USING NEURAL NETWORKS FOR FACIAL SENTIMENT ANALYSIS

By

Aaron Ward

Supervisor(s): Stephen Sheridan

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
B.SC IN COMPUTING
AT
INSTITUTE OF TECHNOLOGY BLANCHARDSTOWN
DUBLIN, IRELAND
2018

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of **B.Sc in Computing** in the Institute of Technology Blanchardstown, is entirely my own work except where otherwise stated, and has not been submitted for assessment for an academic purpose at this or any other academic institution other than in partial fulfillment of the requirements of that stated above.

Dated: 2018

Author: Aaron Ward

Abstract

The purpose of this project was to build a machine learning model to classify facial expressions for sentiment analysis. This analysis tool's use case can be applied to retail and marketing fields to replace approached of measuring customer service such as email surveys. The model was built using Python and TensorFlow and is tested using a proof-of-concept web application developed with Node.JS. The model was put under rigorous testing and the findings have been documented, which determined a testing accuracy of 84.47%.

Keywords: Artificial Intelligence, Convolutional Neural Networks, Facial Sentiment Analysis, Machine Learning, Tensorflow

Acknowledgements

I would like to thank my project supervisor, Stephen Sheridan, for hard work and time he has investment for this project and research. I would also like to thank my parent for supporting me throughout the course of the year.

Table of Contents

List of Tables

List of Figures

Abbreviations

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
DBN	Deep Belief Networks
LSTM	Long Short-Term Memory
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
ML	Machine Learning
NLP (1)	Neuro-Linguistic Programming
NLP (2)	Natural Language Processing

Chapter 1

Introduction and Background

The field of artificial intelligence and machine learning has become exponentially prominent in recent years. From business to social media, machine learning algorithms are being used to change the definition of efficiency and user experience. For example, machine learning algorithms are utilised by many large companies optimize the experience with image and voice recognition and photo searching (Deshpande, 2016). Companies such as Tesla Motors use computer vision for their self-driving automobiles, video sharing sites such as Youtube use machine learning to optimize the users preferred recommended videos and, in some cases, it is even used in by medical companies for detecting cancer in patients. For example, convolutional neural networks (CNN's) have been used by researchers for brain tumor segmentation (Havaei et al., 2015). The use of CNN's proved to be an appropriate method for tumour segmentation as the results can be given from a range of 25 seconds to 3 minutes (Havaei et al., 2015). Artificial neural networks have been used by radiologists for Computer-Aided detections systems (CADe) and Computer-aided Diagnosis systems (CADx) to improve the accuracy of diagnoses, early detections and to minimize the time spent on evaluation by doctors (Firmino et al., 2014). Many businesses even depend on artificial neural networks for their business model as they can be applied to many industries and disciplines. According to Bhargava and Gupta (2017) artificial neural networks are used in a range of business applications such as forecasting of sales, classification of spending patterns, market targeting, risk analysis and bankruptcy prediction, to name a few.

Thus, considering the benefits and implications that these approaches have, the proposed project aims to illustrate how artificial neural networks work, with the use case of performing facial sentiment analysis for a retail environment. This paper is made of several chapters covering the literature of the field of neural networks, an analysis chapter which will delve into the background and high-level specifications of the project and a system design and implementation. Furthermore, testing and evaluation results will be covered and a discussion about the overall project and work carried out.

Chapter 2

Literature Review

This chapter will review five papers in relation to the topics of facial detection and analysis use cases, the use cases of artificial intelligence in marketing and the process of building neural networks. A critical analysis shall be given on the strengths and downfalls of each paper and the insight shall be given into the relevancy of the findings in each paper to the proposed project.

2.1 Analyzing and Detecting Employee's Emotion for Amelioration of Organizations

Subhashini and Niveditha make the opening statement that emotions usually do not take any place in a work environment in current society. Although the expression of feelings is suppressed in places of work, they suggest that emotions can affect five major areas in competitive advantage. The five given aspects of competitive advantage are as follows: Intellectual Capital, Customer Service, Organizational Reactivity, Production, Employee appeal and retentivity (Subhashini and Niveditha, 2015). In order to counter this apprehensiveness to expression of emotions in the work place, Subhashini and Niveditha suggest the concept of a facial emotion tracking system that will map the facial expressions of an employee's face as they enter the company building.

The system architecture given by Subhashini and Niveditha briefly describes logistics of

the program. Most employees entering a building to an organization must swipe a card to clock into the work hours. They suggest that they have designed a new system that removes the need for card swiping, by performing facial recognition. Not only this, but the system also implements emotion detection. The employee looks into a camera that will prove their presence in the building but will also perform some emotion detection. The system was implemented in the C Sharp programming language and uses skin tone segmentation to detect facial features. The binary image is then converted to an RGB image and an inspection of every individual pixel if performed. If the RGB value is greater than 110, then the pixel colour is refactored to be a white pixel, otherwise it becomes a black pixel. This is done to make it easy to detect facial features in the video stream (Subhashini and Niveditha, 2015). Once detected, the image around the face is cropped. They then apply a Bezier Curve to the regions around the lips and eyes of the person being analysed. The results of the person's identity and emotional status are then stored within a database. They conclude their paper by explaining that this system can be used by management to gain an understanding of their employee's sentimental state.

2.2 Deep Learning for Video Classification and Captioning

Wu et al.'s paper provides an in-depth analysis on the methods for video classification and video captioning in terms of deep learning. They claim that because of the exponential growth in internet bandwidth and computing power, video communications are becoming more and more prevalent, therefore paving the way for new video understanding applications (Wu et al., 2016). They refer to current implementations to prove the growth of interest in the field of computer vision and video analysis, notably the ImageNet challenge.

Move over, they go on to give brief description of the two "deep learning modules" that have been used for visual analysis: Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). It is explained that LeCun et al. that developed LeNet-5 made

a break through when they developed a CNN using the Back-Propagation Algorithm. But its noted that this is limited in performance when the complexity of the tasks is increased. Deep belief networks were developed to train networks in a unsupervised manner in order to counter this problem (Wu et al., 2016). AlexNet, a CNN proposed by Krizhevsky et al. in 2012, introduced two way to increase the performance of CNN's using the Rectified Linear Units(ReLU) activation function and dropout to decrease overfitting (Wu et al., 2016). Secondly, RNN's are brought forward. Wu et al. explain the difference between CNN's and RNN's, stating that CNN's are all feed forward networks that do not use cycling, which can prove be disadvantageous when working with sequence labeling. Two issues can occur with RNN's: Vanishing Gradient and Exploding Gradients as short-term memory is used when cycling through the network. The solution given to this is an RNN variant called Long Short-Term Memory (LSTM).

2.2.1 Image-Based Video Classification using CNN's and LSTM's

They state that Karparthy et al. researched the common architectures for learning spatial-temporal clues in large video datasets. It appeared that models using single frames as input achieve similar results as models using stacks of frames. From this, Simonyan and Zisserman proposed the idea of the Two-Stream Approach, because of the cost effectiveness and time consumption that come with training 3D CNN's. This Two-Stream approach involves training the CNN on single and stack frames concurrently. Both outputs are put through a score fusion. The result is the weighted sum of both scores (Wu et al., 2016).

Although Two-Stream is a good approach, it is not sufficient as it is not capable of dealing with long video clips. Therefore, LSTM's are utilized as they do not suffer from the problem of vanishing gradients. It has been found that CNN's and LSTM's complement each other when working in conjunction with each other (Wu et al., 2016). They conclude their paper with a summary of the written topics about, regarding the growth for the need for video understanding applications, the used of CNN's and RNN's, in addition to the variants of

these deep learning modules.

2.3 Subject independent facial expression recognition with robust face detection using a convolutional neural network

As stated by Matsugu et al., difficulties may arise with facial recognition. In terms of a face being in a smiling-like state, could have different implications. As well as this, a facial recognition system should be able to work with a wide range of variability of faces. They then give some examples of past implementations such as facial recognition with rigid head movement by Black and Yackoob in 1995 and speak about how this does not meet the requirements of dealing with wide variance. A rule based system is proposed (Matsugu et al., 2003). With their model, layer trains on a module-by-module (module being the nose, eyes, mouth etc) basis. Meaning each layer trains on a certain facial feature. Each of the neurons perform an averaging of some local receptive fields then they use a skin tone detector to detect each module on the face (Matsugu et al., 2003). For training, Layer one and layer two are trained for 8 modules using back propagation. Layer three and four train on more complex feature detectors such as the mouth and eyes. The output is then sent to the rule based algorithm for handling variability and robustness.

The rule based algorithm takes the output of the CNN and measures the distance between the features. From these calculation, the rules are applied to determine if the person is in a laughing or smiling state. The rules are summarised as follows (Matsugu et al., 2003):

- The distance between eyes and lip get shorter.
- The horizontal length of lip gets longer.
- The eyes wrinkle.
- The gradient of lip from the end point to the end point increases.

- Detection of teeth increases.
- The edges (wrinkles) of cheek increase.

In conclusion, they received a 97.6 percent accuracy for 10 test subjects with 5600 images. They assert that their model is significantly more efficient as they only require one CNN due to their rule based algorithm, in contrast to Fasels implementation that uses two CNN's working in synergy with one another.

2.4 Neuromarketing - The Art and Science of Marketing and Neurosciences Enabled by IoT Technologies

A paper by Arthmann and Li describes the growing field of Neuromarketing with an opening statement: Advertisers recognise that there is a relationship between stimulating the emotions of a customer and influencing their actions. Online shopping has drastically affected the store sales, and it is also explained that more than 3500 stores have shut down due to bankruptcy in 2017 (Arthmann and Li, 2017). Their answer to this change in buying is Neuro Linguistic Programming(NLP). Neuro Linguistic Programming, not to be confused with natural language processing, is a method of observing the verbal and non-verbal communication of humans. Eye accessing cues (eye movements) are said to be linked to certain emotions or thoughts. Neuromarketing incorporates NLP and IoT devices to understand the consumer sentiment more extensively. Neuromarketing aims to remove marketing biases by utilizing the consumers subconscious. One example given by Arthmann and Li is the notion of facial coding and motion tracking, which is put in place to determine why consumer make certain decisions. Furthermore, they make their belief clear of retailers benefiting from this when it is put forward that artificial intelligence and machine learning will evolve, giving better results of consumer preference shifts. Additionally, these neuromarketing systems can replace thermoimaging people counting devices that clock people walking into stores that may not be eligible customers (children). Further examples are provided for these technologies. They go on to describe a scenario when a customer is given an image of a product and a

price in front of a webcam, and by performing facial coding and sentiment analysis, we may get a better sense of what the consumer is feeling. Arthmann and Li conclude their paper by declaring the future potential of these systems using when integrated with ”always on” IoT technologies.

2.5 Facial expression recognition with Convolutional Neural Networks: Coping with few data and the training sample order

Lopes et al. open their paper by explaining the definition of facial expression. They describe it as facial changes that occur with someone due to emotional state or social communication. In terms of facial expression recognition software, a lot of systems give misleading accuracy results due to the overlapping of training and test data. They explain that some problems may occur. For example: dealing with ethnicity and variability of faces. Their response to this is training with one data set and testing with another to provide more accurate results (Lopes et al., 2017). They make reference to Liu et al’s work on facial recognition by describing the three stages of training: Feature learning, which is responsible for extraction of all facial features. Secondly, Feature Selection, that selects the best features to learn. And lastly, the classifier, that each expression has one specifically allocated to. Lopes et al. then move on to explain convolutional neural networks (CNN) on a high level. Firstly, the CNN is comprised of a convolution layer, that if given a kernel size. This kernel shifts over the image given to generate a map. This is followed by sub-sampling. Sub-sampling is used to reduce the map size to increase the accuracy in variance. Lastly, the fully-connected layer is introduced. This is a neural network that has fully-connected neurons to it’s previous layer (Lopes et al., 2017). They explain that of all the facial recognition methods, CNN’s prove to be the most advantageous because they can use raw image data input for an accurate prediction.

A related work section is then brought forward to explain similar models that have been developed. Much progress has been made in the field of neural networks have come about in

recent years. This is due to advances in GPU technologies and computing power Lopes et al. (2017). One example of relevant work they provide is the work done by Song et al., in which a CNN was developed for a mobile phone application for facial expression recognition. This CNN used image augmentation techniques, due to the lack of public data for their network, to prevent over-fitting. This increases the amount of data available for training the network (Lopes et al., 2017). Song et al. received and accuracy of 99.2 percent using the CK+ dataset. A CNN with 15 layers was developed by Burkert et al that achieved similar result, of 99.6 percent. They point out that although this network achieved high results, they may prove to be misleading as is it not made clear that their training and test datasets were different (Lopes et al., 2017). They then go on to compare the related works by expressing the flaws of using over-lapping data and the lack of emotion expressions classified.

2.5.1 Facial Expression Recognition System

Following the introduction and related work, Lopes et al. provide a prerequisite understanding for their model. The implementation of their model has two stages: the training stage and the testing stage. The training stage consists of a few preliminary steps. Firstly, new images are made from existing ones in the dataset to increase the trainable data. This is done using a method proposed by Simard et al. called ”synthetic sample generation”, which involves rotating and skewing the existing images. For every photograph that exist, and additional 70 are made, adding noise to the data. This synthetic data is only used in the training stage and advantageous as it allows the model to handle variance in an image (Lopes et al., 2017). Secondly, to address the problem in alignment with of facial features, the notion of rotation correction is introduced that aligned the images using the eyes as the horizontal axis. Image cropping is then used to reduce the amount of background noise as it is said to decrease the accuracy and overall performance of the CNN. This is done by detecting only features that are valid of expression classification and cropping the image around them, excluding the neck, ears and background from the image. Down sampling is then applied for reducing the size of the image, making it 32 x 32 pixels in size. Brightness and contrast can cause

problems with images, therefore intensity normalisation is applied to lower these aspects of the image (Lopes et al., 2017).

As for the testing stage, the same methods are used as the training stage. The CNN outputs the predicted emotional expression with the following number ID's:

0 - Angry

1 - Disgust

2 - Fear

3 - Happy

4 - Sad

5 - Surprise

Their model is comprised of 2 convolution layers, 2 sub-sampling layers and 1 fully-connected layer. The sub-sampling layers use max pooling with a 2x2 kernel. This halves the size of the image. The fully-connected layer consists of 256 neurons and provides 6 prediction outputs. Stochastic gradient descent is applied for back propagation and the Loss is calculated using soft-max. They also use the rectified linear unit (ReLU) for their activation function.

2.5.2 Experiments

Lopes et al.'s experiments include a number of different test cases. These test cases utilize the the CK+ database that has images of a hundred students ages from 18 to 30. An average accuracy is given for each of the test cases. See table 2.1 on page 11 for results.

Test Case Averages	
Test Case	**Average**
no pre-processing	53.57%
just cropping	71.67%
just rotation correction	61.55%
cropping and rotation	87.86%
intensity normalisation	57%
both normalisations	86.67%
spatial normalisation and synthetic data	87.1%
both normalisations and synthetic data	89.76%

Table 2.1: Test cases by Lopes et al.

In summary, it can be seen from the table above that by applying the synthetic and normalized data to the training dataset, a higher accuracy can be achieved when testing.

2.6 Relevant Work and Critical Analysis

Subhashini and Niveditha, 2015's work on analysing and detecting employee's emotions for organizations ties in very well with the proposed project as it involves using sentiment analysis on subject to gain an underlining understanding of their emotions that may or may not be expressed verbally. The main research goal was to achieve employee identification in conjunction with facial emotional analysis and they were successful in execution. They used a Bezier Curve on the subject's lips and eye to detect the emotion expressed by analysing the gradient of the curve. However, although this paper proved that the project was a success, there are some inconsistencies. For example: in the related work, they do not explain how the work is related and only give titles. Secondly, the results show no code snippets or pseudocode to explain the implementation of the system. Only screen shots of the user interface are provided. Furthermore, there are more absences of proof. The paper is lacking statistics and graphs to display the accuracy metrics or progress of the systems performance. There are some bold statements used that are not backed up by citations like when it is said that "emotions were considered a forbidden topic in the working place". Despite these weak areas in the paper, a good aspect of this system is the use of real life subjects used in testing.

The review of Deep Learning for Video Classification and Captioning by Wu et al. provides an in depth look into the aspects of different neural networks and what are they strong and weak points. The motivation for their research is driven by their claim that video communications is growing and that their needs to be better applications for video understanding. The relevancy of this paper provides the concept of the ”Two-stream” architecture. Although it is not planned to develop two convolutional neural networks (CNN), iit may be sought after to develop a score fusion algorithm, similar to the one mentioned in this paper. This paper is very in depth and draws good comparisons between the different techniques that can be used for video classification. Despite the quality of this paper, some aspects need improvement. There is heavy usage of words like ”we” used. Also, some statements are made by (Wu et al., 2016) that are not cited to support their claim. This is evident when its said ”As deep learning for video analysis is an emerging and vibrant field...” (According to whom?).

Subject independent facial expression recognition with robust face detection using a convolutional neural network by Matsugu et al. illustrates the difficulties that may arise when performing facial recognition. They highlight the problems that may occur in terms of being able to handle variability of subject faces, and certain angles. Their approach to this problem is addressed by implementing a rule based algorithm that analysis the results given from the CNN. Furthermore, their model is designed to be segment and be trained on specific facial features instead the face as a whole. Their model proves to be a success as they score an accuracy of 97.6 percent, also they do not require a second CNN working concurrently to achieve similar results as other models have done so previous, which can be cost effective. Some similarities arise between this paper and the proposed project, they both use sentiment analysis and require the ability to handle a wide range of variability. This proves beneficial to the proposed project as it provides inspiration to use a rule based algorithm for determining emotions. Even though this paper is well written, there are some issues. In certain parts there are abbreviations to words given without the full word be given prior which can cause confusion to the reader. For example: ”FP neurons”. In addition to this, their model is specific for smiling faces and doesn’t accommodate for other emotions, which should be at least

provided in a further work section.

Arthmann and Li's paper titled Neuromarketing The Art and Science of Marketing and Neurosciences Enabled by IoT Technologies is a promising insight to the field of neuromarketing. They recognize the association of online shopping and loss of sales for retail stores and give example of how neuro-linguistic programming and IoT technologies can be used as a combination to understand their customers un-explicitly expressed emotions. Furthermore, it's heavily argued that the integration of AI and machine learning will evolve this concept to understand the thoughts of consumers and further tackle loss in productivity. This proves relevant to the proposed project as it is a very similar use case. They both aim to gain a deeper understanding of human sentiment that may not be express verbally. Although this is possibly the most interesting paper of this review, it is severally lacking citations. Also, there are a lot of assumption brought forward with no clear indication as to how this knowledge is known. Additionally, it is also assumed that people will adopt these "always on" IoT devices and agree for their physical aspects to be used for consumer targeted marketing.

Lopes et al. go into great detail of the past implementations of facial recognition systems. They review a number of methodologies to reinforce the idea of data augmentation with their own work. It is stated that due to the lack of datasets that are publicly available and overlapping of data, much of the accuracy results that are given by past works may be inaccurate (Lopes et al., 2017). Furthermore, they enable the reader to grasp a deep understanding to not only how convolutional neural networks operate, but how the common methods of implementation should be done. This is evident when explaining their system's implementation which was broken into two segments: The training phase and testing phase. Additionally, they give a clear and comprehensible description of the methods chosen and why they were taken. For example: using synthetic sample generation, rotation, cropping, down sampling, and normalization. This work proves relevant to the proposed project as it gives an explanation of how a segment of the project should be made. Also, Lopes et al. make it very clear that the CK+ dataset is the most suited dataset for facial and sentimental expression recognition.

2.7 Concluding Remarks of Literature Review

In conclusion, the five papers reviewed topics in relation to convolutional neural networks, facial sentiment analysis, emotion detection and current applications in the real world. Relevant aspects include detecting employees emotions in a work environment, implementing a score fusion algorithm for achieve a summation of two sentiment detecting technologies, the idea of segmenting facial features while training to accommodate for variance and the notion using human emotion understanding for a business solution. Lastly, an indication to what data that should be utilized is obtained throughout the range of papers that have been reviewed.

Chapter 3

Analysis

The following chapter will provide the business background of the project and further justification for such a system to be implemented, an analytic insight to the steps required for designing the proposed software in relation to a business use case, identify some of the problems that may arise and outline objectives for implementation.

Project Background

An article by Hague and Hague for the B2B international states that understanding a customer's satisfaction rate is important as it can show where is the business is doing well or where it needs improvements. It also can give indication to business owners where a further staff training is needed, or how there may be a need for cultural change. In light of this, seeing your business through the eyes of the customer can provide understanding of its downfalls. This in turn prevents more customer churn and can prove to be financially beneficial (Hague and Hague, 2017). They state that the downfalls of customer service surveys is the that the survey must "ask the right question of the right person". Hague and Hague give two reasons why this is difficult: they may not know what to ask the customer in respect to their specific interaction with the business, also they may not have the contact details of the customer to further inquire into satisfaction of the individual. A main reason why customer surveys result may be misleading is the fact that an average of only 10% percent of

online surveys sent to customers are responded to, meaning that these results are not representative of an entire customer base (Willott, 2011) . Furthermore, surveys are described by Hague and Hague as a ”snapshot at one point in time”, and do not accurately represent the feelings of the customer during transaction process, and measuring the satisfaction of customers should ”be a continuous process”. Kirkpatrick (2017) describes the use of artificial intelligence technologies in customer service as a beneficial factor for businesses, as it can automate the tasks that are ”too time consuming”.

In consideration of the aforementioned problems in the customer service sector, the proposed technologies aim to eliminate the process of requesting customers to fill out surveys by performing a facial sentiment analysis during the transaction, and in turn reduce customer churn and the cost of marketing teams investing funds onto campaigns that me be driven by misleading or inaccurate data. In order to achieve the end goal of this project, a number of factors should be addressed. These consist of the type of model that should be implemented, what machine learning library should be used, which programming language is preferable for this project, how may the model be trained and what are the options for deployment. For these problems to be address, the project analysis shall be decomposed in to four components: The Machine Learning model, Data preparation, Training and Hosting/Deployment

3.1 The Machine Learning Model

3.1.1 Algorithm

When dealing with image a classification problem such as facial sentiment analysis, there are two machine learning algorithms that deem worthy. Deep Belief Networks (DBN’s) were developed as an alternative to back propagation by Geoff Hinton. The idea behind DBN’s involves the stacking of a Restricted Boltzmann Machines in order to train in a ”greedy manner” (Hinton, 2017). This model learns to excerpt hierarchical representations of training data. The second algorithm is convolutional neural networks (CNN’s). To understand the structure of convolutional neural networks, firstly it should be briefly explained how an

artificial neural network works.

Artificial Neural Networks and Back Propagation

Artificial Neural networks are based on the biological brain and possess a number of connected nodes. An artificial neural network can be seen as an hierarchical ordering of mathematical constructs called neurons, which are nodes that perform a mathematical function to simulated the biology of the brain on a molecular level (Muir, 2016). A neural network in one of it's most simplest of forms has 3 levels of structure, which are otherwise known as layers. The first layer consists of the neurons that takes in the data. This is called the input layer. This layer can consist of one or more nodes, depending on the number of inputs. The second component in the network is called the hidden layer. Unlike the input layer, hidden layers are considered as active as they can modify data as it passed through them (Smith, 2011). The input layer and the hidden layer are connected through what are known as weights. These weights are values that are randomly initialized. When passing data to the next layer of nodes, a set of matrix operations are done by multiplying the input data by the weight value. A bias is then added to the result of this. An activation function is then introduced to apply nonlinearity. Nonlinearities are used as they can provide methods of solving complex problems (Raschka, 2016). There are several activations that can be used such as Sigmoid, Tanh and Rectified Linear Unit (ReLU) to name a few. However, it has been noted that ReLU provides the best results as prevents issues such as the vanishing gradient problem. The activation of a neuron in a network can be described as the following (Collis, 2017):

$$output = activationfunction((input * weights) + bias) \tag{3.1.1}$$

These steps are repeated for each neuron in the network until a predictive output is produced. This process, in full, is called forward propagation. From this output we can evaluate it to the true expected output to optimize the network. This is part of a process known as

back propagation. Back propagation is expressed as the partial derivative of the cost/loss function, in respect to the any weight or bias that can be found within the network (Nielsen, 2015). In simplistic terms it can be defined as a function for adjusting the weights in the network to minimizing the loss functions values. This is done through what is known as the delta rule. The formula for adjusting these new weights are as follows:

$$New\ weight\ value = weight\ + learning\ rate\ x\ error\ x\ input\ value \tag{3.1.2}$$

There are a wide range of optimization algorithms that can be used for this, the most popular of which is known as gradient descent (Walia, 2017) where we calculate the gradient of the error with respect to the weights. When visualized, the gradient of the error will be equal to zero when the model finds the local minima and the network converges. An example of this can be seen in Figure 3.1.

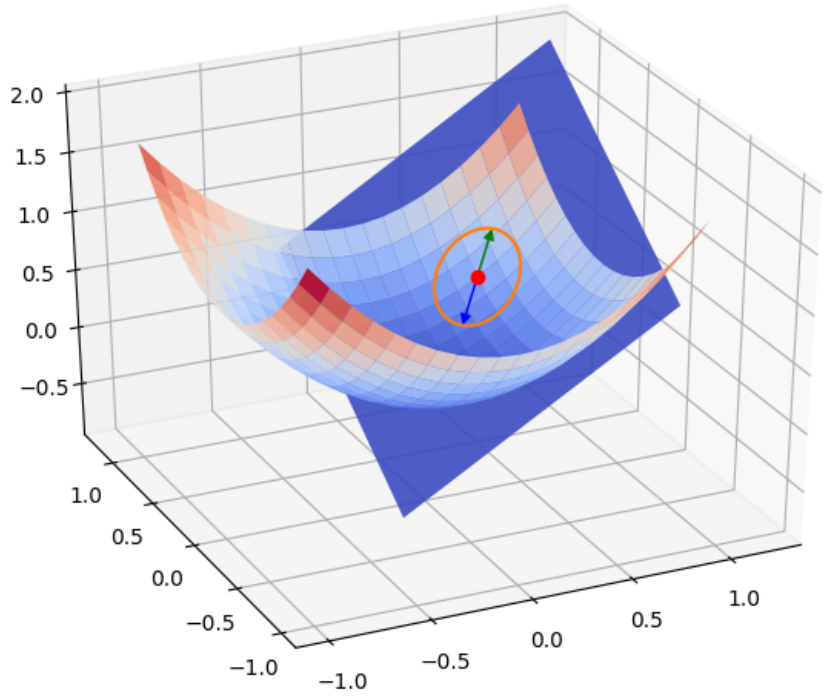

Figure 3.1: Gradient Descent Visualized by Vigier (2017)

Convolutional Neural Networks

A convolutional neural network (CNN) is a form of neural network used primarily for image classification and object recognition. CNN's, like regular artificial neural networks, take biological inspiration as they are based on visual cortex (Deshpande, 2016). CNN's recognize images as three-dimensional objects (or two dimensional if the image is grayscaled). CNN's are made up of a number of components, which include the convolution layers, pooling layers and fully connected layers. Convolution layers are the first in the sequence of components. As explained by Deshpande, the convolution layer should be thought of as a flash light being shined on a certain region of a picture from the top left to bottom right. This analogical flash is referred to as the feature filter or the kernel. This kernel convolves (slides) over the image and multiplies the images pixel values by the values in the filter, in a process which is known as element wise multiplication (Deshpande, 2016). After this image is fully convolved, this results in what is known as a feature map. The second step is referred to as pooling, which is used for down sampling. This is necessary when dealing with highly spatial images age it reduces the data needed to be processed by the network (Kar, 2015). The most popular approach for pooling is max-pooling, where the maximum pixel value found in the kernel (filter) is used as input for the next layer. The last component in the architecture is the fully connected layers. These layers take the output from the layer convolution or activation layer preceding it. Each node in the network is connected to every other node following it, hence the name ”fully connected”. From here the network act the same way as a regular multilayer perceptron ANN.

3.1.2 Programming Language

There are many programming languages that can used for machine learning and artificial intelligence. Brownlee (2016) describes the quest for choosing a programming language to be rather difficult, as the choice should be tailored around ”your own requirements” and entirely depends on the project being undertaken. However, opinions on popular machine

learning languages are given. Firstly, MATLAB/Octave is described as being ”excellent” for dealing with matrices and linear algebra (Brownlee, 2016a). The R language is shown to be a prominent choice for machine learning as it provides many machine learning algorithms and it is a great choice for developing advanced models. Brownlee (2016) states, although it is a good choice, it is seen as having a hard learning curve at first use. Python is said to be competition to MATLAB and R as it's large range of libraries and abilities as widely used in the field of data science (Brownlee, 2016a). An article by Verma (2017) sheds light on most popular programming languages in terms job listings for the year of 2016. Coming in first place was Python, as it's popularity is unmatched by any other programming language. Following Python is Java, then R is seen to be the third most popular (Verma, 2017). See Figure 3.2 for graphical representation Verma's findings.

In light of the proclamations made above, it is clear that the preferred languages should be easy to learn, are backed by a large community, and is capable of complex mathematical operations. Also, it is desirable that the languages chosen for the proposed language is fast and efficient because of the complex computations that will occur during training.

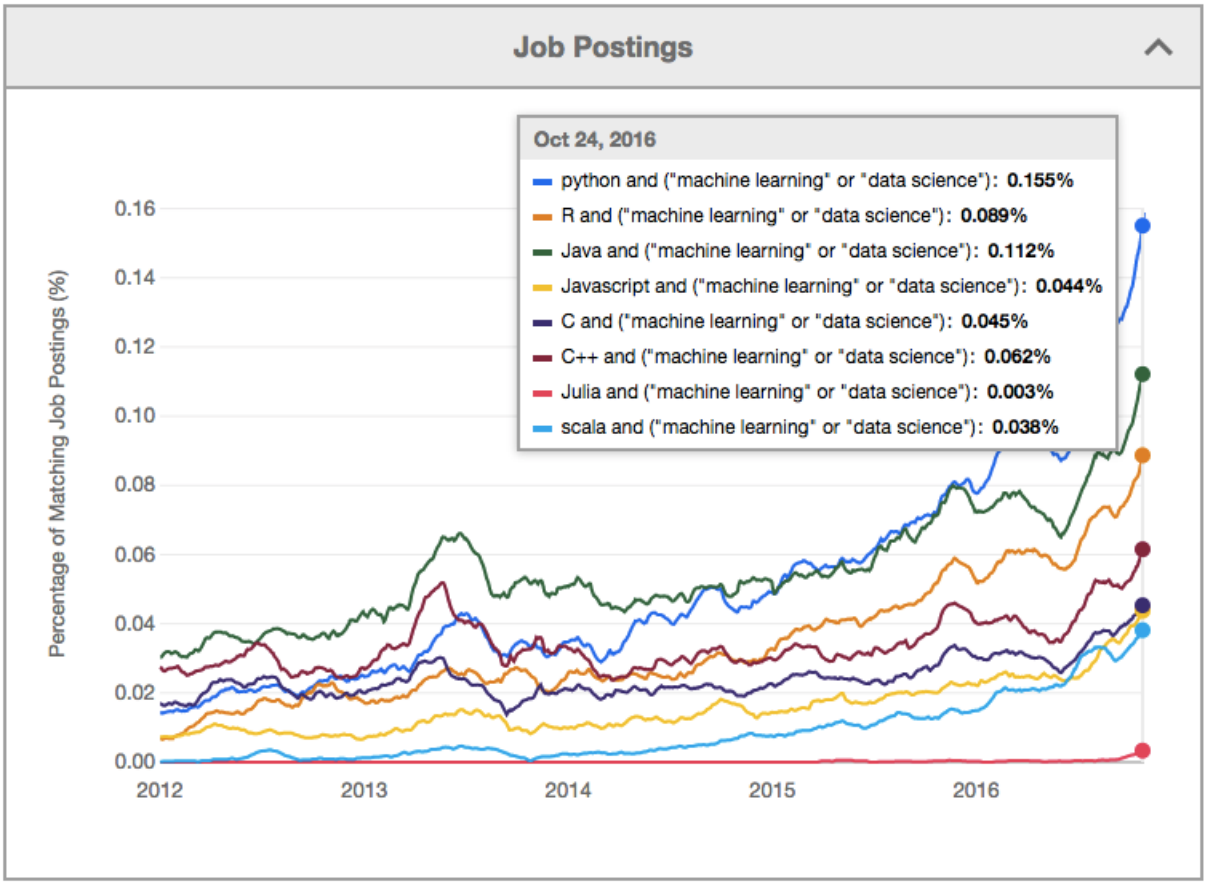

Figure 3.2: Graph of Machine Learning or Data Science Languages for 2016 by Verma (2017)

3.1.3 Machine Learning Library

To speed up the process of implementation, a machine learning (ML) should be used. This will speed up the process of the project as many ML libraries have API's to help eliminate tedious tasks. Also, these can be used to reduce the complexity and the length of the program (Jain, 2017). TensorFlow is an open source ML library developed by the Google Brain team. It can be run on CPUs, GPUs and mobile platforms. For machine learning algorithms, such as gradient descent, TensorFlow provides automatic differentiation which can

prove to be very advantageous in comparison to other ML libraries. It also supports multiple languages such as Python, Java and C++ and Go (Jain, 2017). Graph visualisation is also supported. Multi-threading is also achievable with the use of TensorFlow (Jain, 2017). Higher level wrapper libraries such as Keras can use back-end API's such as TensorFlow provide easier implementation with minimalistic coding but also giving the same efficiency and accuracy (Brownlee, 2016b). Scikit Learn is an open source ML library built on top of other libraries such as Matplotlib, SciPy and NumPy. A good feature that Scikit Learn incorporates is the ability of evaluating, chaining and adjusting model hyper parameters (Jain, 2017). Other ML libraries such as Caffe focuses on speed and modularity and is mainly utilized for convolutional neural networks and computer-vision. Another selling point for Caffe is it's pre-trained models that do not require any coding or training and it supports GPU and CPU computations. A disadvantage of this library is that it is specifically designed for application implementation, not for research and development (Jain, 2017).

3.2 Dataset

For this project, the machine learning model will require a large image dataset. This dataset will be used to train the model on certain facial feature expressions with supervised learning, therefore a labeled dataset is needed. Acquiring a good dataset is one of the most important parts of the project, as the model only as good as the data you train it with (Kaasschieter, 2017).

3.2.1 Public Datasets

As seen in the literature review, there are very few datasets publicly available for use on the Internet that provide labeled data. For example, the CK+ dataset was used by Lopes et al. used labeled data to train their model. The dataset included facial expressions such as sadness, surprise, happiness, anger, fear and disgust. Which is a considerable approach for training the model.

3.2.2 Creating a Dataset

Creating a dataset is another option for training the model. A number of images could be batch downloaded from image search engines such as Bing or Google and split into training and testing sets. As for labeling, the name of the folder the type of picture is in will serve as the label.

3.3 Training

Training is a vital part of the project and will be the most time consuming. The model will use the chosen data preparation method and use that data as input to the model. There are two ways the model can be trained: Locally and on an Infrastructure as a Service (IaaS).

3.3.1 Training Locally

Training locally involves running code on your own machine to train the model. A main advantage to this is that it is low cost. There are no fees in this approach besides the electricity used by the machine. However, this approach may be harmful to the machine as the high rate of computations produces a lot of heat, which may prove to be detrimental to your hardware. Also, should the machine not have the sufficient hardware, such as a high-performing graphical processing unit (GPU), the training elapse time will be significantly larger.

3.3.2 Cloud Training

Training in the cloud involves uploading the model code and dataset to a cloud platform service. These infrastructures provide dedicated hardware for machine learning training such as high-performance GPU's and CPU's. They usually provide container environments to run the training. For example, should a model be implemented in Keras or Caffe, the platforms have dedicated runtime environments with these libraries pre-installed. Notable platforms for machine learning training are TensorPort, FloydHub and Amazon's AWS. Although it is

faster to train a model on one of these platforms, they can be expensive, as the free tiers only provide a certain amount of free server usage time.

3.4 Hosting/Deployment

Deployment of the trained model is required to serve it as a web service endpoint. This model should be hosted as an API to enable use from devices besides the machine it was trained on. The API will require a platform that containerizes and supports the machine learning libraries it was used to train with. Additionally, the API should be lightweight and fast for accurate classifications. There are many Platforms as a Service (PaaS) that can be used for this project, notably Heroku, AWS, Azure and Google App Engine all provide a free tier basis for deployment.

3.5 Concluding Objective

In conclusion, this section outlines the objectives for the proposed project sequentially from the initial steps to the end product. The objectives are as follows:

- Acquire a suitable dataset of facial expression images.
- Build a convolutional neural network for facial expression classification using an ML library.
- Train the model on a large image data set.
- Save the trained model and deploy it as an API.
- Develop a web application that is enabled to record a user's facials expressions
- Use image pre-processing on the web cam images for more accurate predictions.
- Send snapshots of the users face to the Python API in intervals, classify them and display the results to the user.

Chapter 4

System Design

The following chapter will briefly explain the chosen design process for the project. The components of the project are broken into four parts: Data Preparation for insight into the data utilized and how it will be prepared, A section on what algorithm and machine learning library will be used, an explanation on the approach for training the model and a deployment section for the process in which the model was deployed for production. Lastly, the overall system architecture and flow of objectives will be explained.

4.1 Data Preparation

As stated by Lopes, de Aguiar, Souza, and Oliveira-Santos that there is a scarcity of public datasets with for facial expression images, therefore the Cohn-Kanade+ (CK+) dataset shall be used as it provides a wide range of emotions to train the model on. However, in nature of use for this project, the emotions of disgust and contempt will be omitted from the project as it does not fit the nature of the use case, and as they bear too much resemblance to the anger images. Instead, a neutral emotion class will be implemented. The dataset will be preprocessed to reduce dimensionality, and synthetic samples will be generated via data augmentation to increase the size of the dataset and prevent class imbalance. The dataset will then be divided into training and testing sets using split validation.

4.2 Machine Learning Model

For designing the model, the TensorFlow Machine Learning library by Google will be used to design a convolutional neural network. The network consists of several convolution layers, pooling layers and a fully-connected layer. The model has six outputs nodes that provide probabilities of classification for each image fed into the dataset. The model will use supervised learning, with labels for the images it was training on, which will be brought forward in the following section.

4.3 Training

Due to insufficient hardware, the approach of cloud training is needed in order to train the model. The Python code and the dataset are pushed to a containerized TensorFlow environment on the FloydHub cloud platform. The output of the trained model is saved to a Tensorflow *.ckpt* file. This trained model will then be downloaded for further evaluation.

4.4 Deployment

In order to deploy the trained model to production, a Python API will be developed using the Flask package, which is a library for running python code on a server. The API will be hosted on Heroku PaaS as they provide an easy-to-use and free basic tier service for hosting Python applications. The trained model files are also deployed with the API. The model is read and initialized from the respective files when the API successfully builds and deploys.

Secondly, a Node.JS web app shall be developed in order to display how this model can be used in production. The application was also deployed to Heroku. The application uses the clients web cam in order to record their face. This takes snapshots of the users face and sends the images to the Python API in one second intervals. The application then receives

the response from the API and displays it to the screen. See Figure 3 for a general design of the user interface for the node application.

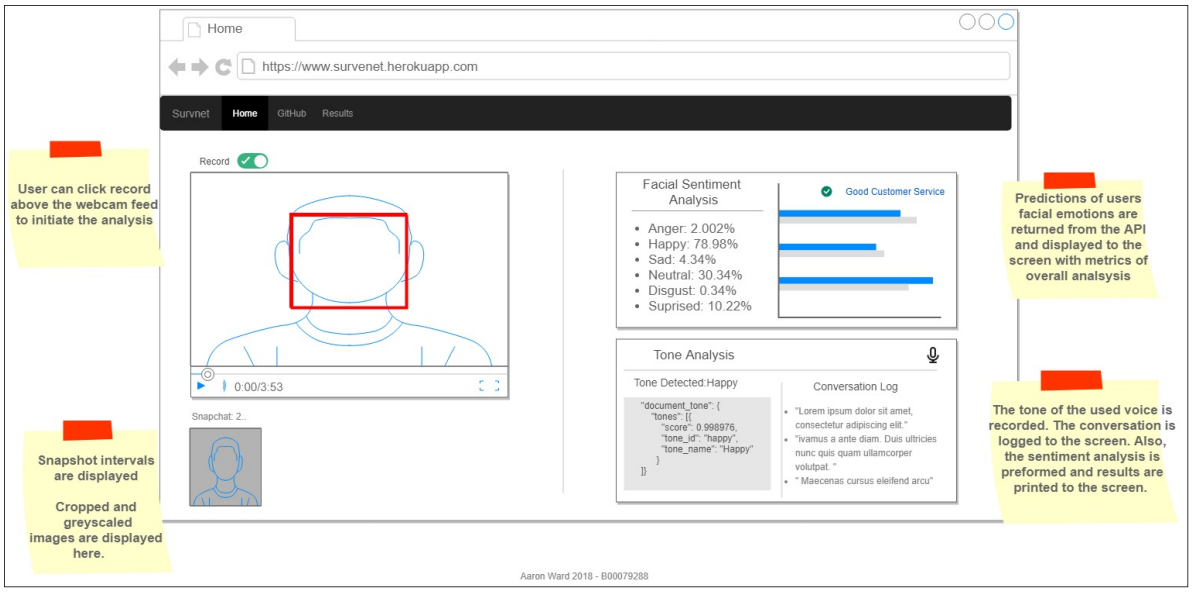

Figure 4.1: User Interface Design For Web Application

4.5 Concluding Remarks of System Architecture Design

In summary, the system architecture will be considered in two parts: Training and deployment. Firstly, the data preparation and neural network scripts shall be written on a local machine. These scripts and dataset will be uploaded to a TensorFlow environment for training on the FlyodHub cloud service. Upon completion of training, the model will be saved and pushed to the *model-serve* GitHub repository that is integrated with the Heroku PaaS. This repository consists of a Python Flask API to accept POST requests. The Node.js web application shall also be deployed to the Heroku for easy user access. This shall send requests to the model API for classification of images of the user. Please refer to Figure 4.2 to see an illustrated topology of the system.

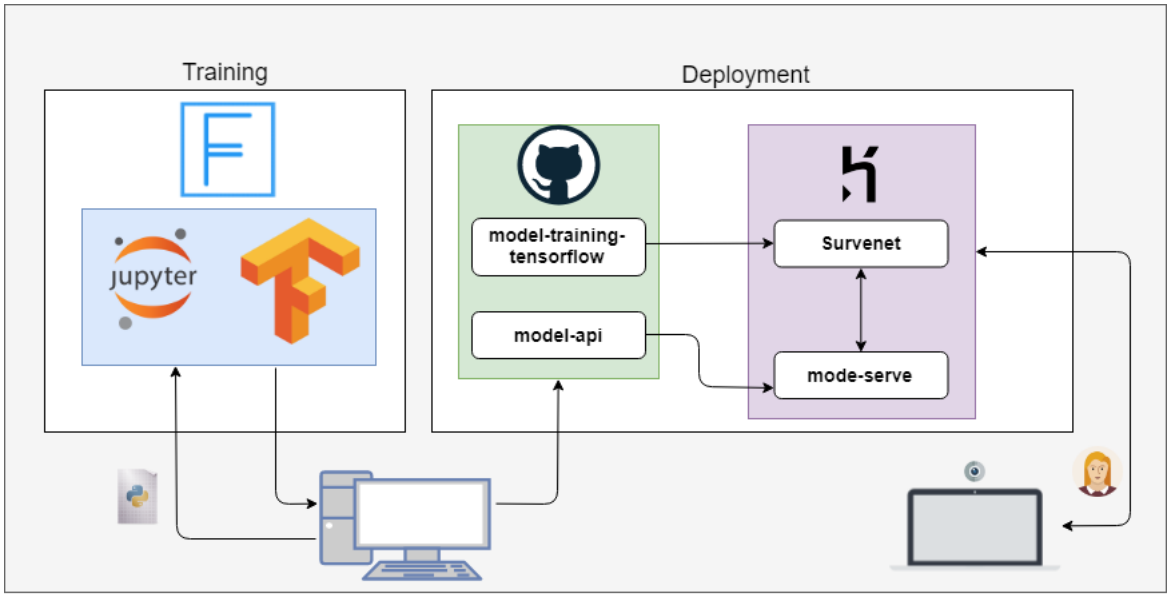

Figure 4.2: System Architecture of Training and Deployment

Chapter 5

Implementation

The following chapter will outline the implementation process of the project and how each component was developed. Firstly, an overview of the dataset used shall be given with information on how the images were preprocessed and augmented for training. A section for the model implementation will be given to describe how the convolutional neural network (CNN) was developed using TensorFlow. This is followed by the deployment process of the TensorFlow Python API. Lastly, the steps in which the Node.JS web application was made shall be touched upon, accompanied by the design and features choices.

5.1 Data Understanding

As stated in the system design, the dataset that will be used is the Cohn-Kanade+ image dataset, which was release in 2000 with the purpose of classifying facial expressions (Lucey et al., 2010). The dataset consists of images of 210 subject between the ages of 18 and 50. Each subject is asked to show a facial expression, beginning with a neutral facial expression that eventually lapses to the target emotion. The digital images come in either a 640x490 or 640x480 pixel format. Furthermore, the images either consists of an 8-bit grayscale or 24-bit colour format (Lucey et al., 2010). The dataset is made up of 593 image sequences and provides labels for each sequence, ranging between 0 and 6. Refer to Figure 5.1 for a summarized illustration of how these sequences are built.

Figure 5.1: Image Sequence From Neutral to Happy

It should be noted that the figure above is not a full image sequence and some transitional images have been omitted. Moreover, it should also be known that there are inconsistencies in the dataset, such as a number subject not having image sequences for each certain emotions and label data missing for some images. This is acknowledged in the *README* file created by Lucey et al. that is contained within the dataset, where it is declared "*IF THERE IS NO FILE IT MEANS THAT THERE IS NO EMOTION LABEL (sorry to be explicit but this will avoid confusion)*".

5.2 Data Preparation and Preprocessing

A number of preprocessing steps were taken in order to prepare the images for training the network. These steps were required to ensure maximum possible performance and minimal training time as the nature of working with neural networks can be computationally expensive when accompanied with large files such as images. The steps are as follows: Image extraction, Grayscaling, Facial Cropping, Data Augmentation and Data Splitting.

5.2.1 Image Extraction

Due to the structure of the image sequence within the datasets, it is undesirable to used all images within each sequence as that do accurately resemble the facial expression it is

used to represent. Therefore, for each sequence the last four images were extracted from their directory and relocated to a new directory under the category of facial expression they represent. As stated in the system design, images for disgust and contempt are omitted from this new dataset. Furthermore, because there is now class for a neutral facial expression, the first four images from a sequence in each subject was extracted. The structure of the image classes for this project are as follows:

- 0 - Angry
- 1 - Fear
- 2 - Happy
- 3 - Neutral
- 4 - Sadness
- 0 - Surprise

5.2.2 Grayscaling

The CK+ dataset consist of a majority of gray images. However, the extended version of this dataset contains some images sequences with coloured images. When working neural networks, it is significantly more computationally expensive to process a coloured image over a gray one due to the number of colour channels. To mitigate this, a Python script was created to traverse through each image of the newly extracted dataset to convert all images to grayscale using the PILLOW library.

```
from PIL import Image
import numpy as np
import os, os.path
img_path = '<DATASET_DIRECTORY>'
```

```
def grayify(file_name):
        image = Image.open(img_path + file_name)
        image = image.convert('L')
        image.save(img_path + file_name)

#Load the directory and traverse over all the image files
list = os.listdir(img_path)
for file in list:
        file_n = file
        print(file_n)
```

5.2.3 Facial Cropping

Following the grayscaling of all images, dimensionality reduction was implemented on the images. This was done by cropping each image down to only the facial surface area to reduce the noise of the data and to remove any features in the backgrounds that may be learned by the network that do not represent the facial expression. To do this, a Python script was written that reads in all the images from the dataset and crop the region of the image that contains the subject faces. This was done using the OpenCV library.

```
def facecrop(image):
        face_cascade = cv2.CascadeClassifier
        ('haarcascade_frontalface_default.xml')

        img = cv2.imread(image)
        minisize = (img.shape[1],img.shape[0])
        miniframe = cv2.resize(img, minisize)
        faces = face_cascade.detectMultiScale(miniframe)

        for f in faces:
```

```
                x, y, w, h = [ v for v in f ]
                cv2.rectangle(img, (x,y), (x+w,y+h),
                (255,255,255))
                sub_face = img[y:y+h, x:x+w]
                fname, ext = os.path.splitext(image)
                cv2.imwrite(fname+"_cropped_"+ext, sub_face)
        return

list = os.listdir(<DIRECTORY_NAME>)
for file in list:
        facecrop(<DIRECTORY_NAME> + '/' + file)
```

This was done for all images, however, some images were not correctly cropped due to noise in the image causing it to misclassify the facial region. Some examples of this might be only half the face being cropped into the new image or just the subjects shoulder being visible. These worthless images were manually deleted after the newly cropped images were evaluated. Please refer to Figure 5.2 for example of the facial cropping.

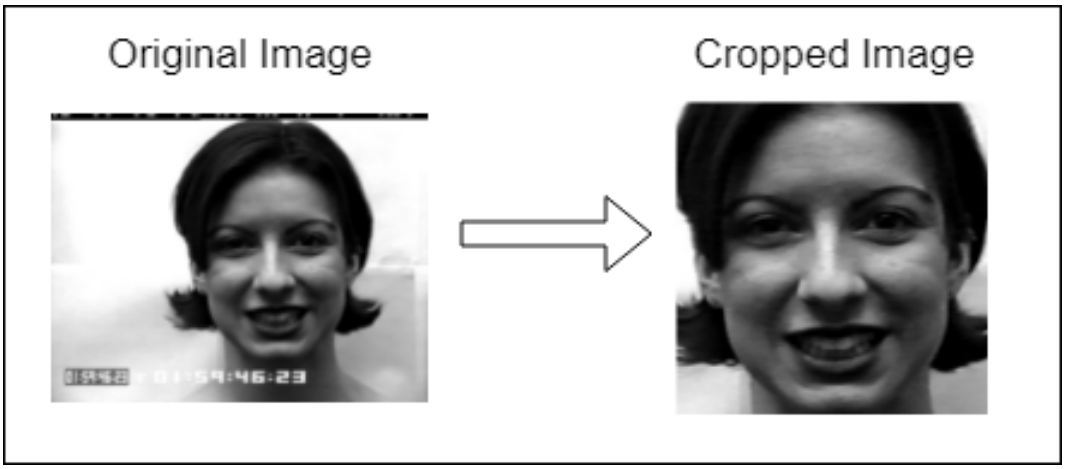

Figure 5.2: Image Cropping on Facial Regions

5.2.4 Image Increasing Through Data Augmentation

Upon inspection of the dataset, in regards to the number of images for certain facial expressions, it was noted that there was an insufficient amount of images to train the network. This can be seen in table 5.1.

Facial Expression	**Cardinality**
Anger	601
Fear	427
Happy	883
Neutral	668
Sad	641
Surprise	639
Total	**3879**

Table 5.1: Initial Image Count

The initial step to increase the size of the data set was to flipped versions of all the images. Not only does this double the size of the dataset but helps the network to better handle facial variance when dealing with new unseen data (Lopes et al., 2017). Also, it will decrease the chance of under-fitting while training the model. Following this step, a check was done on the class balance. Using the Matplotlib Python library, the balance of each class was plotted by counting each image in accordance to its respective label, as seen in Figure 5.3 which shows the cardinality for each training sample.

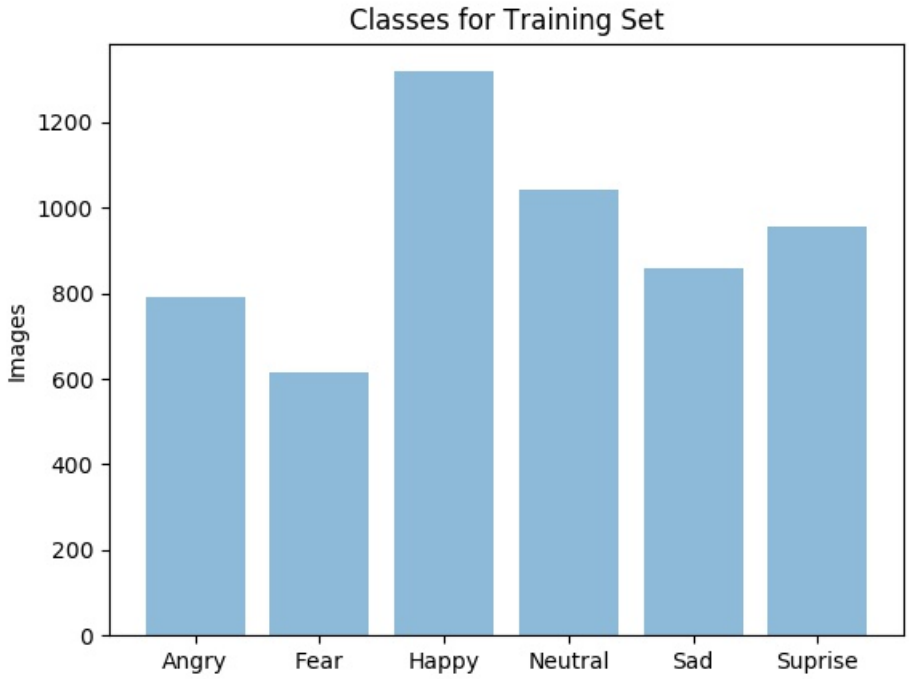

Figure 5.3: Class Imbalance

As seen above, expression 'Anger' and 'Fear' are underrepresented while 'Happy' is overly represented, relative to the rest of the dataset. How this problem was addressed was by creating synthetic sample images from the existing ones by the means of skewing and augmenting the images. This can be seen in Figure 5.4, where the top row displays the original cropped images, and beneath, are the slightly skewing images. This was implemented using a Python library called Augmentor, which allows you to specify the directory and number of altered samples you would like, and it randomly picks images within the directory, creating an entirely new back of images that have been stretched to a degree.

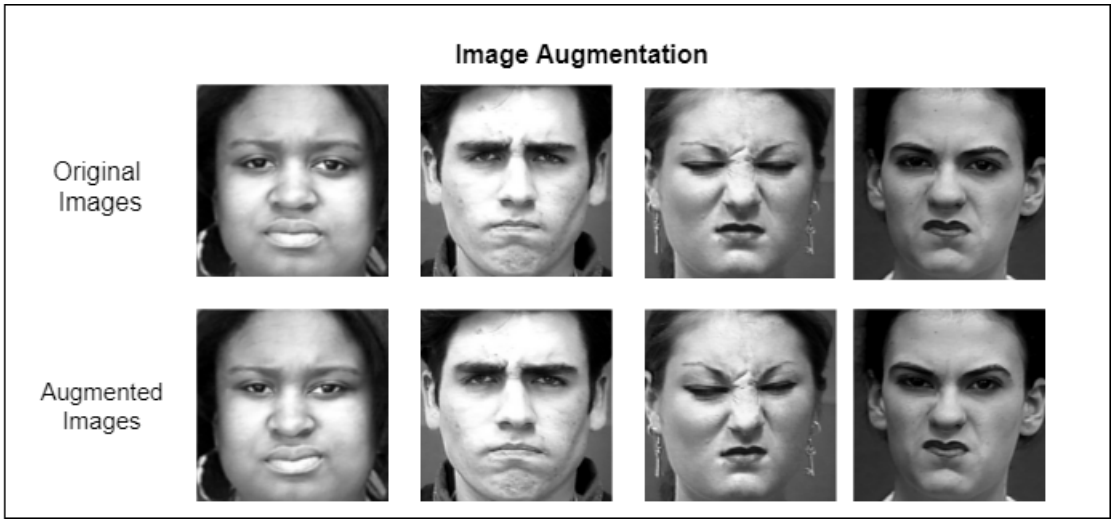

Figure 5.4: Example of Augmented Images

This was done for all classes to increase the number of samples in the dataset, until a number of 1750 images are present for each class. This amounts to a total number of 10500 images in total across the entire dataset. The class balance can be seen in the bar chart illustration in Figure 5.5, which has been plotted using the Matplotlib library. In conclusion, an additional 6,621 images have been created from the original dataset.

Figure 5.5: Class Balance

5.2.5 Data Splitting

The last step in the preparation phase was to apply split validation to the dataset. The dataset was split into training and testing sets at a divide of 80/20 - meaning 80% of the dataset will be used for training (1400 images per class) and 20% will be used for testing the model (350 images per class). Please see the Python code below showing how this was achieved.

```
import os, os.path
import math
from PIL import Image

old_path = 'C:/Users/aaron/Desktop/Cropped_Dataset/happy/'

new_train_path ='C:/Users/aaron/Desktop/data/training/happy/'
```

```
new_test_path = 'C:/Users/aaron/Desktop/data/testing/happy/'

#Load the directory and traverse over all the image files
list = os.listdir(old_path)

# define the size of the first 80 percent of images
num_files = len(list)
num_training_files = num_files * .8
num_training_files = math.ceil(num_training_files)

num_testing_file = num_files * .2
num_testing_file = math.ceil(num_testing_file)

#Add the first 80 percent to the training folder
for img in list[1:num_training_files]:
        i = Image.open(old_path + img)
        i.save(new_train_path+img)



#Add the remaining 20 percent to the training folder
for img in list[num_training_files:]:
        i = Image.open(old_path + img)
        i.save(new_test_path+img)
```

5.3 Implementing the Machine Learning Model

When implementing the machine learning model with TensorFlow, the initial step used is to import all the libraries that deemed necessary for preprocessing and training. These include TensorFlow, Numpy, Skimage, PIL etc. Secondly, the image data was imported. A list is used to store the image RGB data for every image in the dataset. Secondly, as stated in the

data understanding, there is some inconsistency in the dataset such as some images not being labeled. Therefore, the class folder in which a certain image is imported from was used as the label, and added to the labels list. Refer code below to see how this was implemented.

```
def load_data(TRAINING_DIR):
images = []
labels = []

directories = [d for d in os.listdir(TRAINING_DIR)
if os.path.isdir(os.path.join(TRAINING_DIR, d))]

# Traverse through each directory and make a list
# of files names if they end in the PNG format
for d in directories:
        label_directory = os.path.join(TRAINING_DIR, d)
        file_names = [os.path.join(label_directory, f)
        for f in os.listdir(label_directory)
                                if f.endswith(".png")]

        #Traverse through each file, add the image data
        # and label to the 2 lists
        for f in file_names:
                images.append(skimage.data.imread(f))
                labels.append(int(d))
        return images, labels

images, labels = load_data(TRAINING_DIR)
```

Following this, shuffling of the data is performed, this is done to prevent overfitting of the model. Then each image in the list of images are sub sampled from their original sizes down to a 50x50 pixel size. This was done with the Skimage library to reduce the dimensionality of the images for the network to better handle the input data. The list images were traversed

and resized using the transform function.

```
images = [transform.resize(image,(50, 50))
        for image in images]
```

After the basic data imports and preprocessing, defining the computation graph for TensorFlow is required. TensorFlow placeholder were then declared for later use when tensors need to be fed to the TensorFlow session, along with another few initial variables such as number of epochs, the batch size, dropout rate etc. When all the main variables needed for TensorFlow are made, the structure of the network was mad. This was done using the tf.slim library, which a TensorFlow package that allows you to define layers in a network easily.

In total there are 10 convolution layers with 5 Max Pooling layers. The convolution layer is used to extract features within the image while pooling reduces the dimensionality of the feature maps, but retains majority of the useful data. Please refer to Figure 5.6

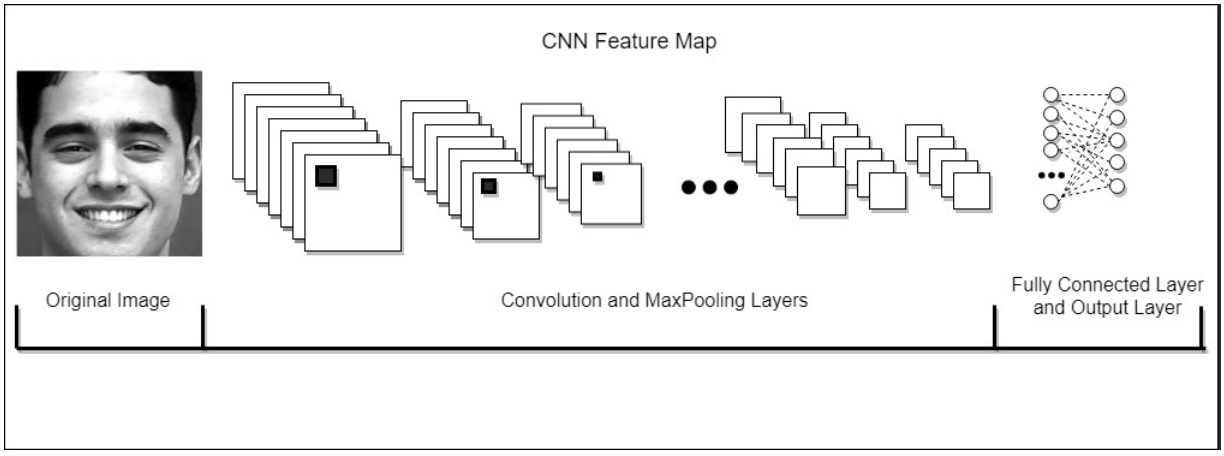

Figure 5.6: Feature Maps of CNN

Lastly, a fully connected layer is defined, which will be used to produce a predicted output. Dropout is a regularization technique that is used in this case is to reduce the chances of over fitting the network (Srivastava et al., 2014). After each training iteration (epoch), only 80 percent of the trained neurons are kept.

```
        net = slim.conv2d(net, 32, 3)
        net = slim.conv2d(net, 64, 3)
        net = slim.conv2d(net, 64, 3)
        net = slim.max_pool2d(net, 3, stride = 1 )
        net = slim.conv2d(net, 96, 3)
        net = slim.conv2d(net, 96, 3)
        net = slim.max_pool2d( net, 2, stride = 2)

        net = slim.conv2d(net, 128, 3)
        net = slim.conv2d(net, 128, 3)
        net = slim.max_pool2d( net, 2, stride = 2)

        net = slim.conv2d(net, 128, 3)
        net = slim.conv2d(net, 128, 3)
        net = slim.max_pool2d(net, 2, stride = 2)

        net = slim.conv2d(net, 128, 3)
        net = slim.max_pool2d(net, 2, stride = 1)

        net = slim.dropout(net, keep_prob = keep_rate
                , is_training = is_training )
```

Following the structure set up of the network, a function is defined that runs the TensorFlow session, which takes the **x** placeholder as a parameter. Within this function we define the loss function used within the network which is softmax with cross entropy. Also, the Adam optimizer is used for back propagating through the network and adjusting the weights. The Adam optimization algorithm is said to be a suitable optimizer to use as it is very robust when dealing with high dimensionality (Havaei et al., 2015). Which, in the case of a CNN,

is highly preferable. A learning rate of 0.002 was chosen for the final implementation after showing in experimentation that this reduced the loss the most.

```
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits
    (labels = y, logits = output))
total_losses = tf.losses.get_total_loss
        ( add_regularization_losses=True ) + loss

update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
with tf.control_dependencies(update_ops):
train_op = tf.train.AdamOptimizer( learning_rate=0.002 )
                        .minimize( total_losses )
```

When the session is ran, the TensorFlow variables are initialized and the training begins. The list of images and labels are segmented into batches of size 100 to prevent an OOM error, as loading all the images into memory can cause an overflow. The images and labels are fed into the TensorFlow runtime environment for training. After each epoch, the accumulated loss and batch accuracy is calculated and printed to the screen. After the network have been trained for the specified number of epochs, it evaluated the accuracy on the test data and a TensorFlow *.CKPT* model file is saved to disk.

```
op, ac, loss_value, total_loss_value, pred_classes,
label_classes = sess.run([train_op, acc, loss, total_losses,
 pred_class, label_class ],
feed_dict={x: train_batch_x, y: train_batch_y,
 is_training : True})
```

5.4 Deploying the Trained Model

In order to deploy the trained TensorFlow model a Python application programming interface was created using the Flask library. Flask is a microframework developed for running Python

scripts on a server, similar to Node.js and Express. In the API, routes were set up to handle HTTP POST requests. When image data in the form of Base64 is sent to the server, it is decoded from Base64 and written to disk in the form of a PNG file. This file is then preprocessed to have the same characteristics as the data the model was trained on. The image is converted to grayscale, then resized to be 50x50 pixels and lastly a new dimension is added to the image so that it can be interpreted by TensorFlow trained model. After a classification results is returned from the model, the results is converted to a JSON string and returned to the application from which the request came.

```
@app.route('/api', methods=["POST"])
@cross_origin()
def evaluate():

imageData = request.get_data()
data_preprocessing(imageData)
print('1: Image was converted')

image = Image.open('output.jpg')
image = image.convert('L')
image = image.resize((50, 50), Image.ANTIALIAS)
print('Resized and Grayscaled Image')

image = np.expand_dims(np.array(image), axis = 0)
classification = sess.run(prediction, feed_dict =
        {x: image, is_training : True})

# add highest probability result to classes
classes = np.argmax( classification , axis = 1 )

res = 'undefined'
if    classes[0] == 0:
```

```
        print('Predicted: Angry')
        res = 'Angry'
elif classes[0] == 1:
        print('Predicted: Fear')
        res = 'Fear'
elif classes[0] == 2:
        print('Predicted: Happy')
        res = 'Happy'
elif classes[0] == 3:
        print('Predicted: neutral')
        res = 'Neutral'
elif classes[0] == 4:
        print('Predicted: sad')
        res = 'Sad'
elif classes[0] == 5:
        print('Predicted: Suprised')
        res = 'Suprised'

return jsonify(res)
```

For deployment, the Heroku PaaS was used with GitHub integrations. Whenever a commit was made to the master branch the API was automatically deployed to the server, a build is made and then the API is ready for use.

5.5 Implementing the Node.JS Application

To develop a web application for the user to interact with, Node.JS was used to implement a simple one-page application that accesses the users webcam. Firstly, a Javascript web-server was created using the Express npm module, which provides functions for routing and serving html pages. A port is initially set using either port 5000 for local development, or a randomly

allocated port for a deployment environment.

```
app.set('port', (process.env.PORT || 5000))
```

This is followed by providing the home page when the base route is called by from the client. When the **'/'** route is called, a html page containing the application is delivered to the users browser and displayed in the DOM.

```
app.get('/', function (req, res) {
  res.sendFile(path.join(__dirname, '/index.html'));
});

var server = app.listen(app.get('port'), function () {
  var host = server.address().address
  console.log('Application running on port', app.get('port'))
})
```

5.5.1 Facial Tracking With a Users Webcam

The user interface design of the application is very simple. On the left-hand side of the page the webcam feed is displayed. This accesses the user's machine's webcam and displays it to the screen using the html **canvas** element. Canvas is a container for drawing graphics to the page. For facial tracking, a library called Tracking.js was imported. Tracking.js is an open source library created by Eduardo Lundgren for object, colour and face detection using Javascript. The reason for this library choice was because other alternatives, such as OpenCV, deemed to be very heavy weight in comparison and API services that have also been considered such as Cloudinary would reduce classification speed as it would be computationally expressive to make multiple requests for facial cropping while using the application. In light of this, Tracking.js provides a lightweight and quick solution for detecting faces. Using what is known as haar cascades, object classification and detection is made simplified and provides fast calculation speeds (Viola and Jones, 2001). In terms of the web

application, the canvas drawing of the webcam feed is read and interpreted by Tracking.js and the facial features are detected. A yellow square is drawn around the detected facial area.

```
var tracker = new tracking.ObjectTracker('face');
tracker.setInitialScale(4);
tracker.setStepSize(.5);
tracker.setEdgesDensity(0.1);
tracking.track('#video', tracker, { camera: true });
tracker.on('track', function(event) {
context.clearRect(0, 0, canvas.width, canvas.height);
```

One function that tracking.js does not provide is the ability to crop the detected area of the stream. To get around this, Javascript code was developed to mark the area in which the square was being drawn around the user's face. In each frame of a user's face being detected a new square is drawn to the canvas, so every time that a new square is displayed to the screen, this marks the new coordinates of the area in which is needed for cropping.

```
rect_width = rect.width; //
rect_height = rect.height;
rectX = rect.x;
rectY = rect.y;


 ...
ctx.drawImage(video, rectX+ 100, rectY +90,
         rect_width +70,  rect_height +50, 0, 0, 300, 150);
```

After it is known where the users face is positioned in the shot, the cropped area is displayed in a separate 100x100 pixel canvas area, named ctx, below the feed. To ensure that the trained TensorFlow API does not become overflowed with image data, an interval timing is implemented. This involves setting a timer as to when an image is cropped because it would be disadvantageous and computationally expensive to have every frame to be sent to the API. In this case, the interval is set to 2 seconds. To initiate a session with the model and

begin the facial sentiment analysis, a button was added to the UI which begins the analysis. Please refer to the Figure 5.7 for a screenshot of the facial tracking in action.

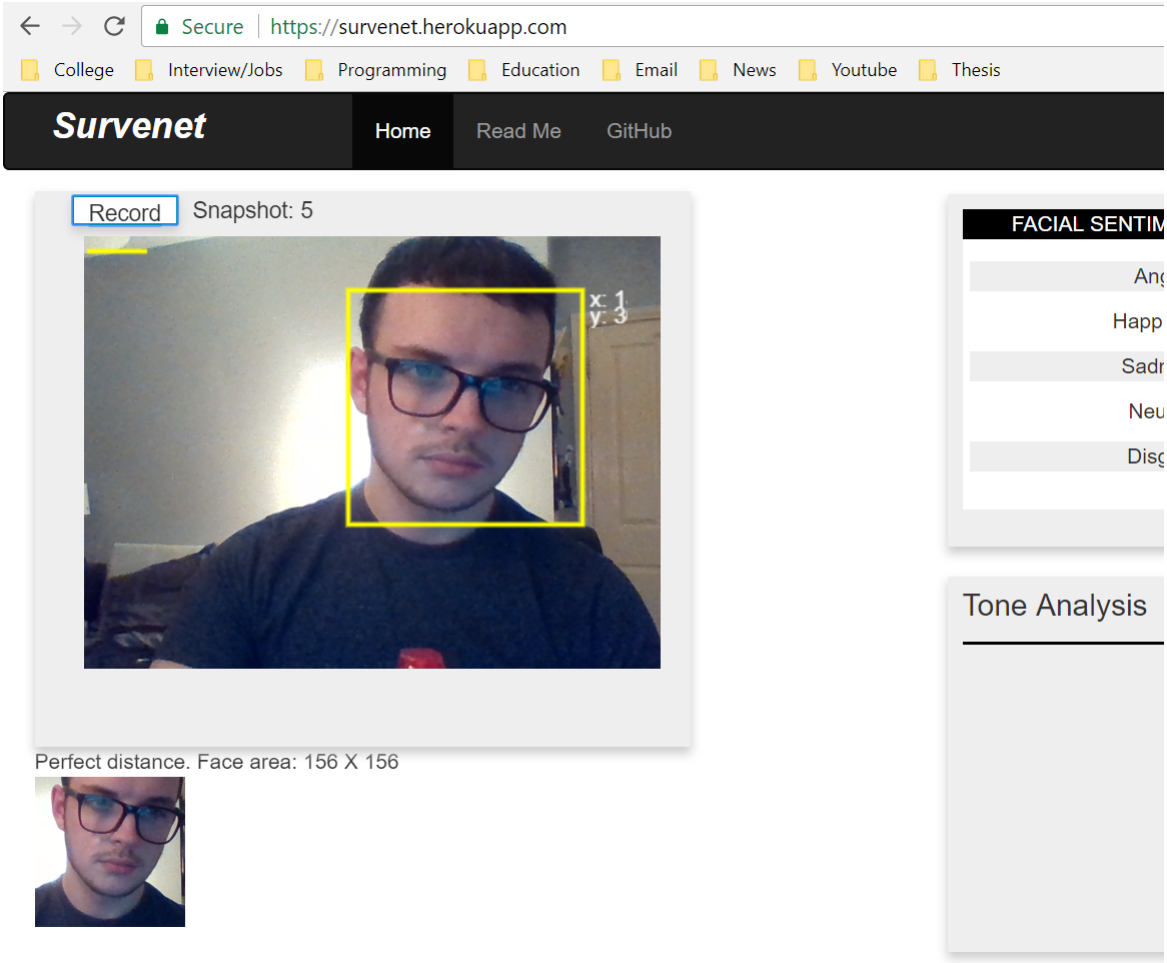

Figure 5.7: Webcam Feed with Facial Tracking and Cropping

5.5.2 Facial Sentiment Analysis

For each of the images sent to the model API, a classification is returned. This classification of facial expression is not only just displayed to screen, but it is used to calculate the percentage of emotions over all. To calculate the relative percentage of a certain emotion, the number of times it is classified is counted, and it is divided by the total number of classifications made. Then this result if multiplied by 100/1. Please see Equation 5.5.1.

$$Class\ percentage\ = \frac{number\ of\ classifications}{Total\ number\ of\ classifications} X \frac{100}{1} \tag{5.5.1}$$

These results are displayed to the screen and are updated constantly after each image is taken. Furthermore, when the Percentage of happiness is above 40%, a notice is given that good customer service is being given, as shown in Figure 5.8.

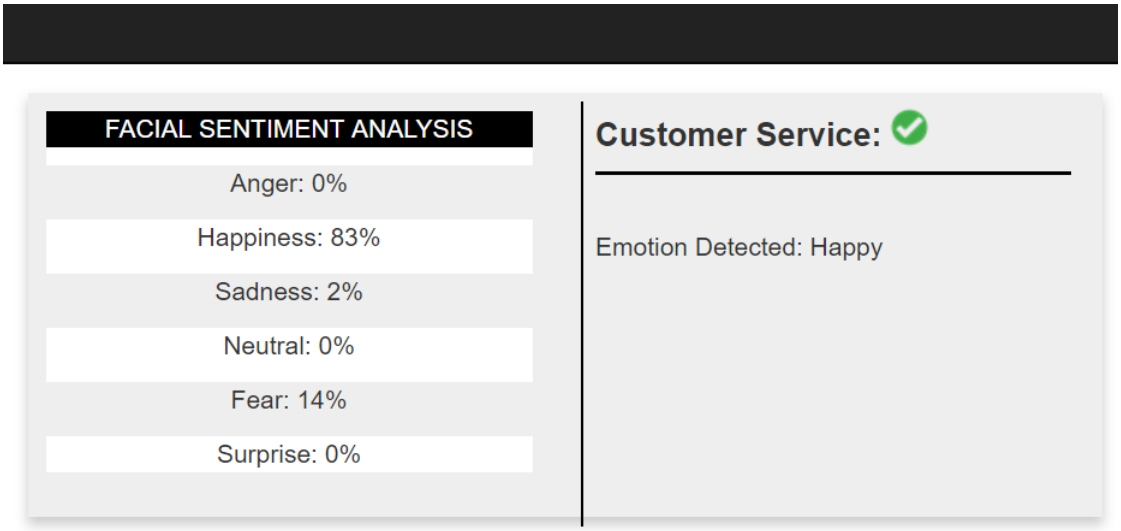

Figure 5.8: Emotion Classification Metrics when Mostly Happy

Also, when the happiness level of the user reaches below 40%, is is notified that poor customer service is being given. See Figure 5.9

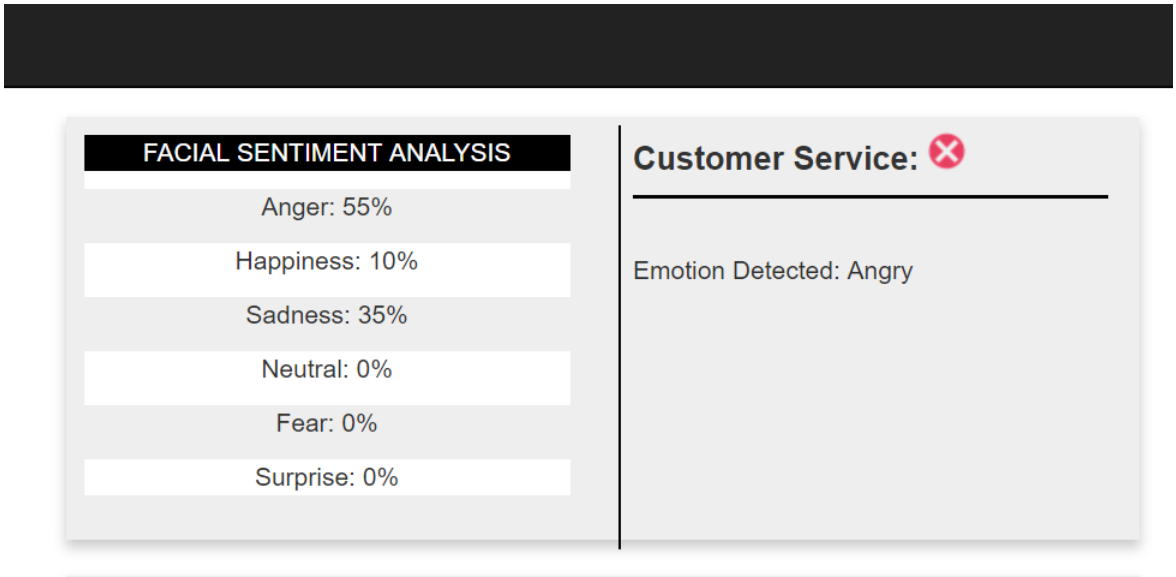

Figure 5.9: Emotion Classification Metrics when Mostly Angry

After 100 iterations of classification, the interaction stops with the model and the user is notified whether or not they have been delivering good or bad customer service.

5.6 Deploying the Node.JS Application

For deploying the web application, the Heroku PaaS of service was used. Using GitHub integrations, whenever a new version of the application was pushed to the remote repository a new build was commenced. If the build was successful, the application would be live for use at *https://survenet.herokuapp.com*.

5.7 Concluding Remarks of Implementation

In summary, many data preprocessing steps were taken. These include grayscaling, facial cropping, data increase by creating flipped copies and image augmentation. These steps were taken to ensure the best possible results when evaluating the model. Secondly, the data

was split into training and testing sets. A fully working TensorFlow convolutional neural network was created in Python and trained on the FloydHub could training platform. The model was then deployed to a server for production use. A facial tracking web application was built for recording the used face and deployed to the Heroku PaaS. The application can accurately calculate the level of satisfaction displayed with the used facial expression for each class.

Chapter 6

Testing, Results and Evaluation

The following chapter will cover the steps taken for evaluating the model in terms of performance and accuracy. Several performance measures are used for determining how well the model handles data that it has not been trained with. These include training evaluation, testing with the testing data and inference with new data. Also, usability testing of the web application will be discussed in detail.

6.1 Training Results and Evaluation

During the training of the model, a number of metrics were recorded for evaluation. As mentioned in the machine learning model section of Chapter 3, The loss (often referred to as the cost function) is calculated by measuring the error of the predicted label in comparison to the true label. This is useful for determining the level of optimization being carried out on the network when using back propagation to adjust the weights. The closer the loss is to zero, the more accurate the model is at classifying. As seen in Figure 6.1, the initially loss for the first epoch was approximately 185. This value decreased gradually throughout the training of the network, getting closer to 0.

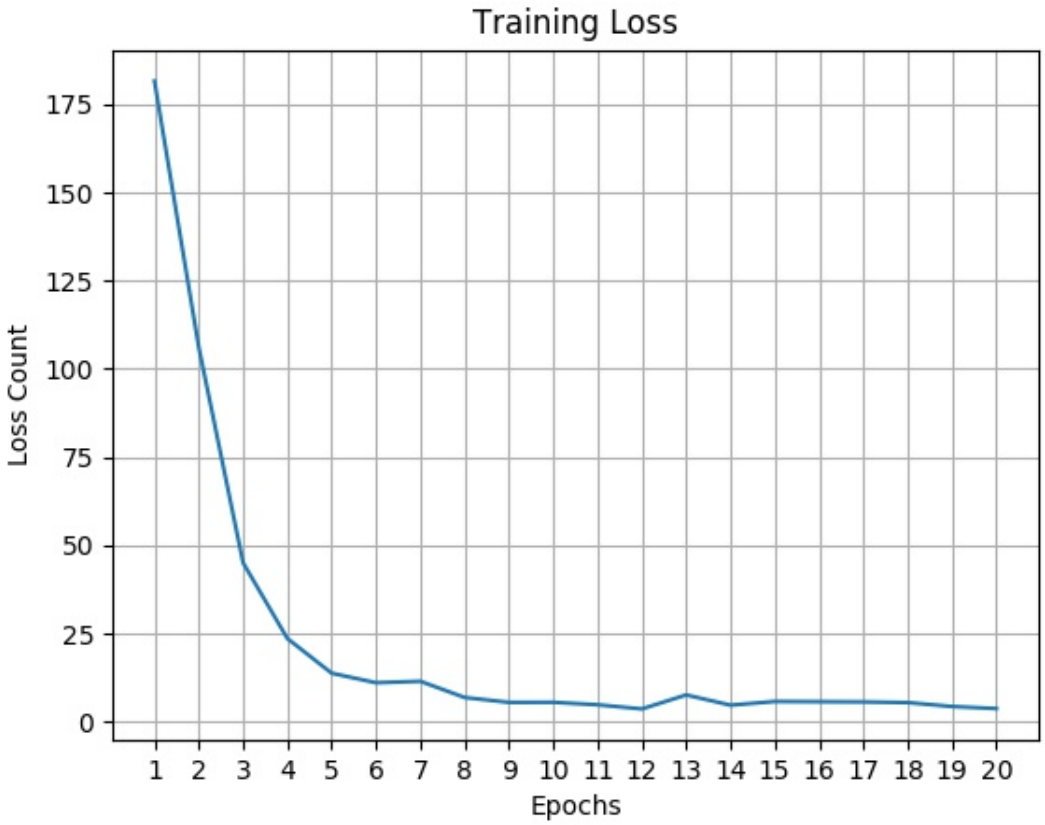

Figure 6.1: Loss Function Values

The second training measurement taken into account was the accuracy throughout each step in training. This is measure by the adding the true positive with the true negative results, and dividing the sum by the total number of samples in the dataset(See equation 6.1.1).

$$Accuracy = \frac{True\ positive\ +\ True\ negative}{Total\ population} \tag{6.1.1}$$

Intuitively, it is desired to have the highest accuracy possible, but this may not always be the case, due to what is known as the accuracy paradox in predictive analytics. It is said that using accuracy is not a desirable method of measuring performance, as it can give misleading results if there is a high class imbalance, but this is not problem in our case as it was made

certain that each class was represented equally in the preprocessing steps. The level of accuracy throughout training was measured and plotted in graph for evaluation. See Figure 6.2.

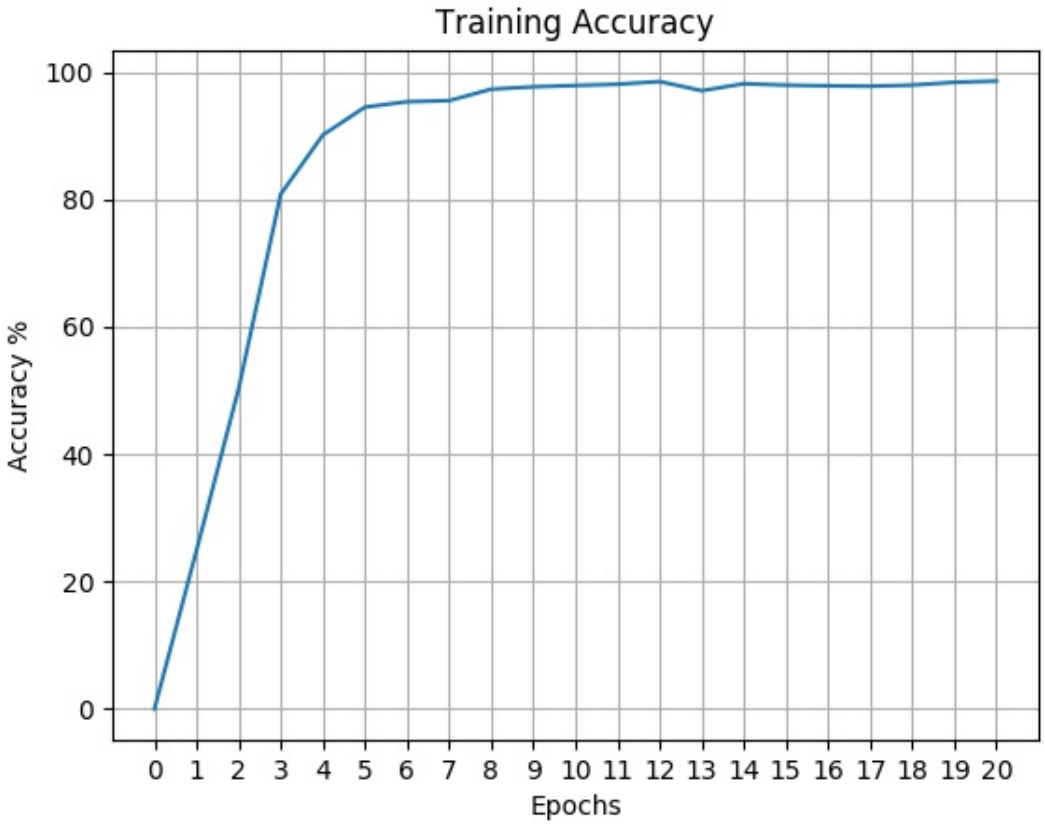

Figure 6.2: Accuracy Values

6.2 Testing Model with Testing Data

As stated in the implementation chapter, the dataset was split into training and testing sets. out of the 1750 images for each class, 1400 of them were used for training, and the remaining 350 were used for testing the model's performance. A python evaluation script was used for loading each set of classes individually and feed them to the trained model.

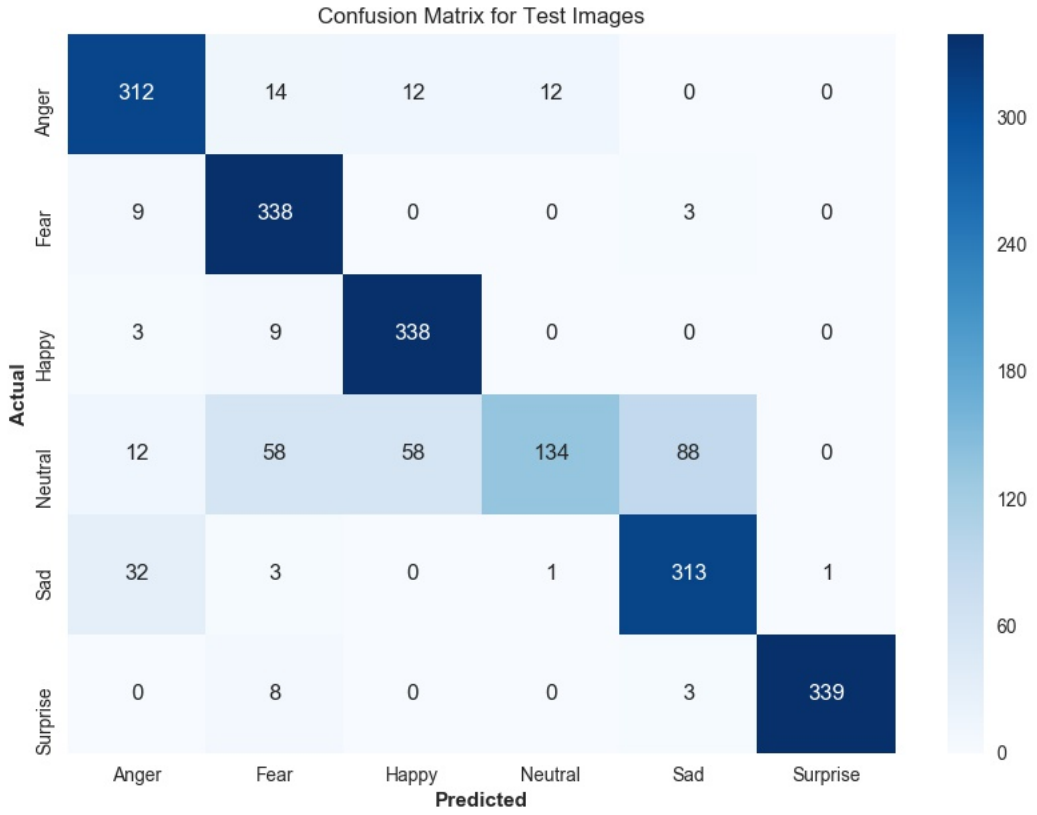

Figure 6.3: Confusion Matrix Heatmap

From this confusion matrix we are able to plot out a number of performance metrics besides the basic training accuracy and loss. Firstly, the recall of each is measured. The recall of a class can be described are the number of true positives divided by the number of true positives and false negatives, as seen in equation 6.2.1.

$$Recall = \frac{True\ positive}{True\ Positive + False\ Negative} \tag{6.2.1}$$

Also, the precision of model can be calculated. The precision is a measure of the number of true positives classified divided by the sum of true positives and false positives, which can be seen in equation 6.2.2

$$Precision = \frac{True\ positive}{True\ Positive + False\ Positive} \tag{6.2.2}$$

From these equations the scores for recall, precision and accuracy are calculated using the data shown in Figure 6.3. Also, the averages for each metric was calculated, as can be seen in Table 6.1.

Testing Performance Measures		
Class	**Precision**	**Recall**
Anger	84.7%	89.14%
Fear	78.6%	96.5%
Happy	82.8%	96.5%
Neutral	91.15%	38.2%
Sadness	76.9%	89.4%
Surprise	99.7 %	96.8%
Accuracy	84.47%	
Average Precision	85.64%	
Average Recall	84.42%	

Table 6.1: Table of Performance Measure Results

6.3 Testing Model with Single Inference

Single inference was performed with images taken from Google Images to see how well the model performed without any of the preprocessing steps. One image was chosen at random for each emotion. See Figure 6.4.

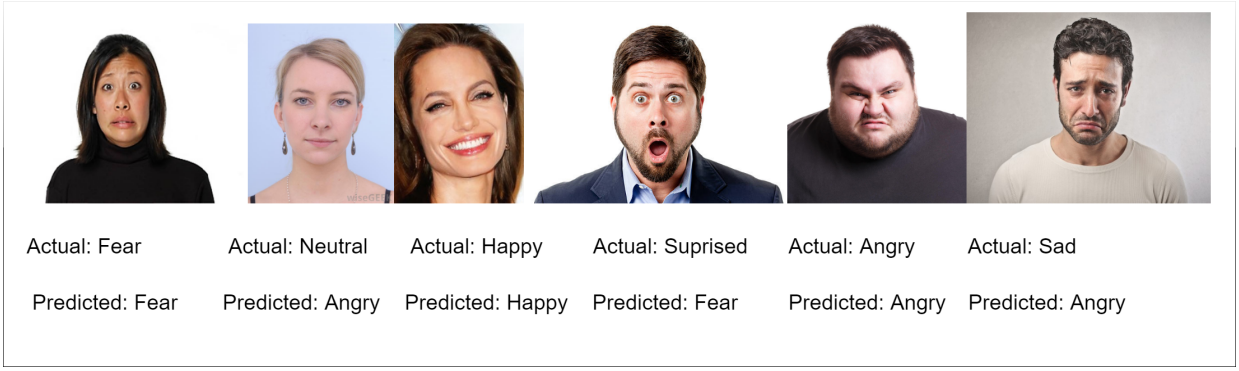

Figure 6.4: Single Predictions on Non-preprocessed Images

6.4 Usability Testing

After model was deployed to the web application, some black box testing was performed. For each of the emotion classes, a 30-snapshot test was conducted using the same facial expression to determine how well the model would classify using the data from the web application. The same facial expression was trailed three times each and the average scores were recorded. These results are percentages of how many times the model classified a certain emotion. Results are shown in Table 6.2

Predicted	**Anger**	**Fear**	**Happiness**	**Neutral**	**Sadness**	**Surprise**
Anger	**71%**	0%	8%	43%	59%	17%
Fear	29%	**89%**	0%	32%	0%	57%
Happiness	0%	11%	**92%**	11%	0%	23%
Neutral	0%	0%	0%	**0%**	0%	0%
Sadness	0%	0%	0%	3%	**41%**	1%
Surprise	0%	0%	0%	11%	0%	**2%**

Table 6.2: Usability Testing Results

6.5 Concluding Remarks of Testing and Evaluation

In conclusion, the training performance evaluation was discussed in regard to the loss and accuracy of the model during the training stage. The loss begins at approximately 185 and is deduced to 3.0 upon the completion of training. The final accuracy displayed in Figure 6.2 displayed a high score of approximately 98%.

Testing was done on the trained model with 350 images per class with a highly correct classification score excluding the neutral class. Recall and precision scores were calculated using a confusion matrix. The neutral class did not have as high of an accurate classification score as the rest of the classes. A speculated reason for this might be because the facial features of the subject may just naturally appear like other facial features, such as sadness, which was classified 88 times out of 350. Also, it is possible that by the augmented images for this class may have morphed the image too much, therefore making them appear to look like another class (happy or sad in this case).

Inference was performed using randomly selected images sourced from Google images in order to examine how the model performed. In terms predictive performance the classifications were only moderately accurate (classifying 3/6 images correctly), but it should be noted that facial cropping needs to be performed on each image for more accurate results.

Lastly, the usability of the web application in conjunction with the trained and deployed model performed pretty good for some classes. Emotions such as happiness, anger and

fear were very well classified when testing, as can be seen in Table 6.2 . The other three classes however did not have as high as a classification percentage. The worst of which was the neutral class which was classified 0% of the time during testing, which is evidently correlated with the fact that the model could not extract accurate features for the neutral class while training. Other factors should also be taken into account: Lighting, webcam quality and facial accessories such as glasses may have an affect and skew the results returned from the model as the dataset did not have the sufficient image data to deal with these conditions.

Chapter 7

Conclusion and Further Work

7.1 Conclusion

In conclusion, this paper examined the literature in regards to artificial intelligence and its use for facial sentiment analysis. Five papers were described in detail and reviewed in terms of quality and consistency. It was found that neuromarketing with artificial intelligence could deem to be a very useful tool for preventing customer churn. Following the assessment of current literature, an analysis was brought forward to give justification for the project and why such a tool would be useful. As stated by Willott, only 10 percent of customer service emails receive a response and that artificial intelligence can automate this tedious task. Furthermore, the technologies required to build such a project were discussed, which included a machine learning algorithm type and library, a large dataset and a cloud platform to host the project for public use. The design framework was laid out in Chapter 4, which describes that a number of data preparation steps needed to be taken, a TensorFlow convolutional neural network needs to be implemented, training should be conducted on the FloydHub cloud service and lastly, the project and API should be deployed to a server on the Heroku PaaS.

In Chapter 5, the implementation of the project was discussed in great detail. A number of data preparation steps were taken to cleanse and increase the size of dataset. Each image was cropped to only the facial regions and grayscaled. Data augmentation was used to create

synthetic samples of the original images. A convolutional neural network was developed using the TensorFlow machine learning library. The model consisted of 10 convolution layers, 5 subsampling layers. Techniques such a shuffling and regularization were taken to prevent over fitting of the data. A web application was built with Node.JS and express to perform facial tracking of user's face. A number of methods for testing were conducted to assess the model and highlight some performance measures. The model was tested on unseen data and the precision, recall and accuracy were analysed using a confusion matrix. It was seen that the model performed very well with an average 84.47% accuracy, 85.64% precision and 84.42% recall. However, the neutral class did not perform as well as the rest of the classes. Furthermore, usability testing was orchestrated to evaluate the performance of the model when given images for the facial cropping in the web application.

7.1.1 Further Work

The work of this thesis project highlighted the promising features that can be added. These steps can be taken to ensure better model performance and make a more user-friendly interaction with the application. The further works are as follows:

- Increase the size of the training dataset by possibly using multiple datasets to provide more variance in the data. This will, in turn, increase the performance, as stated by Lopes et al..
- Perform more data preprocessing steps such as image intensity normalization to allow the model to extract features within an image that may have high contrast lighting.
- Implement a tone analysis tools such as IBM Watson to give further insight into the user's sentiment than just facial expressions. This will record the users voice and convert it to text, then a sentiment analysis to be performed to investigate whether the user's facial expressions match the tone of their spoken words.

- As stated in the Algorithm section of Chapter 3, an alternative to using CNN's was deep belief networks. Should the opportunity arise to do this project again, a possible approach would be to implement both models and determine how each perform against each other.

Bibliography

Christopher Arthmann and I-Ping Li. Neuromarketing the art and science of marketing and neurosciences enabled by iot technologies, 2017. URL https://www.iiconsortium.org/pdf/2017_JoI_Neuromarketing_IoT_Technologies.pdf.

Nikhil Bhargava and Manik Gupta. Application of artificial neural networks in business applications. *GEOCITIES*, 1:3–4, 2017.

Jason Brownlee. Best programming language for machine learning, Sep 2016a. URL https://machinelearningmastery.com/best-programming-language-for-machine-learning/.

Jason Brownlee. Introduction to python deep learning with keras, Oct 2016b. URL https://machinelearningmastery.com/introduction-python-deep-learning-library-keras/.

Jaron Collis. Glossary of deep learning: Bias deeper learning medium, Apr 2017. URL https://medium.com/deeper-learning/glossary-of-deep-learning-bias-cf49d9c895e2.

Adit Deshpande. A beginner's guide to understanding convolutional neural networks. https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/, 2016. [20th July 2016].

Macedo Firmino, Antnio H Morais, Roberto M Mendoa, Marcel R Dantas, Helio R Hekis, and Ricardo Valentim. Computer-aided detection system for lung cancer in computed tomography scans: Review and future prospects, Apr 2014. URL https://biomedical-engineering-online.biomedcentral.com/articles/10.1186/1475-925X-13-41.

Paul Hague and Nick Hague. Customer satisfaction survey: How to measure satisfaction, 2017. URL https://www.b2binternational.com/publications/customer-satisfaction-survey/.

Mohammad Havaei, Axel Davy, David Warde-Farley, Antoine Biard, Aaron C. Courville, Yoshua Bengio, Chris Pal, Pierre-Marc Jodoin, and Hugo Larochelle. Brain tumor segmentation with deep neural networks. *CoRR*, abs/1505.03540, 2015. URL http://arxiv.org/abs/1505.03540.

Salakhutdinov Hinton. Deep belief networks, Oct 2017. URL http://deeplearning.net/tutorial/DBN.html.

Rashmi Jain. 7 powerful programming languages for doing machine learning, Jan 2017. URL http://blog.hackerearth.com/powerful-programming-languages-for-machine-learning.

Reinoud Kaasschieter. Quality data, a must have for ai, Oct 2017. URL https://www.capgemini.com/2017/10/quality-data-a-must-have-for-ai/.

Pathy Kar. Cs231n convolutional neural networks for visual recognition, 2015. URL http://cs231n.github.io/convolutional-networks/#pool.

Keith Kirkpatrick. Ai in contact centers: Artificial intelligence technologies are being deployed to improve the customer service experience. *Communications of the ACM*, 60(8):18 – 19, 2017. ISSN 00010782. URL http://search.ebscohost.com/login.aspx?direct=true&AuthType=sso&db=bth&AN=124418580&site=eds-live.

Andr Teixeira Lopes, Edilson de Aguiar, Alberto F. De Souza, and Thiago Oliveira-Santos. Facial expression recognition with convolutional neural networks: Coping with few data

and the training sample order. *Pattern Recognition*, 61(Supplement C):610 – 628, 2017. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2016.07.026. URL http://www.sciencedirect.com/science/article/pii/S0031320316301753.

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, 2010.

Masakazu Matsugu, Katsuhiko Mori, Yusuke Mitari, and Yuji Kaneda. Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks*, 16(5):555 – 559, 2003. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(03)00115-1. URL http://www.sciencedirect.com/science/article/pii/S0893608003001151. Advances in Neural Networks Research: IJCNN '03.

Dylan Richard Muir. What is the difference between biological and artificial neural networks?, 2016. URL https://psychology.stackexchange.com/questions/7880/what-is-the-difference-between-biological-and-artificial-neural-netw

Michael A. Nielsen. Neural networks and deep learning, 2015. URL http://neuralnetworksanddeeplearning.com/chap2.html.

y Sebastian Raschka. Kdnuggets, 2016. URL https://www.kdnuggets.com/2016/08/role-activation-function-neural-network.html.

Steven W. Smith. The scientist and engineer's guide todigital signal processingby steven w. smith, ph.d., 2011. URL http://www.dspguide.com/ch26/2.htm.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

R. Subhashini and P.R. Niveditha. Analyzing and detecting employee's emotion for amelioration of organizations. *Procedia Computer Science*, 48(Supplement C):530 – 536, 2015.

ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2015.04.131. URL http://www.sciencedirect.com/science/article/pii/S1877050915006407. International Conference on Computer, Communication and Convergence (ICCC 2015).

Adarsh Verma. Most popular programming languages for machine learning and data science, Jan 2017. URL https://fossbytes.com/popular-top-programming-languages-machine-learning-data-science/.

Pierre Vigier. *Gradient Descent Visualization.* 2017. URL https://camo.githubusercontent.com/60aa9cbd0909bab77723a43e21bd430039843c81/68747470733a2f2f707669676965722e6769746875622e696f2f6d656469612f696d6

Paul Viola and Michael Jones. Robust real-time object detection. In *International Journal of Computer Vision*, 2001.

Anish Singh Walia. Types of optimization algorithms used in neural networks and ways to optimize gradient descent, Jun 2017. URL https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to

Lindsay Willott. Customer satisfaction surveys - 5 reasons why they don't work, 2011. URL https://www.customerthermometer.com/customer-surveys/5-reasons-online-customer-satisfaction-surveys-dont-work/.

Zuxuan Wu, Ting Yao, Yanwei Fu, and Yu-Gang Jiang. Deep learning for video classification and captioning. *CoRR*, abs/1609.06782, 2016. URL http://arxiv.org/abs/1609.06782.

Appendices

Appendix A
Code Snippets

It should be noted that this Appendix only contains the main source code snippets and does not contain any of the code written for the data preprocessing steps or the Node.JS web app as there is too much code. For further insight please visit https://github.com/SurveNet.

Appendix A.1
TensorFlow Training Script

```
import tensorflow as tf
import os, os.path
import pandas as pd
import time
import numpy as np
from numpy import ndarray
import skimage
from skimage import data, io, filters, transform, exposure
import random
from PIL import Image
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
```

```
print('imported')

TRAINING_DIR = '/data/training'
TESTING_DIR = '/data/testing'

MODEL_PATH = '/output/trained_model.ckpt'
SAVE = '/output/'

# Define initial variables
batch_size = 100
num_class = 6
num_epochs = 20
image_size = 50 # in pixels, square assumed

################################################
'''
This function traverses throwe ach directory of training images
Two lists are made:
- The RGB image values are added to the images list
- For every photo in say the 'angry' directory of images, a
corresponding label is added to the label list

'''
def load_data(TRAINING_DIR):
images = []
labels = []
directories = [d for d in os.listdir(TRAINING_DIR)
if os.path.isdir(os.path.join(TRAINING_DIR, d))]
# Need to sort these because
# floyd hum jumbled up the order
```

```
directories = sorted(directories, key=int)

# Traverse through each directory and make a list
# of files names if they end in the PNG format
for d in directories:
label_directory = os.path.join(TRAINING_DIR, d)
file_names = [os.path.join(label_directory, f)
for f in os.listdir(label_directory)
if f.endswith(".png")]
#Traverse through each file, add the image data
# and label to the 2 lists
for f in file_names:
images.append(skimage.data.imread(f))
labels.append(int(d))

return images, labels

images, labels = load_data(TRAINING_DIR)

print('data read...')

'''
Shuffle the entire dataset and labels
'''

from sklearn.utils import shuffle
images, labels = shuffle(images, labels)

'''
This cell is  for converting to one hot
```

```
'''
num_images = len(images)
images = np.array(images, object)
labels = np.array(labels, dtype = np.int32)

_labels = np.zeros((num_images, num_class))
_labels[np.arange(num_images), labels] = 1.0
labels = _labels



'''
Plot an image and label  of the same indices after
being randomly shuffled to confirm they
represent the same expression
'''
print('Angry = 0, fear = 1, happy = 2,
neutral = 3, sadness = 4, suprise = 5')

plt.axis('off')
plt.imshow(images[9], cmap='gray')
plt.subplots_adjust(wspace=0.1)
plt.show()



print(labels[9])

################################################
'''
import test data and labels
'''
def load_test_data(TESTING_DIR):
```

```
test_images = []
test_labels = []
directories = [d for d in os.listdir(TESTING_DIR)
if os.path.isdir(os.path.join(TESTING_DIR, d))]
# Need to sort these because
# floyd hum jumbled up the order
directories = sorted(directories, key=int)

# Traverse through each directory and make a list
# of files names if they end in the PNG format
for d in directories:
label_directory = os.path.join(TESTING_DIR, d)
file_names = [os.path.join(label_directory, f)
for f in os.listdir(label_directory)
if f.endswith(".png")]
#Traverse through each file, add the image data
# and label to the 2 lists
for f in file_names:
test_images.append(
transform.resize(skimage.data.imread(f),
 (image_size, image_size) ) )
test_labels.append(int(d))

return test_images, test_labels

test_images, test_labels = load_test_data(TESTING_DIR)

print('Imported...')

#Shuffle data and labels
test_images, test_labels = shuffle(test_images, test_labels)
```

```
'''
This cell is for converting to one hot
'''

num_images = len(test_images)
test_images = np.stack( test_images )
test_labels = np.array( test_labels , dtype = np.int32 )

_test_labels = np.zeros((num_images, num_class))
_test_labels[np.arange(num_images), test_labels] = 1.0
test_labels = _test_labels

print('converted...')

print('Angry = 0, fear = 1, happy = 2,
neutral = 3, sadness = 4, suprise = 5')

plt.axis('off')
plt.imshow(test_images[5], cmap = 'gray')
plt.subplots_adjust(wspace=0.1)
plt.show()

print(test_labels[5])

########################################
'''
This cell is for image downsampling and transformation
This is on the fly to resize the images to a 50x50 size
'''
```

```
print('Down scaling train images...')
images = [transform.resize(image, (50, 50))
        for image in images]

# print('equalizing exposure...')
# images = [exposure.equalize_adapthist(image,
                clip_limit=0.0001)for image in images50]

print('Images Downscaled...')

'''
This cell is for initializing variables for the tensorflow
session and placeholders for holding the data.
'''

# Initialize placeholders
x = tf.placeholder(dtype = tf.float32, shape
= [None, 50, 50], name='X_placeholder')
y = tf.placeholder(dtype = tf.float32, shape=
[None, num_class],name="Y_placeholder")
is_training = tf.placeholder( dtype = tf.bool, shape =
(), name = "is_training" )

#define variables for dropout
keep_rate = .8
keep_prop = tf.placeholder(tf.float32)
print('initialized')

################################################
```

```
'''
Network stucture using tensorflow slim to easily
implement layers

'''
import tensorflow.contrib.slim as slim

def convolutional_network_v2(x, is_training):
net = tf.reshape(x, shape=[-1, 50, 50, 1]) # add channel dim

with slim.arg_scope( [ slim.conv2d ],
padding = "SAME",
activation_fn = tf.nn.relu,
stride = 1,
weights_initializer = tf.truncated_normal_initializer
(stddev=0.01),
weights_regularizer = slim.l2_regularizer(0.0005),
normalizer_fn = slim.batch_norm,
normalizer_params = {'scale' : True, 'trainable' : False,
'is_training' : is_training }):

net = slim.conv2d(net, 32, 3)
net = slim.conv2d(net, 64, 3)
net = slim.conv2d(net, 64, 3)
net = slim.max_pool2d(net, 3, stride = 1 )
net = slim.conv2d(net, 96, 3)
net = slim.conv2d(net, 96, 3)
net = slim.max_pool2d( net, 2, stride = 2)

net = slim.conv2d(net, 128, 3)
net = slim.conv2d(net, 128, 3)
```

```
net = slim.max_pool2d( net, 2, stride = 2)

net = slim.conv2d(net, 128, 3)
net = slim.conv2d(net, 128, 3)
net = slim.max_pool2d(net, 2, stride = 2)

net = slim.conv2d(net, 128, 3)
net = slim.max_pool2d(net, 2, stride = 1)

net = slim.dropout(net, keep_prob = keep_rate,
 is_training = is_training )

with slim.arg_scope([slim.fully_connected ],
weights_regularizer = slim.l2_regularizer(0.0005)):
net = slim.flatten(net)
output = slim.fully_connected(net, num_class,
                              activation_fn = None)
prediction = tf.nn.softmax(output, name = "Prediction" )

return output, prediction

#####################################################
'''
Shuffle the batches on the fly

'''


def randomize(batch_x, batch_y):
batch_x, batch_y = shuffle(batch_x, batch_y)
return batch_x, batch_y
```

```
#####################################################



def train_network(x):
output, prediction = convolutional_network_v2(x, is_training)

#Use Softmax cross entropy for the loss function
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits
(labels = y, logits = output))
total_losses = tf.losses.get_total_loss(
add_regularization_losses=True ) + loss

update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
with tf.control_dependencies(update_ops):
train_op = tf.train.AdamOptimizer( learning_rate=0.002 )
.minimize( total_losses )

# Use these to print out predicted and actual labels
pred_class = tf.argmax(prediction,1)
label_class = tf.argmax(y, 1)

#Accuracy Metric
correct = tf.equal(tf.argmax(prediction,1), tf.argmax(y, 1))
acc = tf.reduce_mean(tf.cast(correct, 'float'))

with tf.Session() as sess:
sess.run(tf.global_variables_initializer())
saver = tf.train.Saver()
```

```
time_full_start = time.clock()
print("RUNNING SESSION...")
for epoch in range(num_epochs):
train_batch_x = []
train_batch_y = []
epoch_loss= 0
epoch_total_loss = 0
accuracy = 0
time_epoch_start = time.clock()
i = 0
number_of_batches = 0

#For all images in the DS, batch into sizes of 100
while i < len(images):
start = i
end = i + batch_size
train_batch_x = images[start:end]
train_batch_y = labels[start:end]

#Randomize the batches even more
train_batch_x, train_batch_y = randomize(train_batch_x,
 train_batch_y)

#Feed batches into tensorflow
op, ac, loss_value, total_loss_value,
pred_classes, label_classes =  sess.run([train_op, acc, loss,
 total_losses, pred_class, label_class ],
feed_dict={x: train_batch_x, y: train_batch_y,
 is_training : True})

epoch_loss += loss_value
```

```
epoch_total_loss += total_loss_value
accuracy += ac
i += batch_size
number_of_batches += 1

accuracy /= number_of_batches
print('Epoch:', epoch+1, 'total loss: ', epoch_total_loss  ,'
 loss: ', epoch_loss  ,' acc: {:%}'.format(accuracy))

time_epoch_end = time.clock()
print('Time elapse: ', time_epoch_end - time_epoch_start)

time_full_end = time.clock()
print('Full time elapse:', time_full_end - time_full_start)

if epoch_loss < 100:
save_path = saver.save(sess, MODEL_PATH)
print("Model saved in file: " , save_path)

print('Accuracy:', acc.eval({x: test_images, y: test_labels,
 is_training : True }))
##############################################

train_network(x)
```

Appendix A.2
TensorFlow Flask API

```
import numpy as np
import os as os, os.path
import sys
```

```
import re
# sys.path.append(os.path.abspath('./model'))
from scipy.misc.pilutil import imsave, imread, imresize
from PIL import Image
from flask import (Flask, request, g, redirect, url_for,
abort, Response, jsonify)
from flask_cors import CORS, cross_origin
import base64
import tensorflow as tf

"""
Import all the dependencies you need to load the model,
preprocess your request and postprocess your result
"""
app = Flask(__name__)
CORS(app) # needed for cross-domain requests
app.config['CORS_HEADERS'] = 'Content-Type'
MODEL_PATH = os.getcwd() + '/data/'

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

sess = tf.Session('', tf.Graph())
with sess.graph.as_default():
saver = tf.train.import_meta_graph(MODEL_PATH +
 "trained_model.ckpt.meta")
saver.restore(sess, MODEL_PATH + "trained_model.ckpt")

# Get pointers to relevant tensors in graph
graph = tf.get_default_graph()
x = graph.get_tensor_by_name("X_placeholder:0")
y = graph.get_tensor_by_name("Y_placeholder:0")
```

```
is_training = graph.get_tensor_by_name( "is_training:0" )
prediction = graph.get_tensor_by_name( "Prediction:0" )



def data_preprocessing(data):
imgstr = re.search(b'base64,(.*)',data).group(1)
with open('output.jpg','wb') as output:
output.write(base64.b64decode(imgstr))

# Every incoming POST request will run the 'evaluate' method
@app.route('/api', methods=["POST"])
@cross_origin()
def evaluate():

imageData = request.get_data()
# CODE FOR DATA PREPROCESSING
data_preprocessing(imageData)
print('1: Image was converted')

image = Image.open('output.jpg')
image = image.convert('L')
image = image.resize((50, 50), Image.ANTIALIAS)
print('Resized and Grayscaled Image')

image = np.expand_dims(np.array(image), axis = 0)
classification = sess.run(prediction, feed_dict =
{x: image, is_training : True})
classes = np.argmax( classification , axis = 1 )

res = 'undefined'
if   classes[0] == 0:
```

```
print('Predicted: Angry')
res = 'Angry'
elif classes[0] == 1:
print('Predicted: Fear')
res = 'Fear'
elif classes[0] == 2:
print('Predicted: Happy')
res = 'Happy'
elif classes[0] == 3:
print('Predicted: neutral')
res = 'Neutral'
elif classes[0] == 4:
print('Predicted: sad')
res = 'Sad'
elif classes[0] == 5:
print('Predicted: Suprised')
res = 'Suprised'

return jsonify(res)
@app.route('/api', methods=["GET"])
def api():

return 'Get Request Recieved...'

# Load the model and run the server
if __name__ == "__main__":
print(("* Loading model and starting Flask server..."
"please wait until server has fully started"))
app.debug = True
app.run(host='0.0.0.0')
```

Appendix B
Screen shots

Appendix B.1
Single Inference with Trained Model

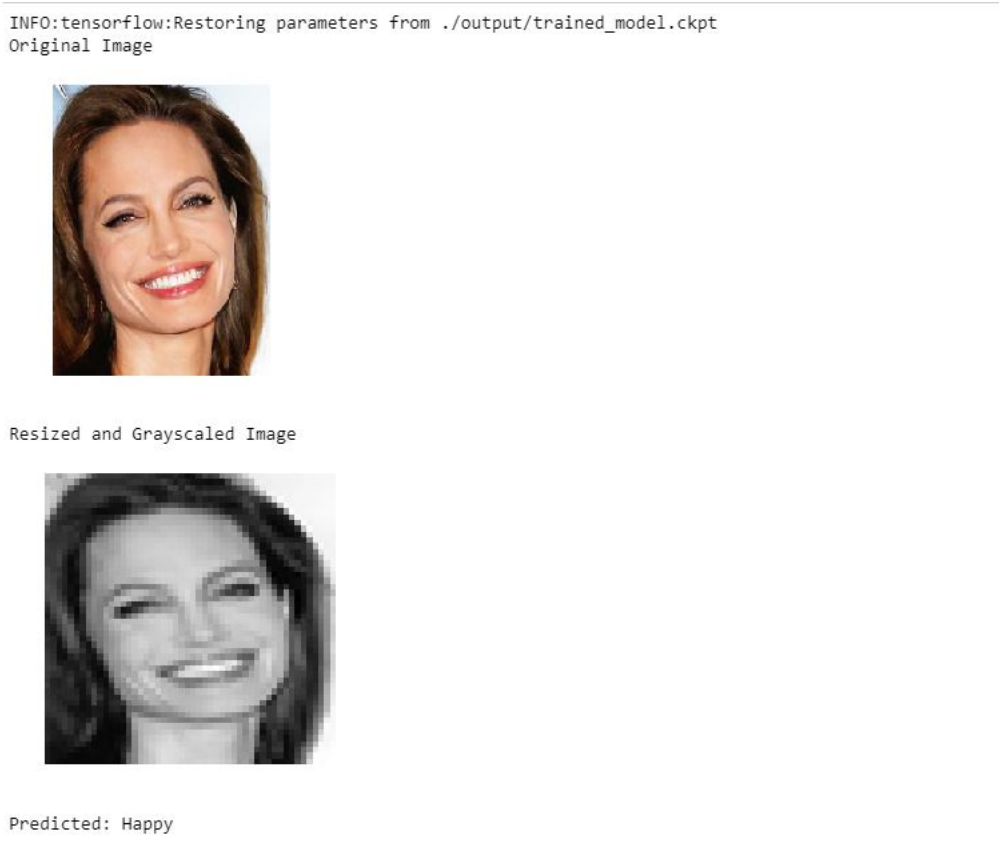

Figure 1: Single Inference with Random Google Image in Jupyter Notebook

Appendix B.2
Web Application

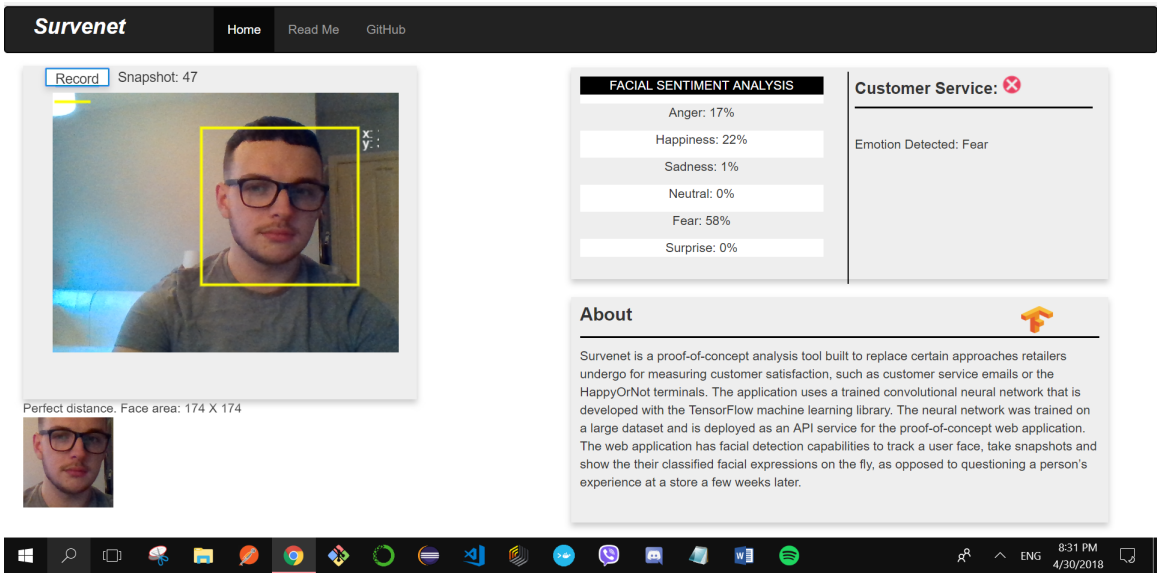

Figure 2: Screenshot of Web Application

Appendix B.3
QR Code for Web Application

Figure 3: QR Code for Survenet - Scan to View Application