2017

Declaration

I, Retshidisitswe Lehata, declare that this thesis "*Automatic Human Emotion Detection*" is my own work, that it has not been submitted before for any degree or assessment at any other university, and that all the sources I have used or quoted have been indicated and acknowledged by means of complete references.

Signature: .... Date: ....

Retshidisitswe Lehata.

Abstract

Customer service is a large revenue stream for some companies, and ensuring that they provide the best quality service is likely to be their main priority. Not all customers readily express their emotions, verbally, regarding the quality of the service they are provided. Mehrabian 1980, states that 55% of communication is facial expression. The motivation for this project is to apply an Automatic Human Emotion Detection (AHED) system in cases where an employee interacts with a customer. The AHED system focuses on emotion recognition using facial expressions. The proposed method for this project, first captures an image from the camera, the image is processed using the Viola Jones Algorithm to detect the face and the Histogram of Oriented Gradients (HOG) to extract the features from the face. Lastly the AHED system is trained using Support Vector Machines (SVM), to classify each emotion. There are six universal human expressions described by Ekman and Friesen, namely Surprise, Fear, Disgust, Anger, Happiness and Sadness. Grayscale frontal images will be used as input for the AHED system.

Acknowledgment

This thesis is a compilation of the efforts of many people that helped and me through the years. I would first like to thank my supervisor Mehrdad Ghaziasgar and co-supervisor Reg Dodds for encouraging me during my study. Without our weekly meetings, this work would not have been possible. At this time I would like to extend a very special thanks to Waleed Deaney for being my mentor, without his help I would certainly not be where I am today.

Contents

List of Tables

List of Figures

Glossary

AHED Automatic Human Emotion Detection.

SVM Support Vector Machines, a supervised machine learning model.

HOG Histogram of Oriented Gradients, a feature extraction method used for object detection.

OpenCV Open Source Computer Vision Library.

CK+ dataset The extended Cohn-Kanade dataset of facial expressions.

Scikit-Learn/sklearn Machine learning library in Python.

Chapter 1

Introduction

Communication plays a large role, in daily human interaction. It can take the form of verbal or non-verbal communication, Mehrabian found that over 93% of verbal communication is conveyed through ones tone of voice, 38%, and non-verbal ques, 55% [6]. Understanding non-verbal communication is a valuable skill, in that it is a universal form of communication. Non-verbal communication is a combination of body language, physical gestures and facial expressions. Ekman & Friesen found six facial expressions that are universally identifiable in recognizing Fear, Anger, Disgust, Surprise, Happiness and Sadness [7]. A facial expression is made up of the changes in facial muscles mainly the mouth, eyes and eyebrows. These changes help to reflect ones current state of mind.

The rest of this chapter is organised as follows: Section 1.1 describes the problem statement; Section 1.2 provides the overview of the solution proposed in this project and Section 1.3 outlines the method of implementing the proposed solution.

1.1 Problem Statement

Companies use feedback from their customers as a metric to measure their customer satisfaction rate. Customer feedback can be initiated by the customer as a compliment or criticism, based on the service they were given. Alternatively, a company can offer their customers optional online or physical surveys to be completed. The problem is that customers are more likely to refrain from commenting on a service, unless provoked to do so. Customer service is a large revenue stream for companies whose core business is based around the customer. Ensuring that the customer stays happy is key, for their success.

1.2 Proposed Solution

An Automatic Human Emotion Detection system can be applied in any environment that benefits from understanding facial expressions and human emotion. The proposed solution combines customer satisfaction with an automated system. This is done by using face detection to find the customers face in an image and facial feature extraction to identify the dominant customer emotion features in that image. To get the best results possible the training and classification of the system will be done using a machine learning technique. Companies can later incorporate the results of the system in improving their customer service, ensuring that their customers stay satisfied.

1.3 Proposed Method

A grayscale frontal image of the customer is used as input for the system. The Viola Jones Algorithm is then used to detect the location of the face in the image and the Histogram of Oriented Gradients is used to extract the features. Lastly the AHED system is trained using Support Vector Machines, to classify each emotion.

Chapter 2

Related Work

There are a wide variety of methods that are used in the field of emotion recognition using facial expressions. However, a large number of those methods are implemented using a similar process. Shown in Figure 2.1, initially the image is captured and processed. Thereafter the face is detected and features are extracted. Then a machine learning technique is trained to do the classification of the emotions. The related work looks at all four stages in the implementation process and different methods used by other researchers. The rest of this chapter is organised as follows: Section 2.1 describes the process of obtaining the image from the camera and image processing; Section 2.2 explains how face detection functions; Section 2.3 explains the feature extraction process; Section 2.4 the training of the machine learning technique and Section 2.5 outlines the results achieved by other researchers.

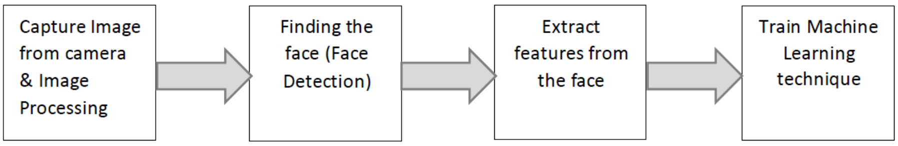

Capture Image from camera & Image Processing
Finding the face (Face Detection)
Extract features from the face
Train Machine Learning technique

Figure 2.1: The four stages for facial expression recognition

2.1 Capture Image from the Camera and Image Processing

An image is taken from the camera and image processing tools are used to help normalize and standardize the input image.

- Goyal and Mittal, achieved the desired resolution and colour for their images by adjusting the brightness and contrast of the image [8].

- Reddy and Srinivas, scaled and cropped their images to 150 × 120, and ensured that the location of the eyes stayed the same in each image. The image was processed further, using an average combination of all the input image histograms. This process is called histogram equalization, see Figure 2.2, and helps in decreasing variation in an image. Histogram equalization is a technique for stretching out the intensity range of an image to enhance the contrast of the image [9].

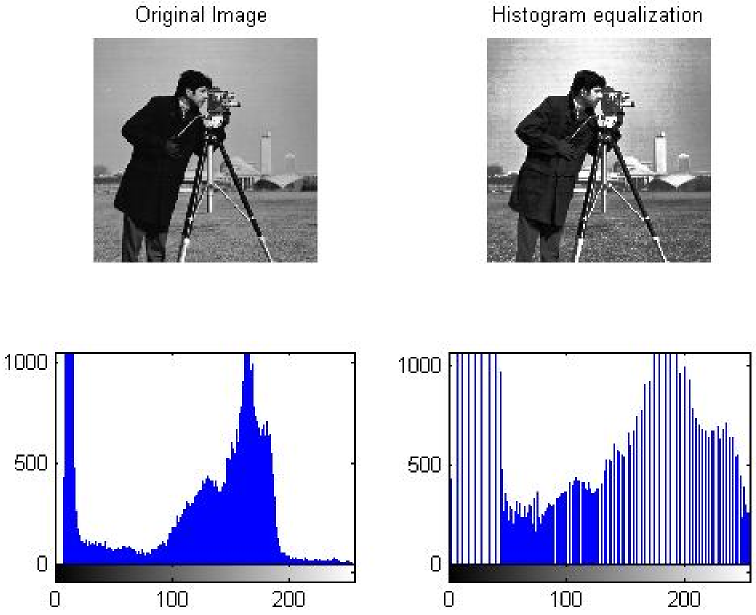


Figure 2.2: Histogram Equalization

- Boubenna and Lee, scaled their images to 100 × 100 pixels [10].

2.2 Face Detection - Finding the Face

Finding the location of the face helps identify the region that contains all the features required to continue with facial expression recognition. The rest of the image is not important for this purpose.

- See Figure 2.3, Reddy and Srinivas applied a fixed oval shaped mask over the image to extract the face region [9]. The images they used only contained faces, making it easier to apply the masks.

- Boubenna and Lee used the Viola and Jones algorithm to detect the location of the face in an image [2]. This algorithm uses Haar-like features

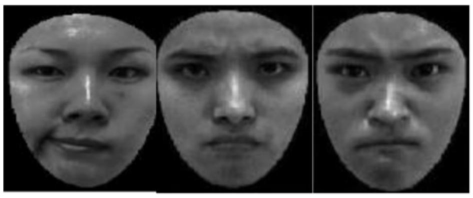


Figure 2.3: Preprocessed Image with Oval Masks

to help find the facial features, such as the eyes, nose and mouth. The Ada Boost algorithm is used to reduce the number of features, if there are too many. They used the canny edge detection operator to detect the edges of the face [10].

2.3 Feature Extraction - Extract the Features from the Face

Once the face has been detected, it is important to identify which features will be used for feature extraction. Either the full-face, or individual features from the face can be used as part of the feature set. These individual features can be the eyes, nose, mouth and eyebrows. The feature extraction algorithm can be applied based on its compatibility with the features chosen.

- Goyal and Mittal extracted the nose, mouth and eyes using the Viola and Jones Haar classifier [8].

- Reddy and Srinivas considered the entire face for the feature extraction not just the eyes, mouth and nose individually . First, they used Gabor filters to generate a bank of filters at 5 spatially varying frequencies and 8 orientations. The filtered outputs were then concatenated. Principle Component Analysis (PCA) was used to reduce dimensionality. PCA is a statistical technique that reduces the dimensions of feature vectors. The high dimensionality of feature vectors can cause over-fitting during classification. The PCA algorithm generates the eigenfaces for each image of dimension $N \times N$. From this their system generated the eigenvector of dimension $2N$ for each image. The vectors that relay the distribution of the face images the best are selected. These vectors are used to define the subspace, i.e., the "face space", of the face images [9].

The face image subspace represents a lower dimensional space $2N$ of the original image with dimensions $N \times N$.

- Boubenna and Lee used Pyramid Histogram of Oriented Gradients (PHOG) to extract features. PHOG represents an image by its local shape and the spatial layout. The local shape of an image is represented by a histogram of edge orientations within an image sub-region, which are divided into K bins. The spatial layout is represented by tiling the image into regions at different levels. Each image is divided into a sequence of increasingly finer spatial grids by repeatedly doubling the number of divisions in each axis direction [11]. The parameters of PHOG were set as follows: 3 for number of levels, 360 degrees for the number of dimensions and 16 for the number of bins. To decrease the number of features, a Genetic Algorithm (GA) was used, which resembles natural selection to find optimal features [10].

2.4 Training the machine learning technique

The training of the machine learning technique based on supervised learning whereby the machine learning technique is given labelled images, *Happy, Sad, Anger, Disgust, Surprise, Fear* and *Neutral*, and is required to learn them. Once the machine learning technique has completed its training, it can then be fed unlabelled images, and the result would be a prediction of which label best suits the given image.

- Goyal and Mittal used an Artificial Neural Network, with one hidden layer. The neural network architecture has three layers: input, hidden and output layers. Figure 2.4 provides a visual layout of the architecture of an individual neuron and a ANN with multiple layers. Feed-forward ANNs allows the signal to travel in one direction from the input layer to the output layer. Recurrent networks contain feedback connections, where the signal moves in both directions. To get accurate results from the ANN, the weights can be set explicitly using prior knowledge, or the ANN can be trained to help find the optimal weights [8, 12].

- Reddy and Srinivas, used an Artificial Neural Network, with two hidden layers [9].

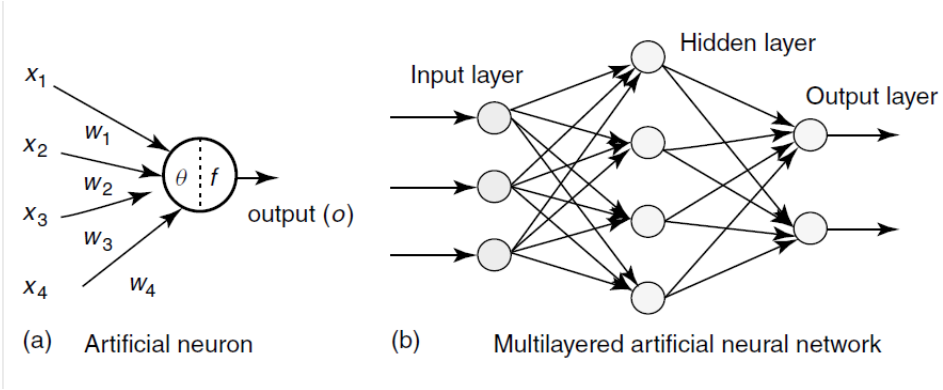


Figure 2.4: Architecture of an artificial neuron and a multilayered neural network

- Boubenna and Lee, used Linear Discriminants Analysis (LDA) and K Nearest Neighbours (KNN). LDA finds the maximum distances within classes, to obtain maximum class separation. LDA only uses up to second order moments, such as the covariance and mean, of the class distribution. KNN classifies unlabelled samples according to the training samples. KNN finds the nearest K in the labelled samples and set them to the closest group, for the unlabelled samples. One distance measure is required, and [10] used the cosine distance measure.

2.5 Results

The results from the three studies are as follows, with [10] having the best overall results for their facial expression recognition system.

- Goyal and Mittal achieved an 80% classification accuracy, using a confusion matrix and a regression plot to verify the results [8].

- Reddy and Srinivas achieved an 85.7% classification accuracy using the JAFFE database [9].

- Boubenna and Lee achieved a 99.33% accuracy, using the Radboud Faces Database (RaFD) [10].

Chapter 3

Image Processing Techniques

3.1 Introduction

This chapter looks at image processing techniques used in obtaining the features needed to do the final classification. The Viola and Jones algorithm, developed by Viola and Jones, is used to detect the location of the frontal face in the image. Once we have this location we then extract the face, which represents our region of interest. The region of interest is then Gray-scaled and is now ready for feature extraction. The Histogram of Oriented Gradients is used for the feature extraction process.

The rest of this chapter is organised as follows: Section 3.2 provides details on Viola-Jones Object Detection and it's key concepts; Section 3.3 covers the image preprocessing techniques used and Section 3.4 explains how Histogram of Oriented Gradients are used for feature extraction. The code implementation for the Viola-Jones face detection and the HOG feature extraction can be found in Appendix B, Section B.1.1.

3.2 Viola-Jones Object Detection

The Viola-Jones algorithm [2] is an object detection method that uses Haar-like features. For this project the Viola-Jones object detection is used to find the location of frontal faces in images. The algorithm uses three concepts to effectively detect objects with certain features:

The first concept is the integral image, which allows for the features in the image to be evaluated much faster. This is also known as an intermediate view of the image. At each point (x, y) in the integral image, there is the sum of the pixels above and to the left of the point (x, y),inclusive.

Referring to Figure 3.1: In order to calculate the sum of the pixels within the rectangle D, only four original image references are needed.

- At point 1 in the integral image, the sum of all the pixels in rectangle A in the original image are used.
- At point 2 in the integral image the sum of all the pixels in rectangles A and B in the original image are added together $(A + B)$.
- At point 3 in the integral image the sum of all the pixels in rectangles A and C in the original image are added together $(A + C)$.
- Lastly, at point 4 in the integral image of all the rectangles are added together $(A + B + C + D)$ in the original image.

Thus, to get the sum of the pixels in rectangle D will result in $4 + 1 - (2 + 3)$.

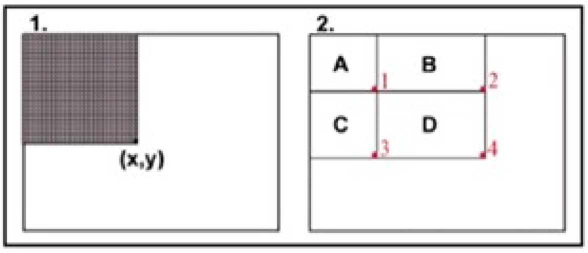


Figure 3.1: Integral Image [2]

The second concept in the Viola-Jones framework [2] is a classifier based on reducing a large feature set down to a smaller set of important features. This is done by using Ada-Boost. Ada-Boost finds a weak classifier and forces it to depend on a single feature, resulting in a stronger classifier. A weak classifier is selected at each stage of the boosting process, or feature selection process. The third concept is a method that combines weak classifiers in a rejection cascade. This increases the speed of detection as the focus is now on promising areas of the image. Each stage in the cascade is formed using Ada-Boost. The Viola-Jones algorithm uses many Haar-like features. I will describe Three of these Haar-like features.

- Two-rectangle feature – is calculated by subtracting the sum of the pixels of one rectangle from the sum of the other. The rectangles need to be the same size, shape and need to be vertically or horizontally adjacent.

- Three-rectangle feature – is calculated by subtracting the sum of the two outside rectangles from the middle rectangle.
- Four-rectangle feature – is calculated by subtracting the sum of the pixels of one diagonal pair from the other.

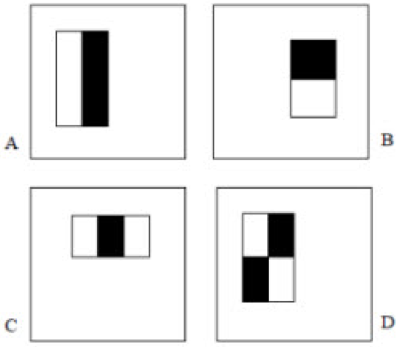


Figure 3.2: Haar-like features [2]

3.3 Image Preprocessing

3.3.1 Resizing

The image is resized once we obtain our region of interest using Viola-Jones to detect the location of the face. The main benefit of resizing the image is to maintain uniformity in our feature set. Another benefit is that it scales down the number of pixels in a image, resulting in a smaller feature set [13].

3.3.2 Gray Scaling

Converting an image from RGB, colour, to Grayscale helps in reducing the number of colour channels down to a single color channel. A commonly used method is the standard NTSC conversion formula, that calculates the luminance of a pixel [13]:

$$\text{Luminance of a pixel} = (0.2989 \times red) + (0.5870 \times green) + (0.1140 \times blue)$$

3.4 Histogram of Oriented Gradients

The Histogram of Oriented Gradients is a feature extraction method for images. Where a image is divided into cells, of $C_x \times C_y$ pixels, that form a grid over the image. Histograms are calculated for each of these cells based on

the orientation of the gradients of the pixels in the cell. This is followed by the image further being divided into a grid, of $B_x \times B_y$ cells, which are called blocks. Each block is used to contrast-normalize the histograms (cells) present in the block. The dimensions of the final feature vector calculated by:
total number of blocks×number of cells in each block×number of orientation bins

Further details of the Histogram of Oriented Gradients will be discussed following Dalal & Triggs HOG feature extraction chain,see Figure 3.3, excluding the linear SVM and classification [3].

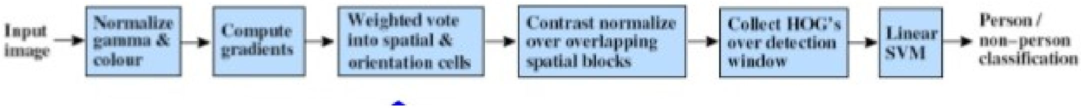

Input image
Normalize gamma & colour
Compute gradients
Weighted vote into spatial & orientation cells
Contrast normalize over overlapping spatial blocks
Collect HOG's over detection window
Linear SVM
Person / non−person classification

Figure 3.3: HOG feature extraction chain [3]

3.4.1 Input Image

Given an image, first identify the region of interest in the image. This region then forms your image window.

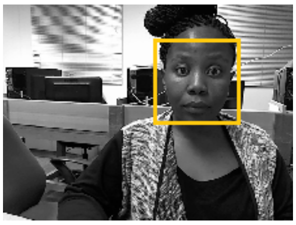


Figure 3.4: Region of interest

Figure 3.5: Image window

3.4.2 Normalize Gamma & Colour

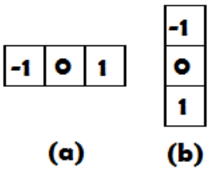
Normalizing the image window is an optional addition to the HOG. Dalal & Triggs found that normalizing the image pixels (p) at this the stage did not have a noticeable impact on the performance at the detection stage of their research. However when choosing to normalize the image window Gamma (power law) normalization had a negative impact on the results, while Square-root normalization had more of a positive impact on the results.

Gamma Normalization: $log_(p)$

Square-root Normalization: $\sqrt{p}$

3.4.3 Gradients

The gradient for the image window is computed by applying a one dimensional mask in both X (G_x) and Y (G_y) directions.



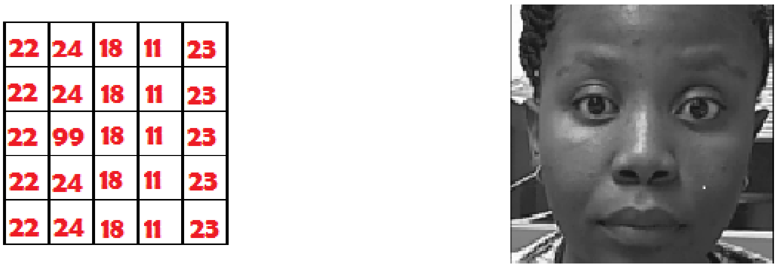
-1	0	1

-1
0
1

Figure 3.6: One dimensional masks, (a) X-direction and (b) Y-direction

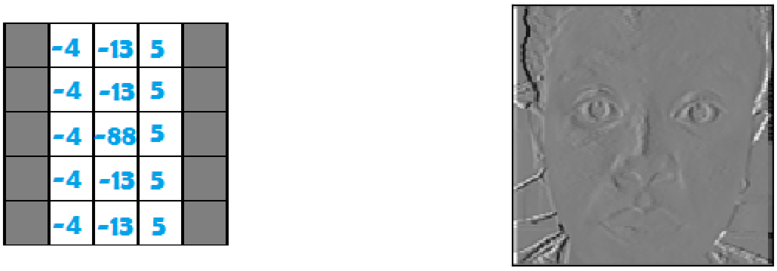
The mask convolves over the image window. At each point where the mask is placed the pixels are multiplied by the mask. After that the two outer pixels are added together and the result is placed in the position of the center pixel. The mask is not able to compute the gradients on the pixels around the edge of the image. Unless extra pixels are added to the edges of the image before hand. The example below shows how the loss in pixels affects the resulting gradient image.

3.4.3.1 Example:



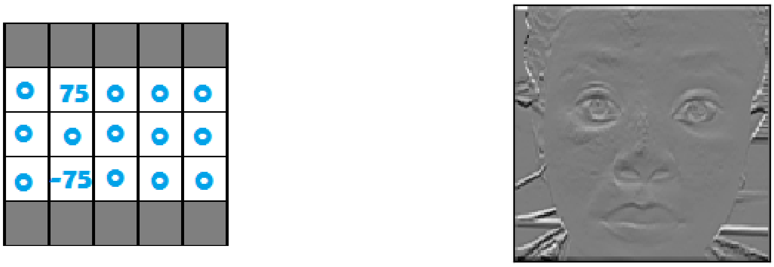
22	24	18	11	23
22	24	18	11	23
22	99	18	11	23
22	24	18	11	23
22	24	18	11	23

Figure 3.7: Image window



	-4	-13	5	
	-4	-13	5	
	-4	-88	5	
	-4	-13	5	
	-4	-13	5	

Figure 3.8: The result of a one dimensional mask applied in the X-direction



0	75	0	0	0
0	0	0	0	0
0	-75	0	0	0

Figure 3.9: The result of a one dimensional mask applied in the Y-direction

Now that we have the gradients, we can compute the magnitude and orientation of the gradients from G_x & G_y.

Magnitude: $G = \sqrt{G_x + G_y}$

Orientation: $\theta = \arctan(\frac{G_y}{G_x})$

3.4.4 Weighted Vote in Spatial & Oriented Cells

We can now decide on the dimensions of each cell, before calculating the HOGs. In their research Dalal & Triggs found that the size of the cell are dependent on the size of the features you need to extract, e.g eyes, nose, mouth.

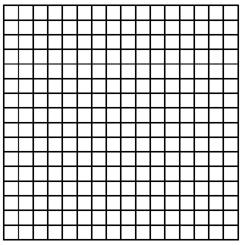


Figure 3.10: 16 × 16 pixel image

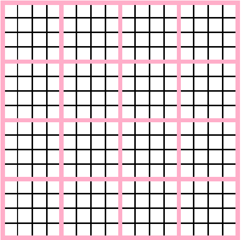


Figure 3.11: Image divided into 4 × 4 pixel cells

The next parameter is the number of orientation bins. The orientation of the gradient can be described as the angle of the gradient. There are two options available when choosing the range of the gradient angle:

- Signed [0°, 360°]

- Unsigned [0°, 180°]

Unsigned gradients in the range [0°, 180°], with the number of orientation bins in the range [9, 12] are the preferred values for the orientation bins.

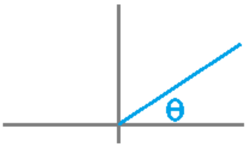


Figure 3.12: θ as an angle

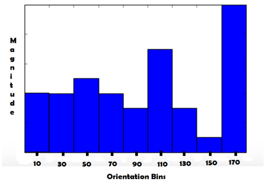


Figure 3.13: Unsigned gradients with 9 orientation bins

Looking at a single cell. Each pixel of the gradient magnitude image contributes to a orientation bin of a cells histogram. The value of the same pixel in the gradient orientation image helps you identify which orientation bin to place the gradient magnitude of the pixel.

3.4.5 Contrast Normalize over Overlapping spatial cells

Contrast normalization is used to ensure that the cells are not affected vastly by changes illumination and contrast in the image. Starting with dividing the image into blocks that can fit at least 2-3 features, these blocks are allowed to overlap one another for more detailed feature set. Contrast normalization works by taking the sum of the histograms in a block S_b and dividing each of the histograms H_{hist} by $\sqrt{S_b^2+\epsilon^2}$. The result is a normalized histogram H_{norm} in each cell.

$$\text{Contrast normalization : } H_{norm} = \frac{H_{hist}}{\sqrt{S_b^2+\epsilon^2}}$$

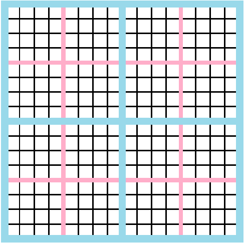


Figure 3.14: Blocks of $B_x \times B_y$ cells

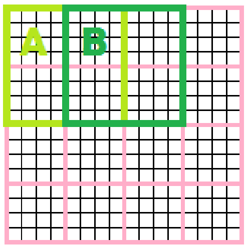


Figure 3.15: Block A & B with a 50% overlap

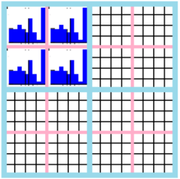


Figure 3.16: Cell histograms for contrast normalization in a block

3.4.6 Collect HOGs over Detection Window

The final step is to concatenate all the normalized histograms to form a one dimensional feature vector $[H_{norm}, H_{norm}, H_{norm}...]$. This feature vector is then used for the classification and training of the system.

Chapter 4

Implementation

4.1 Introduction

This chapter looks at the high-level and low-level views of the system and code documentation. The high-level view in Section 4.2 provides an outline of the processes followed during the implementation of the system, while the low-level view in Section 4.3 provides a more detailed description of the implementation of the system.

4.2 High-Level View of the System

The high-level view of the system provides an overview of all the stages that the system follows when classifying an image given as input. These stages include: Capture Frame, Face Detection, Feature Extraction, Train Machine Learning Technique and Emotion Classification. Figure 4.1 serves as a visual aid for the content that follows.

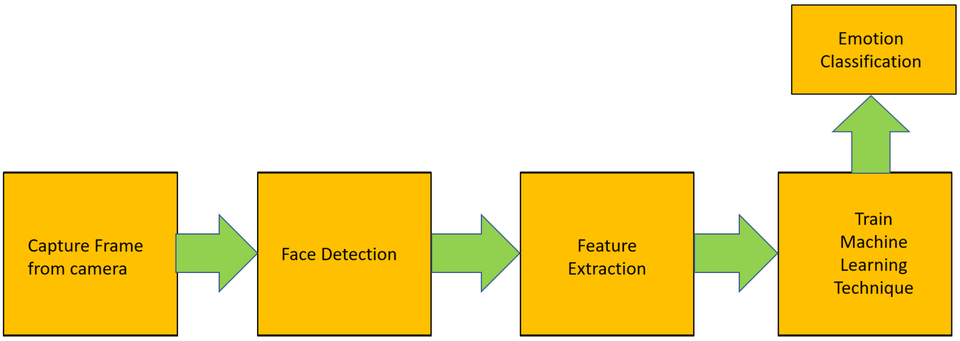


Figure 4.1: High-Level View of System

Looking at Figure 4.1, the High-level view is explained as:

1. **Capture Frame** – The web camera records a constant stream of video input. The video input consists of a sequence of multiple image frames.

The system captures each frame for processing as it is displayed on the video feed.

2. **Face Detection** – Now that we have captured a single frame, we need to check if there is a face present in the frame. This is done using a face detection algorithm. If a face is present in the frame, the location of the face is extracted. The rest of the image is disregarded at this point.

3. **Feature Extraction** – Every emotion displayed facially has it's own set of unique identifying features. By applying feature extraction we are able to represent these features in a way that a computer can understand and process. The feature extraction method is applied to the region of the image that contains the face.

4. **Train Machine Learning Technique** – Machine learning is a method used by computers to learn how to identify patterns in a given set of features. This process is called training. When we train the system, our features are labelled numerically from 1 to 7 and correspond with the classes (*Angry, Disgust, Fear, Happy, Sadness, Surprise* and *Neutral*). Labelling the features helps guide the computer in the learning process.

5. **Emotion Classification** – When the training is complete, classification helps to test the accuracy of the trained model. At this point the model should be able to identify emotions given unlabelled features.

4.3 Low-Level View of the System

The high-level view of the system dives deeper into the details of the components used to implement the system. This is done following the same stages used in the high-level view of the system. In this section we will look at three conceptual low-level views that relate to our system. These views are aligned to the processes followed in image processing, Support Vector Machine model testing & training and implementing the final system.

4.3.1 Low-Level View of Image Processing

The first low-level view is a visual representation of how the high-level view relates to the image processing techniques discussed in Chapter 3 is presented in Figure 4.2.

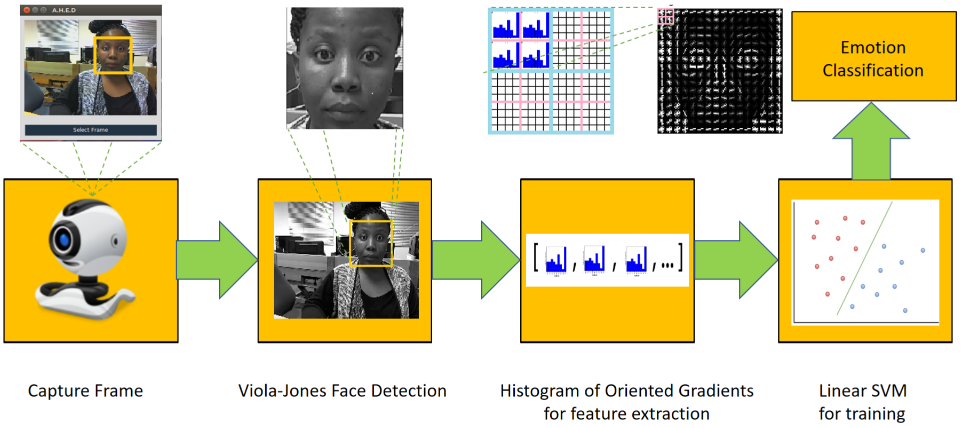

A.H.E.D
Select Frame
Emotion Classification
Capture Frame
Viola-Jones Face Detection
Histogram of Oriented Gradients for feature extraction
Linear SVM for training

Figure 4.2: Low-Level View of Image Processing

4.3.2 Low-Level View of SVM Model Optimization, Testing & Training

The second low-level view relates to the training and testing of the SVM model used to classify the emotions. Where the 'Capture Frame' stage is replaced by 'Get Images from Dataset'. The results given in this section aim to contextualize the information given. The results pertaining to the testing or classification of the SVM model after training will be discussed further in Chapter 5. The code used to implement the SVM Optimization, Testing and Training can be found in Appendix B, Section B.1.2.

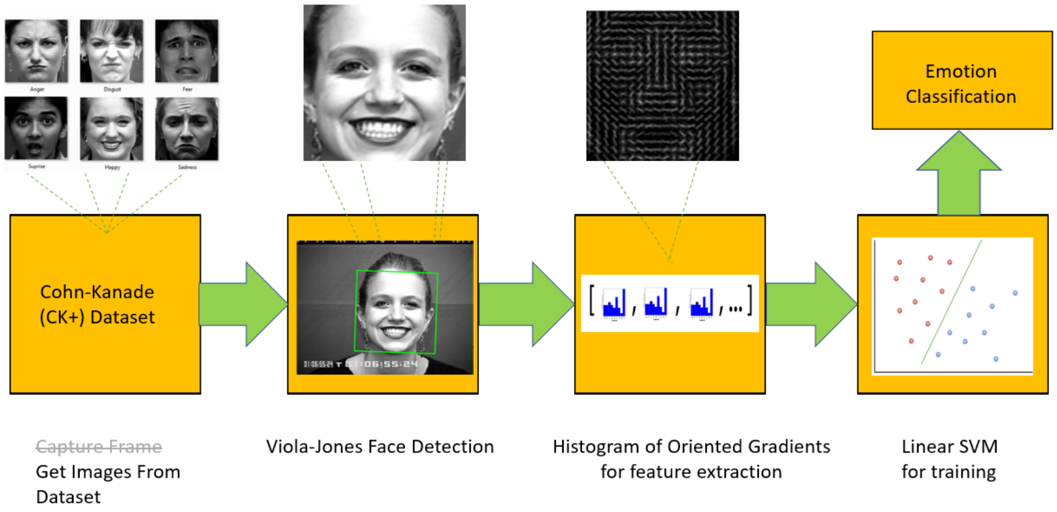

Cohn-Kanade (CK+) Dataset
Emotion Classification
~~Capture Frame~~
Get Images From Dataset
Viola-Jones Face Detection
Histogram of Oriented Gradients for feature extraction
Linear SVM for training

Figure 4.3: Low-Level View of SVM Model Testing & Training

Looking at Figure 4.3, the Low-level view for SVM model testing and training is explained as:

1. **Get Images From Dataset** – The Extended Cohn-Kanade Dataset [5] is used as data for the training and testing of our SVM model. The images are divided into seven groups, whose labels are: *Angry, Disgust, Fear, Happy, Sadness, Surprise* and *Neutral.* The Table 4.1 provides the total number of images present for each label.

Emotion Label	**N**
Angry	45
Disgust	59
Fear	25
Happy	69
Sadness	28
Surprise	83
Neutral	35

Table 4.1: CK+ Image Labels and Totals

2. **Face Detection** – The Viola-Jones face detection is used to extract the face from each image in the CK+ dataset. The face region of the image is stored as a 56 × 56 pixel grayscale image.

3. **Feature Extraction** – The resulting images from the face detection are used as inputs for the Histogram of Oriented Gradients which gives

a one-dimensional feature vector for each image. Each image is given a numerical label from 1 to 7 based on the emotion displayed in the image, see Table 4.2 for the labels. The feature vector is stored in the feature dataset with the corresponding numerical emotion label(e.g. [Numerical Label][Feature Vector]).

Emotion Label	Angry	Disgust	Fear	Happy	Neutral	Sadness	Surprise
Numerical Label	1	2	3	4	5	6	7

Table 4.2: Emotions with corresponding Labels for the feature vectors

The parameters used for the HOG are listed in Table 4.3. The optimization of the HOG parameters is discussed in Section 4.3.4.

Parameter	**Size**	**Type**
Image	$(56, 56)$	Pixels
Cell	$(4, 4)$	Pixels
Block	$(3, 3)$	Cells
Overlap	66.66%	Blocks
Bins	9	(0°-180°) Unsigned Gradients
Feature Vector Size: 11664		

Table 4.3: HOG Parameters

4. **Train Machine Learning Technique** – The machine learning technique used to do the classification for our system is Support Vector Machines. [5] Used SVMs to test the accuracy of their CK+ dataset due to its proven accuracy with face and facial action detection. Binary classification is the simplest example for explaining how SVMs work. Where the SVM attempts to separate the closest negative and positive points in each class from each other. Once this separation is achieved it makes it easier to separate negative and positive points that are further away from each other, as the similarities in these points are less than those in the points that are closer. The distance between these points is calculated by subtracting the positive and negative points from each other. The positive and negative points closest to each other are considered our support vectors, these help in determining the separating hyperplane. The SVM needs to ensure that the distance between the

support vectors is maximized. Figure 4.4 visualizes the concepts discussed.

Considering that we have multiple classes in our dataset the ‘one versus all’ method was used to train the SVM model. This is where one class is labelled as positive (c) whilst the other classes are labelled as negative ($n - c$). All of the classes (n) are given the opportunity to be labelled as the positive class and the SVM model is generated.

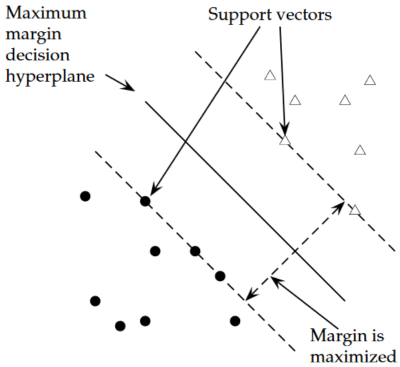


Figure 4.4: Key Functions of a SVM [4]

(a) *Cross-Validation* – The feature dataset compiled after completing the feature extraction is used as input for the SVM. Where the dataset is divided into a training and testing dataset. The percentage of the split is dependent on how well the model performs with each split. A 60% training and 40% testing split was used for the SVM. Both these datasets should contain an equal distribution of the emotion labels in each dataset. This step ensures that all labels appear equally in both datasets. The Cross-validation score using a stratified K-fold of 3 is approximately 84%. Cross-validation measures the overall performance of the SVM model on different training and testing data splits [14]. This tests the independence of the model to the dataset and helps to prevent overfitting in our model. K-fold cross-validation is done by splitting the dataset into K subsets of equal length. Each subset is then tested on the SVM model of the remaining K-1 subsets. Stratified K-fold cross-validation ensures that the classes are distributed equally in each subset. The training dataset is used as input for training our SVM and creating the SVM model that will be used in our system.

(b) *Grid-Search* – After optimizing the SVM model with grid-search a Linear Kernel with a C of 1 was used. Grid search helps to find the optimal parameters for the SVM model. Where C determines the extent to which the SVM model should avoid misclassification in the training [14]. Larger values of C decrease the margin of the hyperplane which aims to increase the accuracy of the training. A smaller value for C increases the margin of the hyperplane, but carries the downside of more misclassified training data points. In [14] it is recommended that the optimal range of that should to be investigated in order to find the best value for C is $C = 2^{-5}, 2^{-3}, ..., 2^{15}$.

5. **Emotion Classification** – The testing dataset is used to asses the performance of the the trained SVM model on unseen data. Where the SVM model is given the testing dataset features without the corresponding labels. The results of the SVM model classification are then compared to the original labels to test the accuracy of the SVM model. The SVM model trained for our system has an overall accuracy score of 88.2%.

4.3.3 Low-Level View of Final System

The third low-level view shows how the process of Automatic Human Emotion Detection is streamlined for user interaction. Where classifications are performed live as the user changes their facial expressions. At this point we use the 'Trained SVM Model' with the HOG parameters from Table 4.3 above.

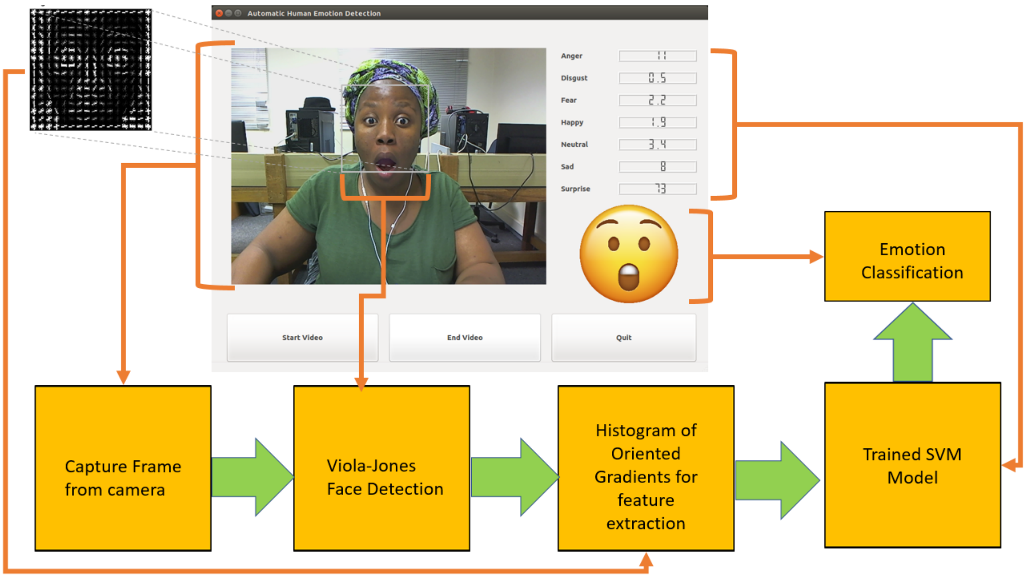

Automatic Human Emotion Detection
Anger
11
Disgust
0.5
Fear
2.2
Happy
1.9
Neutral
3.4
Sad
8
Surprise
73
Start Video
End Video
Quit
Emotion Classification
Capture Frame from camera
Viola-Jones Face Detection
Histogram of Oriented Gradients for feature extraction
Trained SVM Model

Figure 4.5: Low-Level View of Final System

4.3.4 Optimizing HOG features

This section covers the different combinations for the HOG parameters that were considered before the values in Table 4.3 were chosen. The HOG implemented from scratch by us is compared to the OpenCV HOG using the Python Sklearn SVM library. All the other parameters remained the same at this point, the dataset had a 60% training and 40% testing dataset. The SVM model optimized to a Linear Kernel with a C of 1 for each iteration of the HOG features optimization. Both HOGs maintained a bin size of 9 and 50% overlap for Block $(2, 2)$, with a 66.66% overlap for Block $(3, 3)$.

Accuracy of HOG Parameters		
-	**Cell** $(8, 8)$ **Block** $(3, 3)$	**Cell** $(4, 4)$ **Block** $(3, 3)$
OpenCV HOG	86%	88.2%
AHED HOG	77.9%	75.7%
	Cell $(8, 8)$ **Block** $(2, 2)$	**Cell** $(4, 4)$ **Block** $(2, 2)$
OpenCV HOG	83.8%	83.8%
AHED HOG	77.9%	82.3%

Table 4.4: HOG Optimization

4.4 Conclusion

The Automatic Human Emotion Detection system was implemented entirely using Python. OpenCV is used for the Viola-Jones face detection, Sklearn was used to implement the Histograms of Oriented gradients and the Support Vector Machines. The HOG implementation done from scratch used python numpy arrays and followed the implementation method discussed in Chapter 3. The highest accuracy achieved for the HOG implemented with OpenCV was 88.2% and 82.3% with the HOG implemented in this project.

Chapter 5

Testing

5.1 Introduction

This chapter looks at the results from training, testing and optimizing of the SVM Model selection process, in Section 5.1.1. Table 5.1 gives an overview of the formulas and terms used in Figure 5.2 and Table 5.2 to describe the results.

SVM Model Evaluation		
Term	**Formula**	**Description**
Type I Error	FP	False Positive
Type II Error	FN	False Negative
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	Evaluates the degree of correctness for the predictions
Precision	$\frac{TP}{TP+FP}$	The Positive predictive value
Recall	$\frac{TP}{TP+FP}$	True positive rate
F1-Score	$2 \times \frac{precision \times recall}{precision+recall}$	Evaluates the accuracy of predictions

Table 5.1: Terminology and formulas used for evaluating the SVM Model [1]

5.1.1 Analysis of SVM Testing Results

The test set for testing the SVM model consisted of 136 entries, which is 40% of the initial CK+ dataset. The overall accuracy of the SVM Model is 88.2%, across all subjects and emotions. Table 5.2 contains the 'SVM Model Classification Report'. The report indicates the performance of each individual class, or emotion, and the overall estimated performance of the SVM model with regards to the precision, recall and f1-score. Classes that had more data available performed better overall as compared to those that had less. Since there was more testing data available for these classes. Working with an uneven dataset, see Figure 5.1, makes it harder to judge the performance of each class in comparison to the other classes.

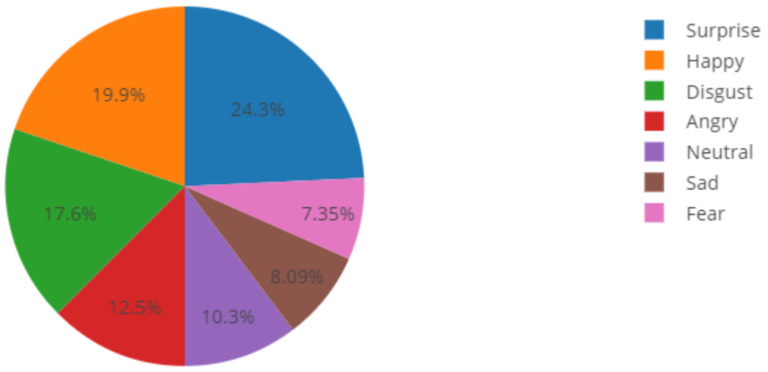


Figure 5.1: SVM test set breakdown

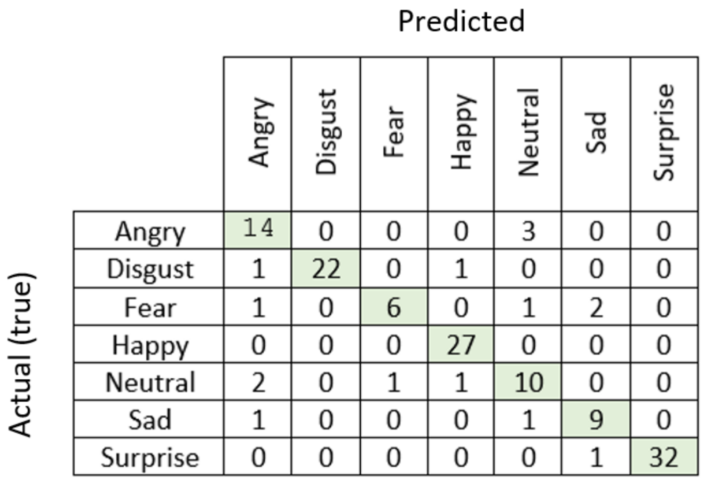
The emotions labelled *Disgust, Happy* and *Surprise* had over 24 subjects in the test set. This resulted in a higher f1-score for prediction accuracy of these emotions. Where the emotions labelled *Sad, Neutral, Fear* and *Angry* had significantly lower f1-score and fewer subjects in the test set.

SVM Model Classification Report				
Label	**precision**	**recall**	**f1-score**	**Sample Total**
Angry	0.74	0.82	0.78	17
Disgust	1.00	0.92	0.96	24
Fear	0.86	0.60	0.71	10
Happy	0.93	1.00	0.96	27
Neutral	0.67	0.71	0.69	14
Sad	0.75	0.82	0.78	11
Surprise	1.00	0.97	0.98	33
Avg Total	0.89	0.88	0.88	136

Table 5.2: SVM Classification Report

5.1.2 Analysis of Confusion Matrix for Test Results

A confusion matrix summarises the outcomes of the classification based on the the actual labels of the testing data and those obtain from testing. The values obtained from the confusion matrix help with analyzing the SVM model. Figure 5.2 shows the confusion matrix for our SVM model. The diagonal starting at the top-left index till the bottom-right index contains all the true positives/negatives for our SVM model. Considering that a large portion of the test set lies on this diagonal, the model created is exceptionally stable and has a few random misclassifications.



		Predicted						
		Angry	Disgust	Fear	Happy	Neutral	Sad	Surprise
Actual (true)	Angry	14	0	0	0	3	0	0
	Disgust	1	22	0	1	0	0	0
	Fear	1	0	6	0	1	2	0
	Happy	0	0	0	27	0	0	0
	Neutral	2	0	1	1	10	0	0
	Sad	1	0	0	0	1	9	0
	Surprise	0	0	0	0	0	1	32

Figure 5.2: Confusion Matrix of SVM Model Classification

5.1.3 Analysis of SVM Testing Results for Subjects

The results in the table,In Figure 5.4 and 5.5, look at all the individual subjects within the test set and their classification performance using the SVM model. The original CK+ dataset is uneven in that not all subjects had all seven emotions present in the dataset. This made it challenging to split the dataset evenly based on subjects. The split was done by ensuring that each emotion had the same test-train ratio for the training dataset and the testing dataset.

In the results table, the green represents a correct classification under the indicated "Emotion Label" and the red blocks indicate a misclassification for that emotion and the misclassification is included in white text. Each subject had at least one emotion classified during testing, with a maximum of five emotions for subject 'S055'. The results in the table show that a vast majority of the blocks are shaded in green which indicates that most of the emotions were classified correctly, only a few of the blocks are shaded in red. None of the subjects had more than one misclassification, this indicates that the model is robust to changes in test subjects. See Figure 5.3 for a sample of the subjects in the CK+ dataset. The subjects are diverse in age, race and gender.

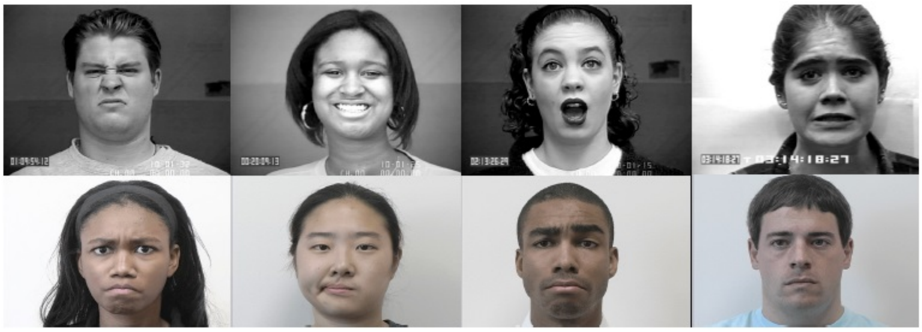
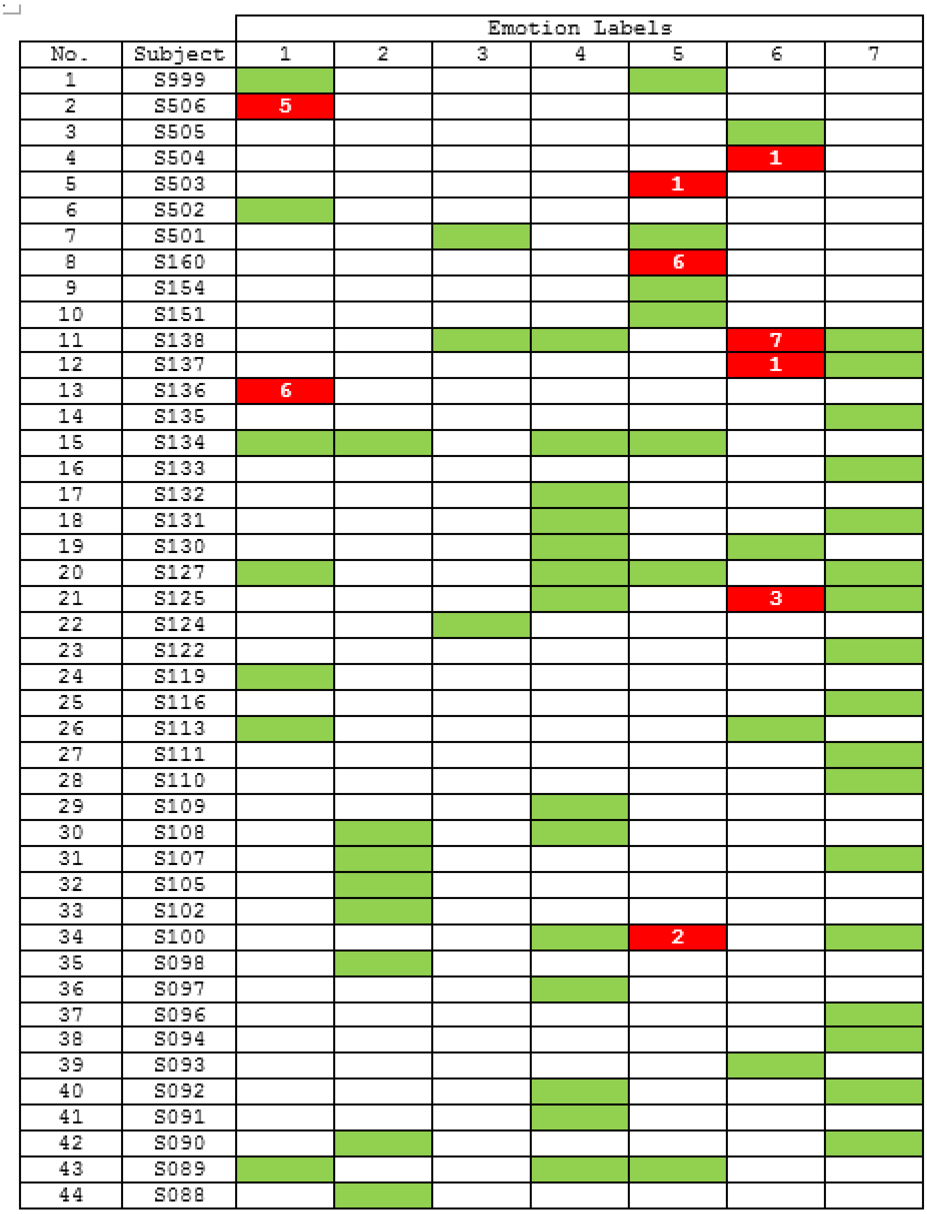
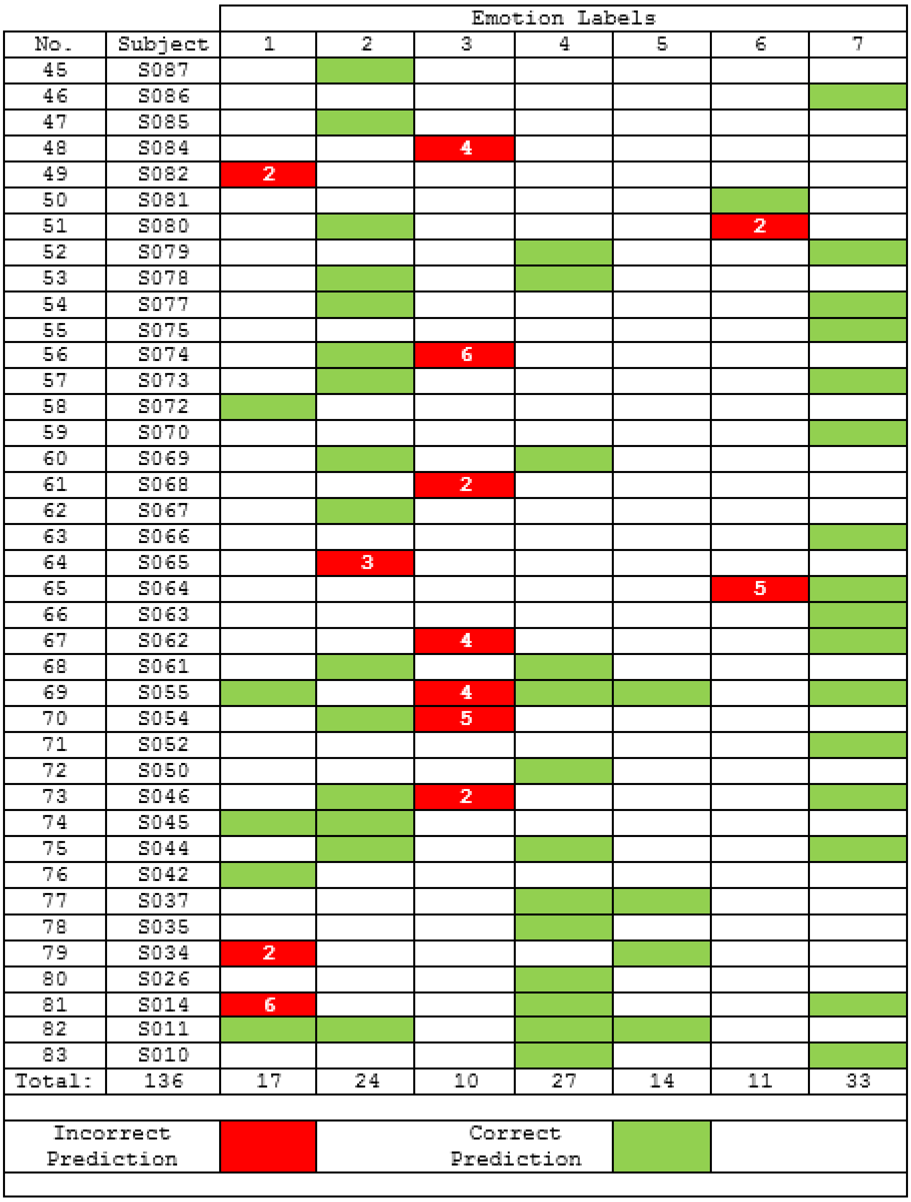


Figure 5.3: Subject sample from the CK+ Dataset [5]



		Emotion Labels						
No.	Subject	1	2	3	4	5	6	7
1	S999	🟩				🟩		
2	S506	🟥 5						
3	S505						🟩	
4	S504						🟥 1	
5	S503					🟥 1		
6	S502	🟩						
7	S501			🟩		🟩		
8	S160					🟥 6		
9	S154					🟩		
10	S151					🟩		
11	S138			🟩	🟩		🟥 7	🟩
12	S137						🟥 1	🟩
13	S136	🟥 6						
14	S135							🟩
15	S134	🟩	🟩		🟩	🟩		
16	S133							🟩
17	S132				🟩			
18	S131				🟩			🟩
19	S130				🟩		🟩	
20	S127	🟩			🟩	🟩		🟩
21	S125				🟩		🟥 3	🟩
22	S124			🟩				
23	S122							🟩
24	S119	🟩						
25	S116							🟩
26	S113	🟩					🟩	
27	S111							🟩
28	S110							🟩
29	S109				🟩			
30	S108		🟩		🟩			
31	S107		🟩					🟩
32	S105		🟩					
33	S102		🟩					
34	S100				🟩	🟥 2		🟩
35	S098		🟩					
36	S097				🟩			
37	S096							🟩
38	S094							🟩
39	S093						🟩	
40	S092				🟩			🟩
41	S091				🟩			
42	S090		🟩					🟩
43	S089	🟩			🟩	🟩		
44	S088		🟩					

Figure 5.4: SVM Testing and Training Results for Subjects



		Emotion Labels						
No.	Subject	1	2	3	4	5	6	7
45	S087		🟩					
46	S086							🟩
47	S085		🟩					
48	S084			4				
49	S082	2						
50	S081						🟩	
51	S080		🟩				2	
52	S079				🟩			🟩
53	S078		🟩		🟩			
54	S077		🟩					🟩
55	S075							🟩
56	S074		🟩	6				
57	S073		🟩					🟩
58	S072	🟩						
59	S070							🟩
60	S069		🟩		🟩			
61	S068			2				
62	S067		🟩					
63	S066							🟩
64	S065		3					
65	S064						5	🟩
66	S063							🟩
67	S062			4				🟩
68	S061		🟩		🟩			
69	S055	🟩		4	🟩	🟩		🟩
70	S054		🟩	5				
71	S052							🟩
72	S050				🟩			
73	S046		🟩	2				🟩
74	S045	🟩	🟩					
75	S044		🟩		🟩			🟩
76	S042	🟩						
77	S037				🟩	🟩		
78	S035				🟩			
79	S034	2				🟩		
80	S026				🟩			
81	S014	6			🟩			🟩
82	S011	🟩	🟩		🟩	🟩		
83	S010				🟩			🟩
Total:	136	17	24	10	27	14	11	33
Incorrect Prediction		🟥	Correct Prediction			🟩		

Figure 5.5: SVM Testing and Training Results for Subjects

5.2 Conclusion

The overall accuracy result for the SVM model was strong and generalized well with unseen data. Taking in to consideration that dataset used was not even, a further unbiased analysis on the results for individual subjects in the dataset was not possible. The dataset was split with the intention of having an equal ratio of subjects for each class in the testing set and the trainging set. As a result of this split analyzing the test results under each class proved to be more logical.

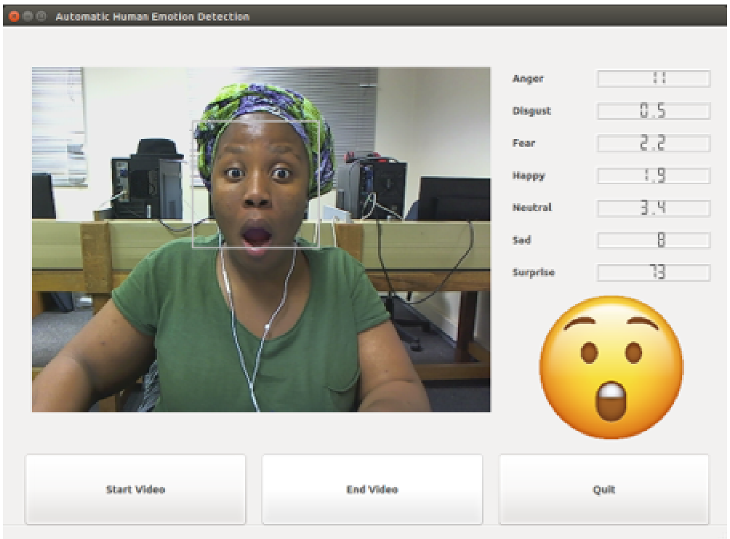
Appendix A

User Guide

A.1 Introduction

This chapter covers all that is required from the users computer when running the AHED system. The AHED system has an interface for user interaction and provides live feedback on facial expressions. The interface is simple, see Figure A.1, and has the following funtionalities:

- A live video output stream.
- Real time emotion classification on the right hand side of the screen. This is where each emotion probability is given for the frame.
- An Emoji(icon) that dispays the dominant emotion in the frame.
- ‘Start Video’ Button – Initiates the video output stream.
- ‘End Video’ Button – Pauses the video output stream.
- ‘Quit’ Button – Closes the program window.



Anger	11
Disgust	0.5
Fear	2.2
Happy	1.9
Neutral	3.4
Sad	8
Surprise	73

Figure A.1: The Graphical User Interface for the AHED System

A.1.1 System Requirements

These are the packages and programs needed to run the AHED system:

- Python 2.7
- PyQt4
- OpenCV
- Webcamera (preferably HD 1080p)
- Scikit-Image
- Numpy
- Pickle
- Windows 10 or Ubuntu 16.04 operating sytem (preferably x64 and needs to support the libraries and packages above)
- Note: Use JetBrains PyCharm on Windows, makes it easier to install missing libraries from python.

A.1.2 Instructions

Download 'AHED.zip' from 'http://cs.uwc.ac.za/ tlehata/index.html' under 'Term 4'. After ensuring that your system requirements meet those stated above, unzip the 'AHED.zip' folder. Open a terminal window or CMD window in the folder that contains 'guiAHED.py'. To continue follow the instructions under your desired operating system.

A.1.2.1 Ubuntu

- $ workon cv
- $(cv) python guiAHED.py

A.1.2.2 Ubuntu

- $ python guiAHED.py

A window should open up after a few seconds that looks the same as the one in Figure A.2. After that you can proceed and press ‘Start Video’ to start interacting with the system.

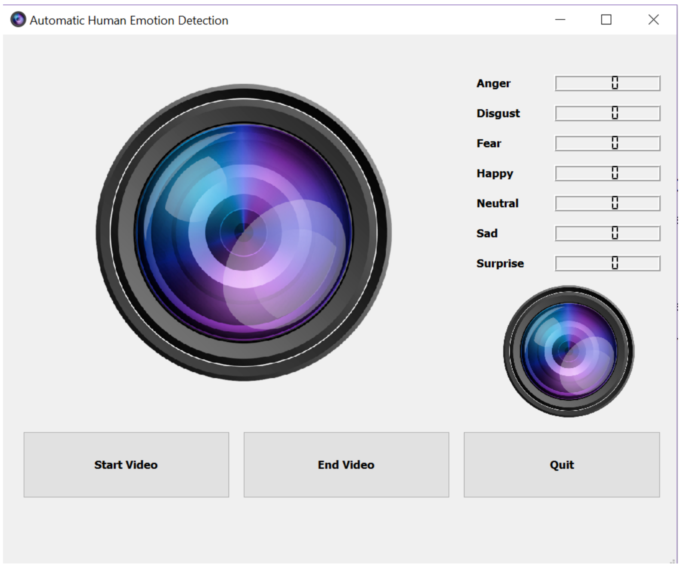


Figure A.2: Initial Start Interface for the AHED System

A.1.3 Feedback

To leave feedback or for further assistance and questions send me an email on ‘3342317@myuwc.ac.za’.

Appendix B

Code Documentation

B.1 Introduction

This chapter includes all the source code used for implementing the Feature Extraction and SVM Training and Testing for the Automatic Human Emotion Detection system.

B.1.1 Feature Extraction Python Code:

```
import sys
import cv2
import numpy as np
from skimage.feature import hog
from skimage import img_as_float

# a trained model for locating faces within an image
faceCascade = cv2.CascadeClassifier('Files/haarcascade_frontalface_alt.xml')

class FeatureExtraction():
    # ------------------------------------------------------------------------------
    # ----------------------------Viola-Jones Face Detection------------------------
    # ------------------------------------------------------------------------------
    def viola_jones(self, image):
        # set the height and width of face images
        height = 56
        width = 56

        # create an temporary 'image' of zeros
        scaled = np.zeros((height, width), dtype=np.float)

        # set the properties of the faceCascade
        faces = faceCascade.detectMultiScale(image, 1.3, 5)

        # set variables for finding the largest face in an image
        max_size = 0 # w*h
        X = Y = W = H = 0

        # keeping track of x,y,w,h in order to find the biggest face
        for (x, y, w, h) in faces:
            if max_size < w * h:
                max_size = w * h
                X = x
                Y = y
                W = w
                H = h

        # cater for when there are no faces in the image 'max_size = 0'
        if max_size != 0:
            # draw a rectangle identifying the location of the face in the image
            cv2.rectangle(image, (X, Y), (X + W, Y + H), (192, 192, 192), 2)

            # store the location with the face as our region of interest 'roi'
            roi = image[Y:Y + H, X:X + W]

            # convert the roi to grayscale
            gray = cv2.cvtColor(roi, cv2.COLOR_RGB2GRAY)

            # resize the roi so that all the roi's are uniform
            scaled = cv2.resize(gray, (height, width))

        return scaled, image

    # ---------------------------------------------------------------------------------------
    # --------------------------------------My HOG-------------------------------------------
    # ---------------------------------------------------------------------------------------
    # Note: gradients work only on grayscale images

    # calculate the gradient in the X direction (gx)
    def gx(self, scaled):
        y, x = scaled.shape
        scaled = np.lib.pad(scaled, 1, 'constant', constant_values=0)
        gx = np.zeros((x, y))
        # sobelX:
        # [1 ,0 ,-1]
        # [2 ,0 ,-2]
        # [1 ,0 ,-1]
        for i in range(y):
            a = np.convolve(scaled[i - 1, :], [1, 0, -1], 'valid')
            b = np.convolve(scaled[i, :], [2, 0, -2], 'valid')
            c = np.convolve(scaled[i + 1, :], [1, 0, -1], 'valid')
            gx[i, :] = np.sum([a, b, c], axis=0)
        return gx

    # calculate the gradient in the Y direction (gy)
    def gy(self, scaled):
        y, x = scaled.shape
        scaled = np.lib.pad(scaled, 1, 'constant', constant_values=0)
        gy = np.zeros((x, y))
        # sobelY:
        # [-1 ,-2 ,-1]
        # [0 ,0 ,0]
        # [1 ,2 ,1]
        for j in range(x):
            a = np.convolve(scaled[:, j - 1], [1, 0, -1], 'valid')
            b = np.convolve(scaled[:, j], [2, 0, -2], 'valid')
            c = np.convolve(scaled[:, j + 1], [1, 0, -1], 'valid')
            gy[:, j] = np.sum([a, b, c], axis=0)
        return gy

    # calculate magnitude of the gradient('intensity')
    def magnitude(self, gx, gy):
        magnitude = np.sqrt(gx ** 2 + gy ** 2)
        return magnitude

    # calculate orientation of the gradient('direction')
    def orientation(self, gx, gy):
        # output is the same output as cv2.cartToPolar
        orientation = (np.arctan2(gy, gx) / np.pi) % 2 * np.pi
        return orientation

    # calculate the HOG using an algorithm developed from my documentation
    def calculate_myhog(self, scaled):
        # gradients gx and gy
        gx = self.gx(scaled)
        gy = self.gy(scaled)

        # magnitude and orientation
        magnitude = self.magnitude(gx, gy)
        orientation = self.orientation(gx, gy)

        # orientation bins
        bin_n = 9
        bins = np.int32(bin_n * orientation / (2 * np.pi))

        # magnitude and orientation cells within a block
        bin_blocks = []
        mag_blocks = []
        epsilon = sys.float_info.epsilon

        # size of block
        blocksize = 3

        # size of cell
        cellsize = 4

        # store the height and width of the face image(roi)
        width = scaled.shape[0]
        height = scaled.shape[1]

        # create the parameters for the block slider
        y = ((height - (cellsize * blocksize)) / cellsize) + 1
        x = ((width - (cellsize * blocksize)) / cellsize) + 1

    # stores all the histograms in an image
    histograms = []

    # loops through each block in a image(i,j)
    # using a block "slider" to capture all posible blocks
    for i in range(0, y, 1):
        for j in range(0, x, 1):
            # magnitude and orientation cells within a block
            bin_block = bins[i * cellsize: i * cellsize + blocksize * cellsize,
                       j * cellsize: j * cellsize + blocksize * cellsize]
            mag_block = magnitude[i * cellsize: i * cellsize + blocksize * cellsize,
                       j * cellsize: j * cellsize + blocksize * cellsize]

            tempHists = []
            sumHists = np.zeros((9,))

            # loops through each cell in a block(m,n)
            # this ensures that all the histograms in each cell are calculated
            for m in range(0, blocksize * cellsize, cellsize):
                for n in range(0, blocksize * cellsize, cellsize):

                    # magnitude and orientation cells within a cell
                    cellBin = bin_block[m:m + cellsize, n:n + cellsize]
                    cellMag = mag_block[m:m + cellsize, n:n + cellsize]

                    # temporarily store the histograms
                    tempHists.append(np.bincount(cellBin.ravel(),
                    cellMag.ravel(), bin_n))

                    # store the sum of the histograms within a block
                    # this will be used for block normalization
                    sumHists = np.sum([sumHists, np.bincount(cellBin.ravel(),
                    cellMag.ravel(), bin_n)], axis=0)

            # sum up all the bins
            sumHists = np.sum(sumHists)

            # normalize all the cell histograms in the block
            for hist in tempHists:
                histograms.append(np.divide(hist, np.sqrt(
                np.add(np.square(sumHists), np.square(epsilon)))))

            # store the magnitude and orientation for the block
            bin_blocks.append(bin_block)
            mag_blocks.append(mag_block)
    # create the HOG feature vector
    hist = np.hstack(histograms)
    return hist

# ----------------------------------------------------------------------------
```

```
    # ---------------------------------OpenCV HOG-----------------------------------
    # ------------------------------------------------------------------------------

    # HOG implementation from skimage, using prefered parameters
    def hog_opencv(self, image):
        image = img_as_float(image) # convert unit8 tofloat64 ... dtype
        orientations = 9  # orientation bins
        cellSize = (4, 4)  # pixels_per_cell
        blockSize = (3, 3)  # cells_per_block
        blockNorm = 'L1-sqrt' # {'L1', 'L1-sqrt', 'L2', 'L2-Hys'}
        visualize = True  # Also return an image of the HOG.
        transformSqrt = False
        featureVector = True
        fd, hog_image = hog(image, orientations, cellSize,
        blockSize, blockNorm, visualize, transformSqrt,
                            featureVector)
        return fd

    # testing features
    def main(self):
        image = cv2.imread('Files/lense.png')
                scaled, image = self.viola_jones(image)
        print(self.calculate_myhog(scaled).shape)
                fd = self.hog_opencv(scaled)
                print(fd.shape)


if __name__ == '__main__': FeatureExtraction().main()
```

B.1.2 SVM Training and Testing Python Code:

```
import sys
import pandas as pd
from sklearn import svm
from sklearn.model_selection import GridSearchCV, cross_val_score, StratifiedKFold,
     train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
import pickle


class Tester():
    def run_test(self, data, modelname):
        print '\n######################################################################'
        print '##########################~', modelname, '~#####################'
        print '######################################################################'

        dataset = pd.read_csv(data)

        X = dataset.ix[:, 1:].values
        y = dataset.ix[:, 0].values

        X_train, X_test, y_train, y_test = train_test_split(X, y,
        test_size=.40, stratify=y, random_state=40)

        # Grid search parameters:

        # C = np.logspace(-5, 15,num=21,base = 2.0)
        # gamma = np.logspace(-15, 3, num=19,base = 2.0)
        param_grid = [
            {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
            {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
        ]

               # choose svm type: SVC - Support Vector Classification(based on libsvm)
        svc = svm.SVC(probability=True)

        print '\n######################################################################'
        print '#############################~SVM␣Grid␣Search~########################'
        print '######################################################################'
        clf = GridSearchCV(svc, param_grid)
        print '\t␣Parameter␣Grid:\n', param_grid

        print '\n######################################################################'
        print '##########################~SVM␣Cross␣Validation~######################'
        print '######################################################################'
        skf = StratifiedKFold(n_splits=3, random_state=None, shuffle=False)
        scores = cross_val_score(clf, X, y, cv=skf, n_jobs=-1)
        print '\t␣Cross␣validation␣scores:␣\t', scores
        print '\t␣Cross␣validation␣Accuracy:
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣\t␣%0.2f␣(+/-␣%0.2f)' % (scores.mean(), scores.std() * 2)

```

```
        print '\n#####################################################################'
        print '#############################~SVM␣Train~#############################'
        print '#####################################################################'
        clf = clf.fit(X_train, y_train)
        filename = modelname
        pickle.dump(clf, open('Files/'+filename, 'wb'))
        print '\t␣Best␣score␣for␣classifier:\t', clf.best_score_
        print '\t␣Best␣C:\t', clf.best_estimator_.C
        print '\t␣Best␣Kernel:\t', clf.best_estimator_.kernel
        print '\t␣Best␣Gamma:\t', clf.best_estimator_.gamma
        print '\t␣SVM␣Best␣Estimator:\t', clf.best_estimator_
        print '\n\t␣SVM␣Grid␣Scores:␣\n', clf.cv_results_

        print '\n#####################################################################'
        print '#############################~SVM␣Predict~###########################'
        print '#####################################################################'
        clf = pickle.load(open('Files/'+modelname, 'rb'))
        y_pred = clf.predict(X_test)
        print 'SVM␣Classification␣Report:'
        print classification_report(y_test, y_pred,
                                 target_names=['Angry','Disgust','Fear',
                                 'Happy','Neutral','Sad', 'Surprise'])
        print 'SVM␣Confusion␣Matrix:'

        print confusion_matrix(y_test, y_pred, labels = [1,2,3,4,5,6,7])
        print '\t␣SVM␣Accuracy␣Score:', accuracy_score(y_test,y_pred,normalize=True)
        print 'file:', data

        print '\n#####################################################################'
        print '#############################~SVM␣Test␣Results~###########################'
        print '#####################################################################'
        for i in range(len(X_test_names)):
            print 'Subject:',X_test_names[i],'\t␣Label:',y_test[i],
                       '\t␣Prediction:',y_pred[i]

    def main(self):
        print 'Output␣is␣printed␣to␣Files/test.out'
        orig_stdout = sys.stdout
        output = open('Files/test.out', 'w+')
        sys.stdout = output
        self.run_test('Files/dataCsv.csv', 'finalized_model.sav')
        sys.stdout.close()
        sys.stdout = orig_stdout
        print 'Done␣^_^'
if __name__ == '__main__': Tester().main()
```

Bibliography

[1] Claude Sammut and Geoffrey I Webb. *Encyclopedia of machine learning.* Springer Science & Business Media, 2011.

[2] P. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 2(57):137–154, 2004.

[3] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference*, volume 1, pages 886–893, 2005.

[4] D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval.* Cambridge University Press, 2009.

[5] Patrick Lucey, Jeffrey F. Cohn, Takeo Kanade, Jason Saragih, Zara Ambadar, and Iain Matthews. The extended Cohn-Kanade dataset (CK+): A complete dataset for action unit and emotion-specified expression. In *Conf. on Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society*, 2010.

[6] A. Mehrabian. *Nonverbal Communication.* Current Content, CA, USA, 1984.

[7] P. Ekman. *Non-verbal Behavior and Communication.* Lawrence Erlbaum Association, New Jersey, 1977.

[8] R. Goyal and T. Mittal. Facial expression recognition using artificial neural network. *HCTL Open Int. J. of Technology Innovations and Research*, 10:1–10, July 2014.

[9] C. Satyananda Reddy and T. Srinivas. Improving the classification accuracy of emotion recognition using facial expressions. *International Journal of Applied Engineering Research*, 11(1):650–655, 2016.

[10] H. Boubenna and D. Lee. Feature selection for facial emotion recognition based on genetic algorithm. *12th International Conference on Natural*

Computation, Fuzzy Systems and Knowledge Discovery, 12:511–517, August 2016.

[11] A. Bosch, A. Zisserman, and X. Munoz. Representing shape with a spatial pyramid kernel. In *CVIR*, pages 401–408, July 2007.

[12] A. Abraham. *Handbook of Measuring System Design.* John Wiley & Sons, Ltd., OK, USA, 2005.

[13] K. Bose and S. Bandyopadhyay. Crack detection and classification in concrete structure. *Journal for Research*, 2(4):29–38, June 2016.

[14] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification. 2003.