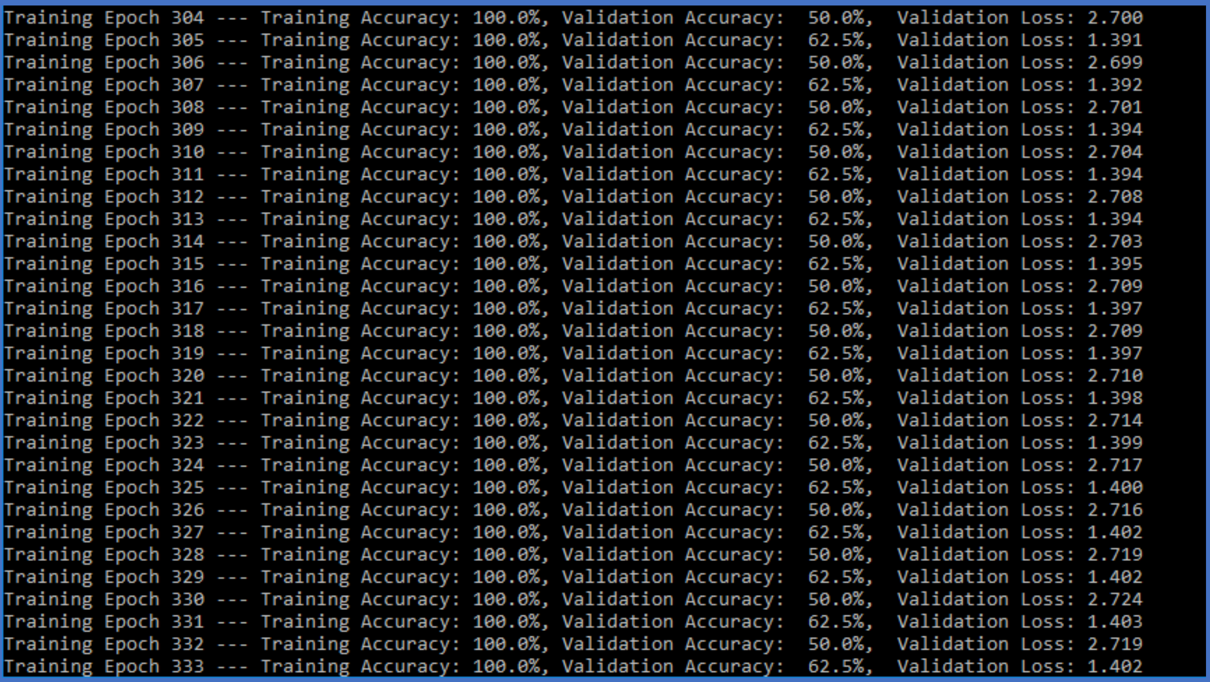
CMP304

Project Report

Human Emotion Recognition



```
Training Epoch 304 --- Training Accuracy: 100.0%, Validation Accuracy:  50.0%,  Validation Loss: 2.700
Training Epoch 305 --- Training Accuracy: 100.0%, Validation Accuracy:  62.5%,  Validation Loss: 1.391
Training Epoch 306 --- Training Accuracy: 100.0%, Validation Accuracy:  50.0%,  Validation Loss: 2.699
Training Epoch 307 --- Training Accuracy: 100.0%, Validation Accuracy:  62.5%,  Validation Loss: 1.392
Training Epoch 308 --- Training Accuracy: 100.0%, Validation Accuracy:  50.0%,  Validation Loss: 2.701
Training Epoch 309 --- Training Accuracy: 100.0%, Validation Accuracy:  62.5%,  Validation Loss: 1.394
Training Epoch 310 --- Training Accuracy: 100.0%, Validation Accuracy:  50.0%,  Validation Loss: 2.704
Training Epoch 311 --- Training Accuracy: 100.0%, Validation Accuracy:  62.5%,  Validation Loss: 1.394
Training Epoch 312 --- Training Accuracy: 100.0%, Validation Accuracy:  50.0%,  Validation Loss: 2.708
Training Epoch 313 --- Training Accuracy: 100.0%, Validation Accuracy:  62.5%,  Validation Loss: 1.394
Training Epoch 314 --- Training Accuracy: 100.0%, Validation Accuracy:  50.0%,  Validation Loss: 2.703
Training Epoch 315 --- Training Accuracy: 100.0%, Validation Accuracy:  62.5%,  Validation Loss: 1.395
Training Epoch 316 --- Training Accuracy: 100.0%, Validation Accuracy:  50.0%,  Validation Loss: 2.709
Training Epoch 317 --- Training Accuracy: 100.0%, Validation Accuracy:  62.5%,  Validation Loss: 1.397
Training Epoch 318 --- Training Accuracy: 100.0%, Validation Accuracy:  50.0%,  Validation Loss: 2.709
Training Epoch 319 --- Training Accuracy: 100.0%, Validation Accuracy:  62.5%,  Validation Loss: 1.397
Training Epoch 320 --- Training Accuracy: 100.0%, Validation Accuracy:  50.0%,  Validation Loss: 2.710
Training Epoch 321 --- Training Accuracy: 100.0%, Validation Accuracy:  62.5%,  Validation Loss: 1.398
Training Epoch 322 --- Training Accuracy: 100.0%, Validation Accuracy:  50.0%,  Validation Loss: 2.714
Training Epoch 323 --- Training Accuracy: 100.0%, Validation Accuracy:  62.5%,  Validation Loss: 1.399
Training Epoch 324 --- Training Accuracy: 100.0%, Validation Accuracy:  50.0%,  Validation Loss: 2.717
Training Epoch 325 --- Training Accuracy: 100.0%, Validation Accuracy:  62.5%,  Validation Loss: 1.400
Training Epoch 326 --- Training Accuracy: 100.0%, Validation Accuracy:  50.0%,  Validation Loss: 2.716
Training Epoch 327 --- Training Accuracy: 100.0%, Validation Accuracy:  62.5%,  Validation Loss: 1.402
Training Epoch 328 --- Training Accuracy: 100.0%, Validation Accuracy:  50.0%,  Validation Loss: 2.719
Training Epoch 329 --- Training Accuracy: 100.0%, Validation Accuracy:  62.5%,  Validation Loss: 1.402
Training Epoch 330 --- Training Accuracy: 100.0%, Validation Accuracy:  50.0%,  Validation Loss: 2.724
Training Epoch 331 --- Training Accuracy: 100.0%, Validation Accuracy:  62.5%,  Validation Loss: 1.403
Training Epoch 332 --- Training Accuracy: 100.0%, Validation Accuracy:  50.0%,  Validation Loss: 2.719
Training Epoch 333 --- Training Accuracy: 100.0%, Validation Accuracy:  62.5%,  Validation Loss: 1.402
```

By: Andrew Peters 1502220

Contents

Introduction

The aim of this project was to build a computer vision application that can distinguish between the universally recognized facial expressions of emotion; joy, anger, sadness, disgust, fear, and surprise. Although there are a variety of ways to approach this, an Artificial Intelligence technique (known as Convolutional Neural Network) was implemented. This would enable maximum accuracy through unsupervised learning.

The project uses three tools to detect emotions. Firstly, OpenCV allowed the project to read images and pre-process them which allowed for a more accurate outcome. Secondly, TensorFlow possesses inbuilt functionality of CNNs which allowed the program to pull in the related functions easily. Finally, Python 3.6.2 was used to create the scripts which will be run from the command line. As most resources available implemented Python it was the most justifiable language to code within. This particular version was implemented due to the fact that TensorFlow is not compatible with the most up-to-date version of Python.

To test the project, the MUG database (N. Aifanti, C. Papachristou and A. Delopoulos) was used with 400 images within it. The data was separated, 80% used for testing and 20% for validation, this was a more robust method which enabled higher accuracy to be gained. Additionally, another python script was made (predict.py)., this script allows the user to test their own image against the training that was gained by the network. It allows the user to see the outcome of new input that was not held within the data set.

The remainder of this document will describe how the project processes the different steps required to obtain a favourable outcome. It will also discuss the results of the training and how well the network performed overall.

Methodology

Pre-processing

The first step that was required was the pre-processing. This stage is necessary to ensure the network detects the lines and the overall shape of an image. This can make the prediction much more reliable especially with this application as we need to be able to detect the eyes and the mouths.

Canny edge detection is used to implement the pre-processing. The following 5 steps are involved when using the canny edge detection;

1. Apply a gaussian filter which will help to remove noise from the image,
2. Find the intensity gradients of the image
3. Get rid of fake responses the edge detection by applying non-maximum suppression
4. Apply a double threshold to determine potential edges
5. Track the edge detection by removing the weak edges that are connected to the strong ones

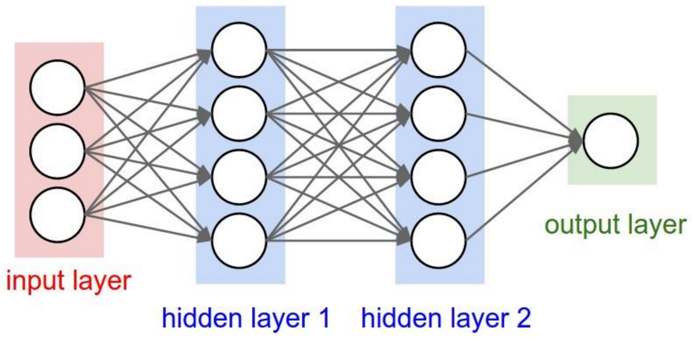
OpenCV contains a function which allows easier use of this method. The code for the script was built by creating variables that are set to the paths of the folders (each emotion has a different folder) where the images are located. It then goes through each of the images in those folders. The script begins by reading the image using OpenCV, applying a grey scale and blurring it. In the next step it calls in the Canny Edge Detection and overwrites the file with the new images ready to be used in the training script. The reason for using the canny edge detection is that it clearly outputs the edges of the eyes, nose and mouth which will be much clearer and help the CNN determine what the image is.

Convolutional Neural Network

The method that is used for the application was Convolutional Neural Network(CNN). This was achieved by accessing and viewing an online tutorial (Sachan, 2018) and adapting it to suit the needs of the project. The way that a CNN works is through different layers. If human faces were taken as an example, the first layer can detect finer details such as edges and dots, the second layer will be able to detect segments like eyes and noses and the third layer will be able to recognise the full face.

The layers take in an image (2d array of pixels) and breaks it up into something resembling a checkerboard. By checking the colour of the pixels around the image, it can predict what the image might be. A potential issue could occur if the image is translated, scaled or slightly rotated, the network could struggle to identify the image and determine that they are not equal.

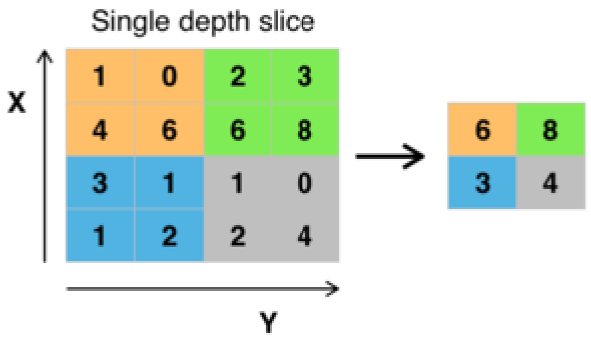
(Figure 1: Cs231n.github.io.)



One of the function of CNNs involves matching up parts of the image instead of the full thing. This is achieved by breaking the image into segments, so it becomes clearer to allow the network to determine which parts are similar or not. Known as features, these small parts can be overlapped with the original to match the image. The way we match them together is through filtering.

The process of filtering is achieved by placing the feature over the image and comparing the pixels. This is achieved by lining up the feature and image patch, multiplying each image pixel by the corresponding feature pixel, add them up then divide by the total number of pixels in the feature. The next step involves taking the same feature and moving it to another location on the image and repeating the filtering process. This essentially becomes a map of where the feature occurs in the image allowing the network to build up what the image looks like through comparison. This is known as convolution. This is reiterated multiple times over the image with other features taken through this process as well to try every possible match. This is called a convolution layer which is essentially a stack of filtered images. The next step involves the pooling layer.

The pooling layer relates to how one shrinks the image stack, which is achieved by selecting a window and then selecting a stride. Following this, the window can be moved across the filtered image which takes in the maximum value. When this is done a similar pattern to the convolution layer is created. However, it is smaller which results in easier use. This aids the network by looking for a feature in the image, regardless of being scaled or rotated it will still pick up the feature. Following this is the normalization layer (ReLUs).



(Figure 2: WIKIPEDIA)

The Normalization Layer changes any negative values to 0. The same process is applied to all the images within the stack of filtered images. This ensures that when the process is performed multiple times the negative numbers will not impact or effect of the overall network.

The next step is to stack up all the layers so that the output of one layer becomes the input of the next layer. The layers can be used multiple times which allows the user to filter the image through the convolutional layers and make smaller through the pooling layers.

The final layer that the CNN goes through is the fully connected layer. Within the fully connected layer every value gets a vote on the potential answer. After going through the other layers, the values get pulled out and brought into a single list and each one of these values connects to one of the classes that it can vote for. The higher the value for the class the stronger the vote. When the CNN receives a new input (an image for example) and it wants to decide on the emotion, then the new image will be passed through all the previous layers. It will compare the weight of the previous votes to the values that came from the new image allowing the network to make its prediction.

Backpropagation

The values that come through the network such as the features and the voting weights are from backpropagation, these are learned by the deep neural network and don’t need to be trained. Backpropagation means that the error in the final answer is used to determine how much the network adjusts. Therefore, the values are added up from the errors of all the classes which helps drive a process called gradient descent. For each feature and voting weight we adjust the gradient up and down to see how the error changes, the amount it changes is determined by the size of the error. The goal is to have the network reach a minimum, so it can perform the best that it can.

The reason a CNN was chosen for this project is because of the way a CNN is based off the idea of the visual cortex which would be the most useful when working with images. It uses unsupervised learning so in the end all we would need to do for the training is add more images to the data set to allow it to train even further. This also means that the code would not need to be greatly altered for someone else to use. The paths of where the data set is located in the script would be required to be modified and the training set would have to be swapped out putting them into separate folders in order to run the training script.

After the network had completed its training a different python script was able to be run. What this did was allow for new input that was not in the original dataset to be tested against the results of the training, so the exact percentages of the prediction could be calculated and seen. This allowed for seeing how close or how far away the network was from coming to the correct solution. As was mentioned previously the weights were compared from the previous results after sending the image through the different layers and the prediction was then printed to the console with percentages to see the effectiveness of the network.

Results

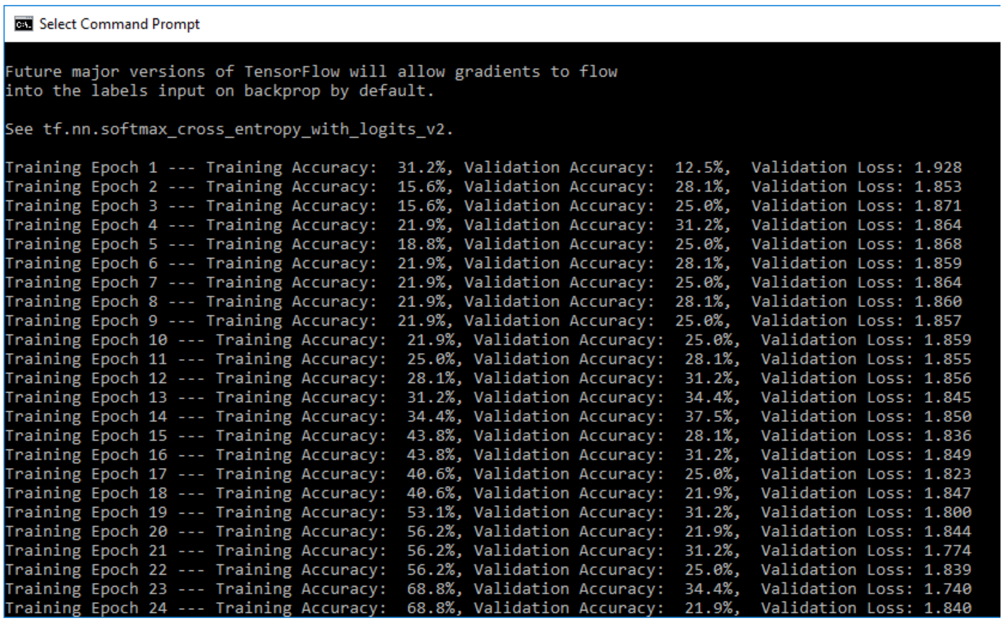
The accuracy of the project can greatly depend on which data set is used. For instance, when the MUG data set was used the accuracy was much higher. This may be a result of the faces all being caught in the same lighting and look the same way. However, with a different data set the accuracy would be lower as there would be more variation within the images. A solution to this would be to add many more of the varied images to have a higher accuracy of the correct prediction.

In the Mug dataset, the Epoch (iteration of the entire data set) improved each time it ran until the training image weights were comprehended 100% of the time and the validation images managed to reach up to 62%. This meant that when new input was entered of an image outside of the dataset there was some correct response from the weights that were built up during the training. This may be improved by adding more images to the dataset with further variation, meaning the weightings became more accurate.

Epoch (Training Iteration)	Training Accuracy	Validation Accuracy	Validation Loss
1	31.2%	12.5%	1.928
2	15.6%	28.1%	1.853
3	15.6%	25.0%	1.853
4	21.9%	28.1%	1.864
5	21.9%	25.0%	1.868
6	21.9%	28.1%	1.859
…	…	…	…
…	…	…	…
329	100%	62.5%	1.402
330	100%	50.0%	2.724
331	100%	62.5%	1.403
332	100%	50.0%	2.719
333	100%	62.5%	1.402

These results highlight that when the network first trains, the training accuracy starts at a reasonable level and the validation is low. The reason for this is that the network got its first set of weights, according to the labels it got from the images, and believed them to be correct. However when it came across the testing data (the other 20% not included in the training) it got confused so the validation was low, this shows in the validation loss and the network is penalized from the deviation between the predicted and true labels (i.e. the emotions). Following this, the training accuracy drops as it restarts in the data set at a random place though with previous weightings and the validation loss to compare with. This showed the network found it made an incorrect decision and tries correcting itself taking on the new weights. This change meant that the accuracy dropped by half however, the validation more than doubled as the network corrected itself.

Running train.py Results

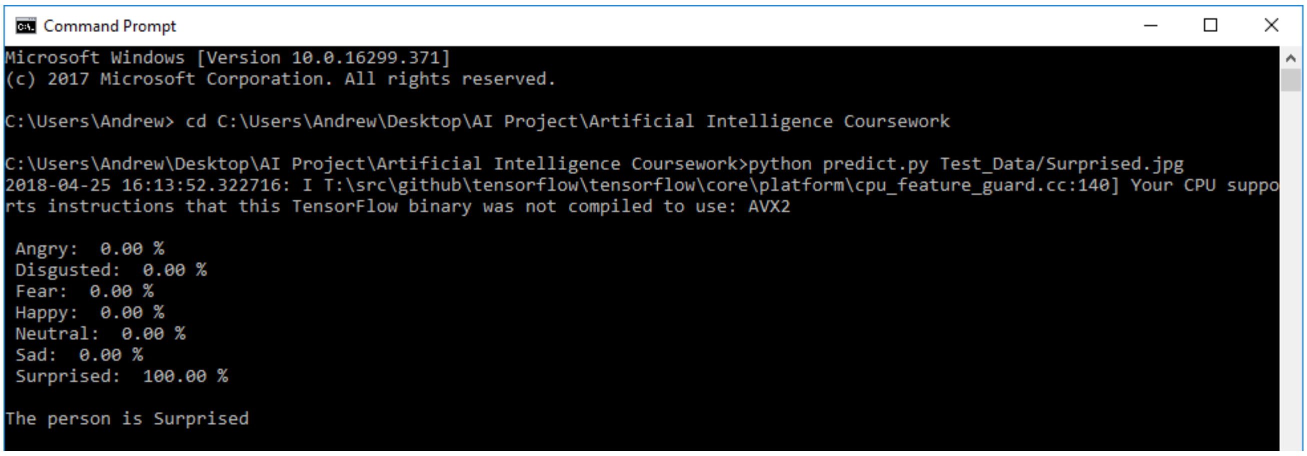


```
Select Command Prompt

Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.

See tf.nn.softmax_cross_entropy_with_logits_v2.

Training Epoch 1 --- Training Accuracy:  31.2%, Validation Accuracy:  12.5%,  Validation Loss: 1.928
Training Epoch 2 --- Training Accuracy:  15.6%, Validation Accuracy:  28.1%,  Validation Loss: 1.853
Training Epoch 3 --- Training Accuracy:  15.6%, Validation Accuracy:  25.0%,  Validation Loss: 1.871
Training Epoch 4 --- Training Accuracy:  21.9%, Validation Accuracy:  31.2%,  Validation Loss: 1.864
Training Epoch 5 --- Training Accuracy:  18.8%, Validation Accuracy:  25.0%,  Validation Loss: 1.868
Training Epoch 6 --- Training Accuracy:  21.9%, Validation Accuracy:  28.1%,  Validation Loss: 1.859
Training Epoch 7 --- Training Accuracy:  21.9%, Validation Accuracy:  25.0%,  Validation Loss: 1.864
Training Epoch 8 --- Training Accuracy:  21.9%, Validation Accuracy:  28.1%,  Validation Loss: 1.860
Training Epoch 9 --- Training Accuracy:  21.9%, Validation Accuracy:  25.0%,  Validation Loss: 1.857
Training Epoch 10 --- Training Accuracy:  21.9%, Validation Accuracy:  25.0%,  Validation Loss: 1.859
Training Epoch 11 --- Training Accuracy:  25.0%, Validation Accuracy:  28.1%,  Validation Loss: 1.855
Training Epoch 12 --- Training Accuracy:  28.1%, Validation Accuracy:  31.2%,  Validation Loss: 1.856
Training Epoch 13 --- Training Accuracy:  31.2%, Validation Accuracy:  34.4%,  Validation Loss: 1.845
Training Epoch 14 --- Training Accuracy:  34.4%, Validation Accuracy:  37.5%,  Validation Loss: 1.850
Training Epoch 15 --- Training Accuracy:  43.8%, Validation Accuracy:  28.1%,  Validation Loss: 1.836
Training Epoch 16 --- Training Accuracy:  43.8%, Validation Accuracy:  31.2%,  Validation Loss: 1.849
Training Epoch 17 --- Training Accuracy:  40.6%, Validation Accuracy:  25.0%,  Validation Loss: 1.823
Training Epoch 18 --- Training Accuracy:  40.6%, Validation Accuracy:  21.9%,  Validation Loss: 1.847
Training Epoch 19 --- Training Accuracy:  53.1%, Validation Accuracy:  31.2%,  Validation Loss: 1.800
Training Epoch 20 --- Training Accuracy:  56.2%, Validation Accuracy:  21.9%,  Validation Loss: 1.844
Training Epoch 21 --- Training Accuracy:  56.2%, Validation Accuracy:  31.2%,  Validation Loss: 1.774
Training Epoch 22 --- Training Accuracy:  56.2%, Validation Accuracy:  25.0%,  Validation Loss: 1.839
Training Epoch 23 --- Training Accuracy:  68.8%, Validation Accuracy:  34.4%,  Validation Loss: 1.740
Training Epoch 24 --- Training Accuracy:  68.8%, Validation Accuracy:  21.9%,  Validation Loss: 1.840
```

Later in the epochs, the accuracy managed to get to 100% with the validation bouncing between 50% and 62%. This shows that the accuracy had reached its optimal point with the small data set it had been given. The weights were over writing each other which gave the 2 different validation losses making the validation accuracy decrease and increase between each epoch as a result.

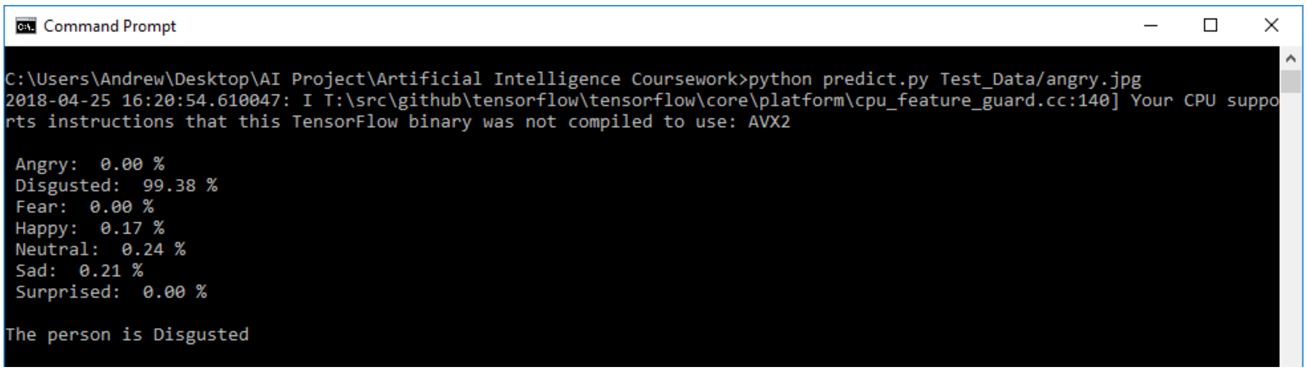
The following images shows the prediction with a new input using the weights that were gained at the end of training:

Surprised Face Input Result:



```
Command Prompt
Microsoft Windows [Version 10.0.16299.371]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Andrew> cd C:\Users\Andrew\Desktop\AI Project\Artificial Intelligence Coursework

C:\Users\Andrew\Desktop\AI Project\Artificial Intelligence Coursework>python predict.py Test_Data/Surprised.jpg
2018-04-25 16:13:52.322716: I T:\src\github\tensorflow\tensorflow\core\platform\cpu_feature_guard.cc:140] Your CPU suppo
rts instructions that this TensorFlow binary was not compiled to use: AVX2

 Angry:  0.00 %
 Disgusted:  0.00 %
 Fear:  0.00 %
 Happy:  0.00 %
 Neutral:  0.00 %
 Sad:  0.00 %
 Surprised:  100.00 %

The person is Surprised
```

Angry Face Input Result:



Command Prompt

```
C:\Users\Andrew\Desktop\AI Project\Artificial Intelligence Coursework>python predict.py Test_Data/angry.jpg
2018-04-25 16:20:54.610047: I T:\src\github\tensorflow\tensorflow\core\platform\cpu_feature_guard.cc:140] Your CPU suppo
rts instructions that this TensorFlow binary was not compiled to use: AVX2

 Angry:  0.00 %
 Disgusted:  99.38 %
 Fear:  0.00 %
 Happy:  0.17 %
 Neutral:  0.24 %
 Sad:  0.21 %
 Surprised:  0.00 %

The person is Disgusted
```

The above image shows that the network was not always able to guess the correct emotion as the weights for angry were over written by the weights from disgust. It can be seen from the result that the network is not fully convinced of this as there are some votes for happy, neutral and sad mixed in.

Conclusion

As aforementioned in the results section of this report, the ability to determine what emotion is being represented has a good accuracy rate considering the size of the data. With more images this would be much higher, and the network would be able to predict the correct outcome on more occasions. To improve on this, the pre-processing could involve another step where the image would be cropped to the face using another algorithm. This would allow the network to only use certain sections of the face which would be passed through all the layers allowing better accuracy.

Another factor which could be improved is the weightings that are stored from the training. Whilst currently the weights are stored from the last training epoch that was run and then overwriting them with new trainings weights. The optimum way for the network to run would be to only save the voting weights if the validation loss was less than the previous validation loss. This would ensure that the application would not bounce between 50 and 62 % but keep at the optimum minimal error to get the highest accuracy, 62.5%

Despite some of the results being negative, the overall ability of the network is high and has allowed for a good rate of accuracy considering that the data set is so small. Improvements can be made to the network, however with the size of the data set the likelihood of it improving would not be very high and might end up confusing the network further. As the project uses unsupervised learning, more classes can be added to it allowing the network to realize a much vaster range of expressions with very little work being needed to adapt the code to suit.

References

Sachan, A. (2018). Tensorflow Tutorial 2: image classifier using convolutional neural network – CV-Tricks.com. [online] Cv-tricks.com. Available at: *http://cv-tricks.com/tensorflow-tutorial/training-convolutional-neural-network-for-image-classification/* [Accessed 10 Apr. 2018].

Cs231n.github.io. (2018). CS231n Convolutional Neural Networks for Visual Recognition. [online] Available at: http://cs231n.github.io/convolutional-networks/ [Accessed 11 Apr. 2018].

Wikipedia (2018). Convolutional neural network. [online] En.wikipedia.org. Available at: https://en.wikipedia.org/wiki/Convolutional_neural_network [Accessed 9 Apr. 2018].

N. Aifanti, C. Papachristou and A. Delopoulos, ”The MUG Facial Expression Database,” in Proc. 11th Int. Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS), Desenzano, Italy, April 12-14 2010.