CMP304 Coursework: Assessment 2

Facial Expression Recognition with Machine Learning

By Aidan Murray 1702270

1. Introduction

Facial expression recognition is the process of a program taking in some kind of image or video, finding faces within them and working out the emotions of the people in the images/videos by predicting what kind of facial expression they have.

This is an important research area as it could be used in many different scenarios such as the government using CCTV cameras to pick up on people's facial expressions in certain areas. This would allow them to see if the people who live in these areas have a healthy mental state or not. Using this data, they could find ways to change these areas for the better, further promoting a healthy life style for the people who live there. People who are diagnosed with autism can sometimes struggle to recognise certain facial expressions and what is meant by them. We can use this method of AI facial recognition to help people with this condition, so they can more easily understand other's feelings. Some practical uses have already come from Facial Expression Recognition such as cars making the driver aware that they are drowsy and should take a break from driving, it has been used in job interviews to see if people are fit for the job, given their mood towards the interviewer and it has even been used in video game testing to see if the player has the expected emotions at different parts in the game (Singh, A. (2018)).

Researchers, Daniel Canedo and Antonio J. R. Neves wrote an article on facial expression recognition (Canedo, D., Neves, A.J.R (2019)) where they use various pre-processing techniques on their images before feeding them into the model. One of the pre-processing techniques is 'Face detection' which detects a face in a given image and can place landmarks at certain places on the face. They then use geometric transformations to correct rotations of the head in the image and to crop the image to the face. The cropped faces are then smoothed to get rid of any noise etc. in the image. They also use data augmentation to create more data to train on by mirroring, scaling and

rotating images etc, as the data-sets for facial expressions are usually pretty small. The features are then extracted using a histogram of oriented gradients and are fed into the program as training data.

The aim of this project is to create a similar algorithm to the one Daniel and Antonio made, but instead of feeding in data from a histogram of oriented gradients, the data will be the normalised distances between certain facial features. This way, the images will not require pre-processing and should be much simpler to implement. By implementing the algorithm this way, we can test whether we will still get decent results with a simpler method. The facial expressions that this AI is trained on for this project will be: sadness, joy, fear, disgust, anger and surprise as these are basic, universal expressions.

In this program you should be able to extract the facial features from both the training set and testing sets of data. The extracted data is then fed into the artificial neural network to train it to learn the differences between the extracted features depending on different facial expressions. After the training and testing data features have been extracted, the model needs to be trained and evaluated. This way we can see how well the model classifies each expression using the micro-accuracy, macro-accuracy and the precision and recall for each class. There should also be a way to test a single image of your choice in the program. This will let you enter the directory of your chosen image and it will output the expression it has extracted from that image.

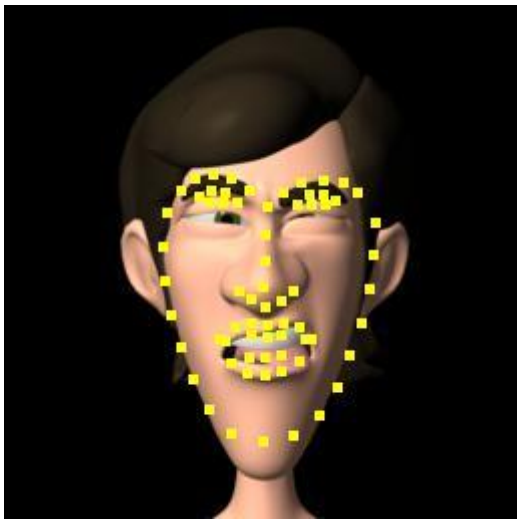
2. Methodology

2.1. Gathering Data:

The first step was to gather data which would be used for training and testing the model. As getting good results requires a lot of data to train on, three different data-sets where used. The MUG facial expression database (Aifanti, N., Papachristou, C. and Delopoulos, A., 2010, April.) was used as there is a decent amount of faces with clear facial expressions. The Cohn-Kanade AU-Coded Facial Expression Database (Kanade, T., Cohn, J.F. and Tian, Y., 2000, March.) was picked for similar reasons, though the quality of this data-set was lower in terms of how good the facial expressions where. The last data

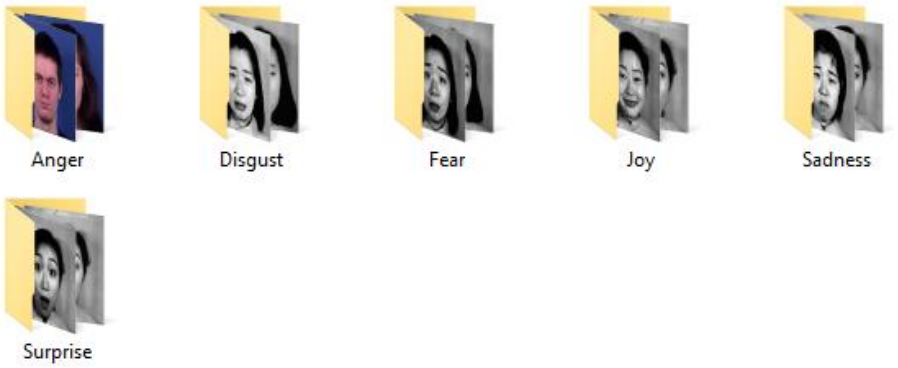
set used was the JAFFE images (Lyons, M., Akamatsu, S., Kamachi, M. and Gyoba, J., 1998) as this set contained images of people of Asian descent which were not included in the other data-sets, although this one wasn't the best as it only contained images of females and some of the people in the images weren't very good at making certain facial expressions.

Another data-set that was considered when picking these sets was the Modeling stylized character expressions via deep learning data-set (Aneja, D., Colburn, A., Faigin, G., Shapiro, L. and Mones, B., (2016)). This wasn't used in the end as the part of the program used for facial detection (DlibDotNet (2017)) didn't pick out the correct features due to the cartoon like models used. Here is an example of this problem:



As you can see, the landmarks for the eyes and lips don't line up properly for the face and this is the case for most of the images in this data-set, that's why this set wasn't used in the final program.

Once all the data was collected, the images were put into separate folders, marked with the name of the expression each image had. This way it would be easy to feed the data into the program and would also make it easier to go through the images manually to see if they were acceptable or not.



Before splitting up the data into training data and testing data, the data-sets had to be manually run through to eliminate outliers in the sets. This can be a lengthy process and can also lead to you having a lot less data then you started with, but can also increase the performance of the model as it will now be training and testing on correct data. Here is one of the outliers found in the MUG facial expression database (Aifanti, N., Papachristou, C. and Delopoulos, A., 2010, April.).

This image is tagged as being part of the 'angry' set, which as you can see, the person in the image does not look very angry. Images like these were removed.

After the data was searched for outliers and organised into different folders, it then has to be separated into training data and test data. This has to be done as we need a certain amount of test data to test if the model is actually giving us decent results. This can be used to find out the various metric values of the system which can help later on with improving the model. The training data and the test data is separated into a ratio of 8:2 with 80% of the data collected being the training data and 20% being the test data. This allows for

there be a lot of data to train on and still a decent amount to test on as the more images for training the better.

2.2. Feature Extraction:

Instead of making the machine learning algorithm train on the pixel colour and placement in the images, feature extraction was used. This works by extracting features from the faces in the images such as the lips and eyebrows. Then the program trains on the normalised distances between certain features which allows it to learn what expressions correlate to different distances between features. The features used in this project are the eyebrows, the left and right side of the lip, the lip width and height, the height and width of both eyes, the distance between the bottom lip and nose, the distance between the left eye and the left side of the lips and the distance between the right eye and right side of the lips as these feature distances are known to change depending on the facial expression.

This method is used as there are many problems when using the method of passing in the pixels from the image. One of the problems being the quality of the image. This is a problem as one image could have less pixels in an area around one of the features on the face leading the program thinking it is something completely different. The image could be framed differently, so the face isn't in the same position in all images, it could also use different colours or it could have noise etc. Using this method would require a lot of image processing before even getting to the training stages, and also wouldn't be able to pick up on a human face in an image if the face was slightly rotated in a certain direction as it would see this as a completely different object. However, using the feature extraction technique and by getting the normalised distances between them, fixes most of these problems as it doesn't matter what quality the faces are and it can work on rotated faces also.

In this project, the DlibDotNet NuGet package by Takuya Takeuchi (DlibDotNet (2017)) was used with Visual Studio 2019 (Microsoft Visual Studio (2019)) to detect a frontal face and mark positions for the eyes, eyebrows, lips, and jaw areas of the face as shown in the images below.

(Aifanti, N., Papachristou, C. and Delopoulos, A., 2010)

(Kanade, T., Cohn, J.F. and Tian, Y., 2000, March.)

As you can see, these images are different in colour, framing and scale, but they are still recognised as faces. This data can be used to find the distances between certain landmarks of interest. For example, the left eyebrow can be calculated as a single floating-point value by adding up the normalised distances between the inner-eye landmark and each point on the eyebrow. These values would be normalised before being added on by first getting the distance between the inner-eye landmark and the inner eyebrow landmark and dividing every other distance by this distance. The reason these features are normalised is because people have different shaped faces than others, so by normalising the distances, this technique can be used for a variety of different shaped faces. If this step wasn't done, the data received be

calculating the distances would be meaningless, as the numbers for each face would differ greatly, leading to inaccurate results.

The feature vectors need somewhere to be store after extracting, so before any feature extraction is done, a CSV file is created with all the headers representing each of the features and the label header being used for the expression used in the image.

```
public void CreateNewCSVFileToExtractTo(string fileName)
{
    // Header definition of the CSV file
    string header = "Label," +
        "LeftEyebrow," +
        "RightEyebrow," +
        "LeftLip,RightLip," +
        "LipHeight," +
        "LipWidth," +
        "LeftEyeHeight," +
        "LeftEyeWidth," +
        "RightEyeHeight," +
        "RightEyeWidth," +
        "LipsToNose," +
        "NoseHeight," +
        "NoseWidth," +
        "LeftEyeToLeftLip," +
        "RightEyeToRightLip\n";

    // Create the CSV file and fill in the first line with the header
    System.IO.File.WriteAllText(fileName, header);
}
```

The individual directories are then specified to be extracted from:

```
public void ExtractData(string directory,string extractTo)
{
    // Extract the neutral folder
    string surpriseDir = directory + "/Surprise";
    this.ExtractExpresionDirectory(surpriseDir, extractTo, "Surprise");

    // Extract the neutral folder
    string sadnessDir = directory + "/Sadness";
    this.ExtractExpresionDirectory(sadnessDir, extractTo, "Sadness");

    // Extract the neutral folder
    string fearDir = directory + "/Fear";
    this.ExtractExpresionDirectory(fearDir, extractTo, "Fear");

    // Extract the neutral folder
    string angerDir = directory + "/Anger";
    this.ExtractExpresionDirectory(angerDir, extractTo, "Anger");

    // Extract the neutral folder
    string disgustDir = directory + "/Disgust";
    this.ExtractExpresionDirectory(disgustDir, extractTo, "Disgust");

    // Extract the neutral folder
    string joyDir = directory + "/Joy";
    this.ExtractExpresionDirectory(joyDir, extractTo, "Joy");
}
```

There is then a loop which goes through each image stored in each of these directories and extracts the feature vectors from each image.

```
private void ExtractExpresionDirectory(string directory, string extractTo, string expression = "Default")
{
    Console.WriteLine($"Extracting Features...");
    string[] inputImages = Directory.GetFiles(directory, "*");

    for (int i = 0; i < inputImages.Length; i++)
    {
        Console.WriteLine($"Extracting from image: {inputImages[i]}");
        ExtractImageFeatures(inputImages[i], extractTo, expression);
    }
}
```

To actually detect the face and the features on the face before doing the extraction, some functions from Dlib (DlibDotNet (2017)) were used.

```
// Set up Dlib Face Detector
using (var fd = Dlib.GetFrontalFaceDetector())

// ... and Dlib Shape Detector
using (var sp = ShapePredictor.Deserialize("shape_predictor_68_face_landmarks.dat"))
{
    // load input image
    var img = Dlib.LoadImage<RgbPixel>(inputFilePath);

    // find all faces in the image
    var faces = fd.Operator(img);
```

The Dlib.GetFrontalFaceDetector sets up the object for detecting faces and then the Operator function is called on that object with an image directory as input to find the faces in a given image. The shape predictor is used to find the facial landmarks in the faces that were found using the face detector, which can then be used to calculate the distances between these features. This is done differently depending on which feature is being calculated. For example, the eyebrows use a function called 'CalculateFeature' which can get the distance between multiple different landmarks, normalise them and added the together to create the eyebrow float.

```
private float CalculateFeature(FullObjectDetection shape, int innerPoint, int normalisePoint, int leftMostPoint, int rightMostPoint)
{
    float feature = 0f;

    // Loop through all the points on the feature adding on the normalised distance between that point and innerPoint
    for (var i = leftMostPoint; i <= rightMostPoint; i++)
    {
        feature += NormalisedDistBetween2Points(shape, innerPoint, normalisePoint, i, innerPoint);
    }

    return feature;
}
```

Whereas, for getting the normalised distance for the width of the lips would only use the 'NormalisedDistBetween2Points' function, which uses an inner point and a point to use to get the distance normaliser. This distance between the two points is then calculated and divided by the distance normaliser to give the final feature float.

```
private float NormalisedDistBetween2Points(FullObjectDetection shape, int innerPoint, int normalisePoint, int firstPoint, int secondPoint)
{
    // Get the positions of the innerpoint and the normalise point and use them to calculate the distance normaliser
    Vector2 innerPointPos = new Vector2(shape.GetPart((uint)innerPoint).X, shape.GetPart((uint)innerPoint).Y);
    Vector2 normalisePos = new Vector2(shape.GetPart((uint)normalisePoint).X, shape.GetPart((uint)normalisePoint).Y);
    double distNormaliser = Vector2.Distance(innerPointPos, normalisePos);

    // Get the position of both points and calculate the distance between them
    Vector2 firstPointPos = new Vector2(shape.GetPart((uint)firstPoint).X, shape.GetPart((uint)firstPoint).Y);
    Vector2 secondPointPos = new Vector2(shape.GetPart((uint)secondPoint).X, shape.GetPart((uint)secondPoint).Y);
    double distance = Vector2.Distance(firstPointPos, secondPointPos);

    // calculate the normalised distance between the points and return the value
    float normalisedDistance = (float)(distance / distNormaliser);
    return normalisedDistance;
}
```

Once all the features have been calculated, they are then concatenated into a string and written onto the next line in the CSV file.

```
//Then write a new line with the calculated feature vector values separated by commas. Check that this works:
if (extractTo != "DontSave")
{
    using (System.IO.StreamWriter file = new System.IO.StreamWriter(extractTo, true))
    {
        file.WriteLine(
            label + "," +
            leftEyebrow + "," +
            rightEyebrow + "," +
            leftLip + "," +
            rightLip + "," +
            lipHeight + "," +
            lipWidth + "," +
            leftEyeHeight + "," +
            leftEyeWidth + "," +
            rightEyeHeight + "," +
            rightEyeWidth + "," +
            lipsToNose + "," +
            noseHeight + "," +
            noseWidth + "," +
            leftEyeToLeftLip + "," +
            rightEyeToRightLip
            );
    }
}
```

This method is applied to both the training data and the test data until all the features vectors have been extracted into two separate CSV files. The training CSV file can then be used for training the model and the testing CSV file can then be used for evaluating the model.

2.3. Training the model:

To train the model, a plugin for visual studio (Microsoft Visual Studio (2019)) called ML.NET (Microsoft ML.NET (No date)) was used. With this you can set up MLContext and IDataView objects to load in the training feature vector CSV file that was extracted in section (2.1) of this report like so:

```
// Load data
IDataView dataView = mlContext.Data.LoadFromTextFile<FaceData>(directory, hasHeader: true, separatorChar: ',');
```

Once the feature vectors have been loaded in, the pipeline is set up to fit the data passed in, so it can be of use for training the model.

```
// Define data preparation estimator
var pipeline = mlContext.Transforms.Conversion
    .MapValueToKey(inputColumnName: "Label", outputColumnName: labelColumnName)
    .Append(mlContext.Transforms.Concatenate(
        featureVectorName,
        "LeftEyebrow",
        "RightEyebrow",
        "LeftLip",
        "RightLip",
        "LipHeight",
        "LipWidth",
        "LeftEyeHeight",
        "LeftEyeWidth",
        "RightEyeHeight",
        "RightEyeWidth",
        "LipsToNose",
        "NoseHeight",
        "NoseWidth",
        "LeftEyeToLeftLip",
        "RightEyeToRightLip"))
    .AppendCacheCheckpoint(mlContext)
    .Append(mlContext.MulticlassClassification.Trainers.SdcaMaximumEntropy(labelColumnName, featureVectorName))
    .Append(mlContext.Transforms.Conversion.MapKeyToValue("PredictedLabel"));

// Train model
model = pipeline.Fit(dataView);
```

Now that the model has been trained using the feature vector data, it can now be saved for testing.

```
// Save model
using (var fileStream = new FileStream("model.zip", FileMode.Create, FileAccess.Write, FileShare.Write))
{
    mlContext.Model.Save(model, dataView.Schema, fileStream);
}
```

2.4. Predicting a Single Image:

For inputting a single image into the system to get the expression used in that image, first the model trained in part (2.3) of the report needs to be loaded in.

```
private void LoadModel()
{
    DataViewSchema dataViewSchema = null;
    using (var fileStream = new FileStream("model.zip", FileMode.Open, FileAccess.Read))
    {
        model = mlContext.Model.Load(fileStream, out dataViewSchema);
    }
}
```

An ML.NET (Microsoft ML.NET (No date)) prediction engine is then set up and the feature vector is extracted from that image into a class called FaceData. The prediction engine can then use this face data to create a prediction, using the model, of what it thinks the expression is. This prediction will then be printed out to the console for the user to see.

```
public void Predict(string imageDirectory)
{
    // Load in the model
    LoadModel();

    // Setup the predictor using the model from the training
    var predictor = mlContext.Model.CreatePredictionEngine<FaceData, ExpressionPrediction>(model);

    // Extract the features from the image and make a prediction
    FeatureExtraction featureExtraction = new FeatureExtraction();
    FaceData faceData = featureExtraction.ExtractImageFeatures(imageDirectory);
    var prediction = predictor.Predict(faceData);

    // Show the predicted results
    Console.WriteLine($"*** Prediction: {prediction.Label } ***");
    Console.WriteLine($"*** Scores: {string.Join(" ", prediction.Scores)} ***");
}
```

2.5. Evaluating the Model:

A function was made to evaluate the model and display the metrics taken. First the model is loaded in, then the test data view is setup and loads in the extracted feature vectors for the test data that was extracted in part (2.1) of the report.

```
// Load in the model
LoadModel();

// Evaluate the model
IDataView testDataView = mlContext.Data.LoadFromTextFile<FaceData>(testData, hasHeader: true, separatorChar: ',');
var testMetrics = mlContext.MulticlassClassification.Evaluate(model.Transform(testDataView));
```

As you can see in the image above, there is also a test metrics variable defined which holds all the test metrics that have been evaluated from the model using the passed in test data.

Using this test metrics variable, useful data can be printed out to the user like so:

```
// Write out to the console
Console.WriteLine($"* Metrics for Multi-class Classification model - Test Data");
Console.WriteLine($"* MicroAccuracy:    {testMetrics.MicroAccuracy:0.###}");
Console.WriteLine($"* MacroAccuracy:    {testMetrics.MacroAccuracy:0.###}");
Console.WriteLine($"* LogLoss:          {testMetrics.LogLoss:#.###}");
Console.WriteLine($"* LogLossReduction: {testMetrics.LogLossReduction:#.###}");
```

The mirco-accuracy, macro-accuracy, log loss and log loss reduction are used here for evaluation as they are good test metrics to use when training a multi-classification model.

The micro-accuracy (also known as the Micro-average Accuracy) shows the combined percentage of correctly classified images of all the classes, so when testing, you want to get this value as close to one as possible.

The macro-accuracy (also known as the Macro-average Accuracy) shows the average accuracy per class. This way every class contributes equally to this value. This is used as well as the micro-accuracy as there may be one class that is underperforming by a large margin and it is important to acknowledge this and try and prevent this. Micro-accuracy is also something that you want to be as close to one as possible.

The log-loss (also known as the Logarithmic loss) shows us the probability of the correct class being picked as the result. This value ranges between 0 and 1, the closer to 0 the better as when this value increases, it shows the predicted probability is diverging from the correct answer.

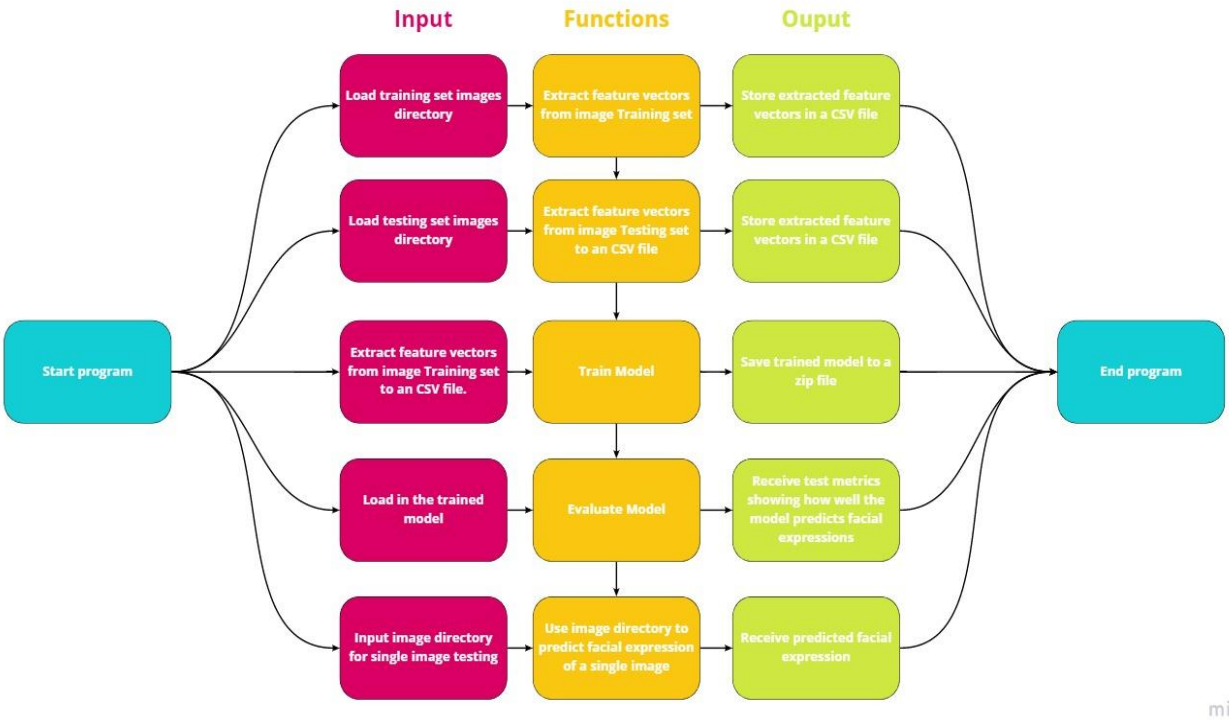
The log-loss reduction (also known as the Logarithmic loss reduction) shows the probability of the system giving a correct prediction.

The program has also been set up to show the precision and recall for each class to make it easier to spot which class is dragging the results down. This way the program can be easily iterated on to find better results.

```
// Precision per class
Console.WriteLine($"* ConfusionMatrixPrecision:");
System.Collections.Generic.IReadOnlyList<double> precisionList = testMetrics.ConfusionMatrix.PerClassPrecision;
for (int i = 0; i < precisionList.Count; i++)
{
    Console.WriteLine($"*     - {(ExpressionType)i} : {precisionList[i]:#.###}");
}

// Recall per class
Console.WriteLine($"* ConfusionMatrixRecall:");
System.Collections.Generic.IReadOnlyList<double> recallList = testMetrics.ConfusionMatrix.PerClassRecall;
for (int i = 0; i < recallList.Count; i++)
{
    Console.WriteLine($"*     - {(ExpressionType)i} : {recallList[i]:#.###}");
}
```

2.6. Program layout:



The program is laid out in a way were the user will be able to choose what they want to do from a menu screen as shown in the graph above. They can choose to either extract the training data, extract the test data, train the model, evaluate the model or use a single image to test and receive a lable for the expression found in that image. The user should also be able to terminate the program after performing any of these as the results and model generated after using these options will be saved. It is recommended to use the menu starting from top till bottom as the connections are shown in the diagram, but there will be pre-made results available to use if the user wants to skip certain stages like the feature extraction as this can take a long time.

3. Results

After extracting the feature vectors and training the model, the extracted testing features were evaluated, giving these results:

```
* Metrics for Multi-class Classification model - Test Data
* MicroAccuracy:    0.785
* MacroAccuracy:    0.772
* LogLoss:          .636
* LogLossReduction: .64
* ConfusionMatrixPrecision:
*    - ANGRY : .844
*    - DISGUST : .8
*    - FEAR : .667
*    - HAPPY : .594
*    - SAD : .857
*    - SURPRISED : .931
* ConfusionMatrixRecall:
*    - ANGRY : .818
*    - DISGUST : .727
*    - FEAR : .588
*    - HAPPY : .905
*    - SAD : .72
*    - SURPRISED : .871
* Press enter to continue:
```

3.1. Micro-accuracy

The micro-accuracy here shows a result of 0.785 which is roughly 79% accuracy overall. This shows that the program has a pretty good chance of giving a correct prediction as a whole.

Even though these are good results, there is still room for improvement. One way to improve these results would be to add a lot more data for the model to train on as after optimizing the data-sets used, there wasn't as many images to work with. This could be achieved by using the training data that is already present and mirroring or scaling the images in different ways in order to create new training data which is different from what the program already has, but still valuable training data. Similar techniques for creating more training data were used in Daniel Canedo and Antonio J. R. Neves's article on facial expression recognition (Canedo, D., Neves, A.J.R (2019)).

3.2. Macro-accuracy:

The macro-accuracy came out to be 0.772 which is roughly 77% accurate. This shows that the average prediction percentage per class was 77%. As this value is slightly lower than the micro-accuracy, it shows that some classes are preforming better than others, but overall, they are still pretty accurate.

This could be improved by trying to balanced out the training and testing data for each of the classes by either adding more training/test data to the class that contains less data or by removing data from a class if it has much more than all the other classes. For example, if the model was training on 100 fear images and only 10 sad images, then the model will be much better at predicting expressions of fear than of sadness, whereas if there was 50 fear images and 50 sad images, the program would have more of an equal chance of predicting each expression.

3.3. Log-loss:

The log-loss result came out as 0.636, this isn't very good as this means than the probability of the program prediction the wrong class for an image is still pretty high.

This is most likely caused by the facial expressions having very similar features, so the likely hood for the predicted value to be another facial expression is still very likely. For example, fear and surprise can have very similar features, such as a wide-open mouth and eyes or very high eyebrows.

3.4. Log-Loss Reduction:

The log-loss reduction result came out to be 0.64 which shows that the model's probability of predicting correctly is 64% better than just guessing randomly. This means that the model is definitely producing meaningful results rather than just guessing each time and getting lucky. This also shows that the results that were achieved for the micro/macro-accuracy weren't just by chance.

Again, these results are most likely due to the problem mentioned in (3.3) where some facial features are very similar, leading to the probability of guessing to be a bit higher, but overall, the results for this are still pretty good.

3.5. Precision Per Class:

The results here show that the precision for: angry was 84%, disgust was 80%, fear was 67%, happy was 59%, sad was 86% and surprise was 93%. This shows that most of the classes have very high and meaningful results. However, the results for happy were relatively low compared to the other classes. This is due to the number of false positives being high, which means other classes are being predicted to be happy when they in fact aren't. This is most likely due to most of the images for disgust being classified as happy. This could be the case as the facial expressions have similar facial features such as the mouth widening and the nose being slightly more pushed up. Humans are able to tell the difference between the two pretty easily, but using a system like this, subtle facial movements aren't detected.

To have the model pick up on these subtle changes, more features could be added for recording more subtle distances changing on the face. This is one of the draw backs of using this feature extraction techniques, as it can be quite tedious when tweaking which parts of the face should be measured and how much each of them should affect the final result. Another solution that could be implemented, would be to add weights to certain features, so some features can have a bigger impact on the results.

3.6. Recall per class:

The results for the recall per class are as follows: angry was 82%, disgust was 73%, fear was 59%, happy was 91%, sad was 72% and surprise was 87%. As you can see, happy performed much better in the recall results than it did in precision. This is due to the number of times happy was classified as a false negative being very low, this means that happy is rarely predicted to be the wrong class, meaning whenever an image appears with the happy expression in it, it is most likely going to predict it as happy. However, fear is 59% for recall compared to its 67% for precision. This means that fear is more likely to be classified as another facial expression. This is most likely due to fear being the smallest data set that the model trained on, as it was harder to find images for fear due to how differently people express fear. The facial feature distances are not very consistent due to large variability of the fear facial expression and because of this it is being categorised as other expressions.

This could be improved by increasing the number of images in the data-set specifically for fear and removing the more subtle looking fear expressions

from the data-set to make it more clear for the model during the training stage. This could be done by searching the web for more fear images or by using the method mentioned previously in (3.1) where you can mirror and scale the current data in the data-set to acquire more fear images.

4. Conclusion

The initial aim of this project was to create a facial expression recognition machine learning model and train it using only normalised distances between different facial features and still achieve reliable/meaningful results. As you can see from the results found by doing this, this project was definitely a success, as a working, reliable, facial expression classifier has been created.

The first time getting the results from the program, the accuracy was as low as 40%. This was when only the MUG facial expression database (Aifanti, N., Papachristou, C. and Delopoulos, A., 2010, April) was being used and there were initially only 7 facial feature distances being calculated. After adding more data-sets to the training and testing data the results improved by a small amount, showing a micro-accuracy of 50%, but it would still jump up and down every time it was tested. This is when the data-sets where optimised by removing many of the outliers, bringing the micro-accuracy to roughly 63%, but the individual precision and recall for some classes were still below 40%, so more features of the face were measured and added to the feature data. This increased the micro-accuracy to roughly 79% and also increased all the individual percentages for the precision and recall as they were now getting more data about what shape the face was in certain images, leading to more correct predictions.

In the article made by Daniel Canedo and Antonio J. R. Neves (Canedo, D., Neves, A.J.R (2019)) they found results with much higher accuracy, which is understandable as they looked at many different factors such pre-processing their images and using a histogram of oriented gradients as the input for the model to train on instead, as this would hold much more data about the movements in the facial muscles. This was expected as the method used in these papers go much more in-depth and take a lot longer to implement, whereas this program focused more on using a simpler approach to training a model to learn facial expressions using only normalised distances between facial landmarks.

Some of the limitations to using this method of facial expression recognition are:

- Some images/data-sets of images can't be used due to Dlib's (DlibDotNet (2017)) frontal face detector not being able to detect faces in some conditions, such as the cartoon characters in the Modeling stylized character expressions via deep learning database of images. This could be improved by trying out different face detector to see if there are any available that would work with cartoon faces as well as real human faces.
- Due to Dlib's (DlibDotNet (2017)) facial landmark placement, there is a limitation on the amount of features you can use. Using something like a histogram of oriented gradients would give you access to much more detail in the facial movements.
- When adding more training and testing data, there can be more outliers causing the performance of the program to drop. To get rid of these you could manually go through all the images and remove them or you could try and find a data-set that someone else has already checked over and verified that there are no outliers in.
- Whenever changes are made to the program, the feature vectors for the training and testing data need to be recalculated which can take quite a while, this is also the case for adding more data to the data-sets. To fix this, the code for extracting the features could be optimised using threading on the CPU or a compute shader on the GPU which will allow multiple images to extract features in parallel.
- As the percentages for prediction aren't 100%, the predictions aren't always going to give the right answer, which could be problematic if the program was to be used professionally. There is no real way to fix this, as the model will never achieve 100% accuracy, but you can get very close by optimising the model. This can be done in many different ways, such as adding more data to train on, balancing the instances of each class that the model is training on, extracting more features, giving different features different weights etc.
- Each extracted feature contributes equally to the results, which can be a problem if some features are more meaningful than others. As mentioned in the previous point, this can be solved by assigning different weight values to each of the features, so you can control how much of an effect each feature has.

To make this program better, it could be upgraded to use video footage as well as images as this would provide a lot more samples for training and testing. It could also be improved using the mirroring and scaling technique to add more to the training data. Weights could be added to each of the features. More features could be extracted, because the more information the model has about each expression the better. CPU threading or a GPU compute shader could be used to speed up the image feature extraction, making it much faster to test and upgrade. A more complex face detector and face landmark positioner could be used, so there would be more data to extract features from.

This project was an attempt to make a facial expression recognition program using machine learning by using the normalised distances between certain key features in order to get reasonable results. Overall, the program was a success as reasonable results were achieved and a facial expression recognition program was made with machine learning. This shows that, to get pretty good results, you don't need to go through lots of pre-processing steps before hand, unless you really want to get the best possible results. Obviously, the program isn't perfect and can still use a lot of improvements, it does what it was made to do in a simple and efficient way.

5. References

DlibDotNet (2017) - *DlibDotNet 19.18.0.20200301* - [Code Library]

Available at: https://www.nuget.org/packages/DlibDotNet/

Accessed: 10 April 2020

Singh, A. (2018) - *Facial Emotion Detection using AI: Use-Cases* - [Online Article]

Available at: https://blog.paralleldots.com/product/facial-emotion-detection-using-ai/?0

Accessed: 10 April 2020

Canedo, D., Neves, A.J.R (2019) - *Facial Expression Recognition Using Computer Vision: A Systematic Review* - [Online Article]

Available at: https://www.mdpi.com/2076-3417/9/21/4678/htm#cite

Accessed: 10 April 2020

Cohn-Kanade AU-Coded Facial Expression Database: Kanade, T., Cohn, J.F. and Tian, Y., 2000, March.

Comprehensive database for facial expression analysis. In Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition

(Cat. No. PR00580) (pp. 46-53). IEEE

Aifanti, N., Papachristou, C. and Delopoulos, A., 2010, April.

The MUG facial expression database. In 11th International Workshop on Image Analysis for Multimedia Interactive Services WIAMIS 10(pp.1-4). IEEE

The JAFFE images: Lyons, M., Akamatsu, S., Kamachi, M. and Gyoba, J., 1998, April. Coding facial expressions with Gabor wavelets.

In Proceedings Third IEEE international conference on automatic face and gesture recognition

(pp. 200-205). IEEE.

Aneja, D., Colburn, A., Faigin, G., Shapiro, L. and Mones, B., (2016).

Modeling stylized character expressions via deep learning.

In Asian conference on computer vision (pp. 136-153). Springer, Cham

Microsoft Visual Studio (2019) - [IDE]

Available at: https://visualstudio.microsoft.com/vs/

Accessed: 11 April 2020

Microsoft ML.NET (No date)

Available at: https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet

Accessed: 11 April 2020