NEA Investigation Into Facial Recognition

Luca Djehiche

candidate Number: 0611

—

Centre Name: Barton Peveril College
Centre Number: 58231

—

GitHub Page: https://github.com/LucaDjV/NEA-Investigation-into-facial-recognition

Contents

1 Analysis

1.1 Statement Of Investigation

Can I write a facial detection algorithm that can be run on images?

More specifically:

- **Can I write a facial detection algorithm that will be able to run on a variety of faces in different lighting and resolutions?**

  This question will guide me to make sure that I avoid overfitting and help me to choose an algorithm that actually ends up being practical. It will also mean that I will need to use image processing algorithms to manipulate low or high resolution images.

- **Can I optimise my algorithm to process images using parallelism to speed up matrix multiplication etc of images 1 at a time?**

  This will make sure that my algorithm will run fast on a variety of machines making it useful on not only powerful computers but phones which often have less throughput.

I plan to investigate Machine Learning techniques and dynamic programming by developing a facial recognition algorithm in which a program will receive an input of an image and decide whether it is a cat or not.

To do this I will need to consider what constitutes a successful algorithm, do research into optimisation techniques and understand the mathematics behind the models.

The program will be tested in the end with a set of test data where I will measure the speed and accuracy of the solution. I will need to check if it runs on a variety of different performance computer systems as well to truly see if it is successful.

1.2 Background

Machine Learning is a vital part of modern life and more specific object detection and object recognition vastly improve systems in areas such as immigration where facial recognition allows people to be identified autonomously, photography where facial detection can apply effects such as depth of field and filters and in detecting tumours in patients. The applications of this technology will only increase in the future and it may form the basis of the medical screening and security systems so it is important to research new methods and look into optimisation of old ones to future proof our systems. Machine learning works by approximating a classification by adjusting the graph's weights and biases according to how inaccurate the output is. It is important to recognise that the computer doesn't see the face or object but only makes an educated guess on what it is based on the previous examples.

Now, after the advent of Boltzmann machines and statistical analysis techniques, the common methods for facial detection can use the image's texture (the dark and light parts of the object) and structure (the edges on the object). These are identified through matrix operations to form a decision tree or forest that is able to classify the the face. Alternatively, facial detection can be done with neural networks which are based off of Boltzmann machines and the idea of a perception. These use a combination of the image's texture and structure of training examples in singular epochs to learn which features make up a face by adjusting the weights and biases of the model in back-propagation.

Examples of implementations of facial detection are:

1. Snapchat

   Snapchat uses machine learning facial recognition in order to find the face on the screen for its filters. It uses an the HOG (histogram of oriented gradients) to create a mathematical representation of the sections in the image. It uses these to learn what a face looks like by passing a large array of data into an SVM which provides division between what a face is and what it isn't. Snapchat then places points on the face in order to create a mesh which it can map the mask to.

2. Cameras

   Older cameras use the Viola Jones algorithm to identify the location of faces on the screen. They use this algorithm in particular as it was much faster than alternatives at the time and was able to process video, boxing out the face on the screen at 15 frames per second.

Another part of my investigation will be in image processing which is a bi product of the nature of face detection. Image processing is widely used in modern day technology where images need to be scaled up and down, rotated or flipped at any moment when processing important documents such as passport photos and displaying bitmap images on a computer screen. This is also very important in machine learning where we have kernels of a fixed size for algorithms like viola jones and need to combine images in approaches involving for example eigenfaces and in general where we don't want to add data to the images that may hinder the final result or result in overfitting.
The blight of machine learning algorithms, overfitting, is an important concept which is where the machine learning model works perfectly for the training data but then ends up failing on test data or real world data. This could be for a number of reasons but we develop techniques such as shuffling the dataset to handle this and reduce the chance of there being any unforseen relationships between the training data.

1.3 Expert

I chose my aunt's friend Elliot as my expert for my project as he is experienced with image processing concerning applying a facial mask to an image. He has done a few machine learning courses and has experience making a deepfake program which had to compare 2 videos and process them image by image to learn what each face was and then switch them.

1.3.1 *Interview Questions*

1. **Which do you think is more important, the quality of the algorithm or the dataset?**

   I think with the same dataset different algorithms will learn differently because certain algorithms will tell you how to connect different points together as a mesh and others will do different things so, concerning the end result, an algorithm that is designed specifically for the purpose will run a lot faster and end up only producing what you need which is more memory efficient. More importantly, the dataset you choose will influence what the end result is capable of as, with images that are very blurry, it will be very hard for an algorithm that is trained on 4k images to classify because images in 4k have really well defined edges and they contain lots of information so the dataset is really important for the end result.

2. **What approaches are there to facial recognition?**

   From my experience the main thing is edge detection and edge learning but there are other algorithms out there like eigenfaces or viola-jones.

3. **What optimisation techniques have you seen?**

   You could 1. queue it all up on your own computer or 2. run it on a virtual machine where you can modify the computing power and how much RAM it has to see how fast or slow it is. You should also look into multithreading or code parallelism in the GPU with libraries like OpenCl or Cuda etc...

4. **Should I attempt to support multiple bit depths and picture formats?**

   I dont think so, I think you would make your life a lot harder without really making the end result much better. You would need to have separate algorithms that would be able to digest and support HEIC files for example in the long term so I would stick to files that are very basic. I would stick to files like PNG files because it is very widely used and you would be able to create the algorithm to actually get the file without having to think about the file type very much. If you had the computing power you could to work with raw files but thousands of raw files will take up a huge amount of space and I think it would be a bit over the top.

5. **What is the best way of storing and reading from a dataset?**

   When I was doing my deepfake program I had to match the faces from videos so to store the video I used a folder of a huge amount of images from the video and then ended up having to delete all of the files manually which was quite cumbersome so I would recommend using a folder of temporary files which are files with an expiration of like 1 day for example so then you can automate the process which is a lot easier.

6. **What aspects of a program like this do you think make it successful?**

   If you were google, you would probably produce something that is both accurate and fast but that is obviously very challenging so, in modern industry, I think that consumers want to get the results as fast as possible because we don't have high attention spans like you wouldn't want a super accurate face id that takes 10 minutes to open your phone so I would focus on speed above accuracy where you have acceptable accuracy like at least 90 percent.

7. **What should I research before I start programming?**

   I would suggest looking into google cloud because you can run the programs on their servers which will keep your computer from becoming blocked up and also they have more processing power. My deepfake program was all done on google cloud and it ran surprisingly fast taking like half an hour but it depends because I'd probably just do the small testing data on my own computer.

8. **What would you say is the easiest way of tuning my model?**

   Make sure to output some graphs and graphics of what exactly is happening and what the algorithm is doing to the image so that you can monitor and understand what is going on and change a few things when you see something that is not right because we are by nature graphical learners and outputting numbers to console won't actually mean anything to you.

9. **How should I deal with lighting in images where a face in one situation could look completely different in another?**

   Maybe when you take your input you could mess around with the contrast and try and make it the same as the other images because you how dark it is by looking at the average pixel values. You could also modify your training data so it is composed of a wide range of lighting and quality.

1.3.2 *Interview Evaluation*

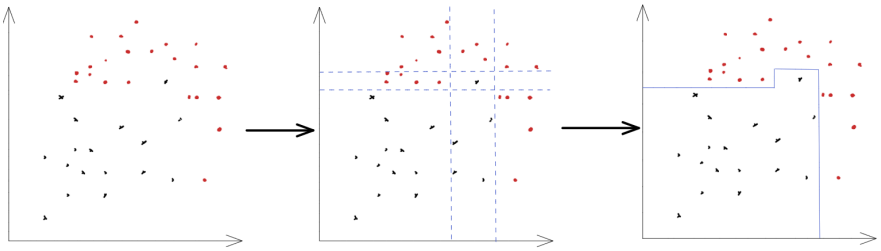
My main takeaways from the interview are:

- Making sure that I use a quality dataset that is personalised to my own needs.
- Having a model fit for the purpose that doesn't output more than I need for analysis and the final classification.
- I will focus my program and research into the optimisation and speed of the project.
- I will research Viola-Jones algorithm, eigenfaces and edge based deep learning.
- Use a Virtual Machine in my testing to analyse the portability of the project onto heritage and slower computers as well as high-end computers.
- Research and choose between Cuda and OpenCl.
- I only need to support PNG format.
- I will store my data in temporary files on the google cloud while the algorithm is training and on my computer when testing.
- I will output graphics of the data while it is running I can tune the model.
- Increasing the variance of the data may be useful to reduce overfitting.

1.4 Initial Research

1.4.1 *Viola Jones Algorithm*

This approach considers the texture and shading of the object you are trying to identify. It relies on the designer identifying the key features of the object in terms of the areas that are on average darker or lighter than their surroundings in what's called a haar cascade[1]. Viola Jones uses **AdaBoost** to generate a forest of stumps that classify the item through each stump's vote. AdaBoost is a useful algorithm for decision trees as it can turn an array of linear classifications into a non linear form which is useful as data in reality is not always linearly split. For example:



1.4.2 *AdaBoost*

AdaBoost is a popular method for training a model based off of decision trees. The steps for AdaBoost are:

1. Add a new value to each record which is the record's record weight. This is calculated as
$$\frac{1}{Amount\ Of\ Records}$$
2. Generate stumps for each field in the dataset that may or may not be indicative of the correct classification. In this case, our stumps are our haar features where the leaves are whether that haar feature would classify the image as a face or not. 'A stump is a tree with just 1 node and 2 leaves'[2]
3. Then use the stumps to get the result of each stump's classification
4. The stumps will produce a range of values that may or may not be correct for each record. Sum the correct and incorrect classifications for each datapoint in each field and then work out the Gini index which is:
$$1 - \left[\left(\frac{AmtIdentifiedCorrectly}{TotalNumberConsidered}\right)^2 - \left(\frac{AmtIdentifiedIncorrectly}{TotalNumberConsidered}\right)^2\right]$$
5. The stump with the lowest Gini index is then the most successful stump.
6. If the stump has a Gini index of 0, that is your correct classifier and the algorithm does not need to run any further.
7. Work out the vote for each stump with formula below where the total error is the sum of all of the record weights of the records that the
$$Vote = \frac{1}{2} log\left(\frac{1 - TotalError}{TotalError}\right)$$
(equation from [2])
8. The vote indicates how accurate the stump is where an vote close to 0 says that the field has a 50 percent chance of outputting a correct classification and a positive vote tells us that there is some link between the stump and giving us a correct classification where the larger the number is, the better at classifying the data. A negative vote would therefore indicate that the stump mostly gets it wrong and therefore there may be an inverse relationship between the field and the correct classification.
9. Then, work out the new record weights of each record. If the record was incorrectly classified by the most successful stump, increase the record weight, if it was correctly classified by the stump, decrease the record weight.
    - to increase weight:
$$NewWeight = RecordWeight \times e^{Vote}$$

- to decrease weight:

$$NewWeight = RecordWeight \times e^{-Vote}$$

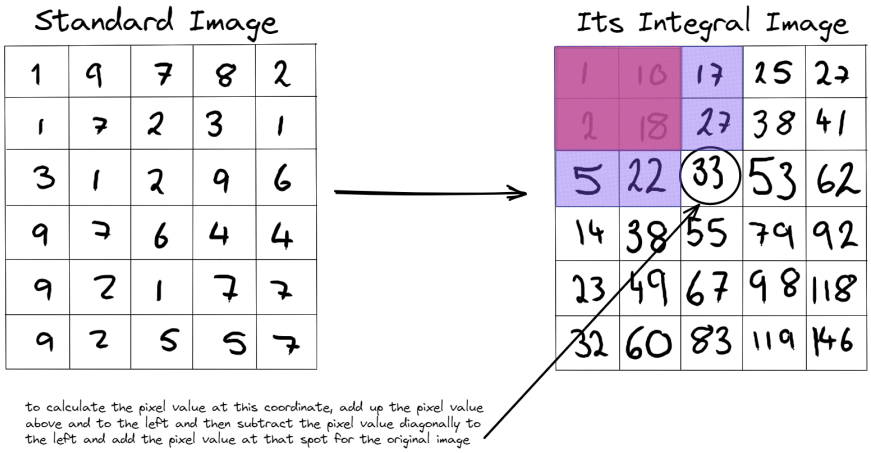
(above equations from [2])

10. Then replace the record weights with the new record weights and normalise them by dividing by the sum of all the new record weights. Then start the algorithm from the beginning to get a forest of weak learners that will classify the data.

11. The forest can be used by running all trees to get those that give a true value as their answer and those that give false, summing the amounts of say of the stumps in each category and choosing the answer that is supported by the largest total vote.

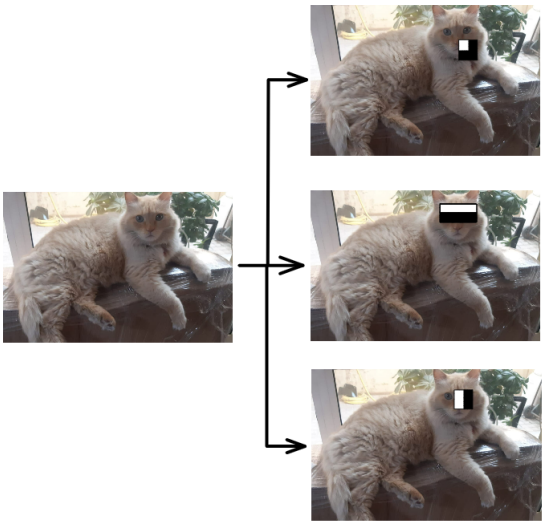
1.4.3 *Integral Image*

An integral image is a way of representing an image where each pixel is the sum of all of the pixels before it, this allows us to quickly and efficiently sum all of the values of the pixels in a block and subtract those that came before.

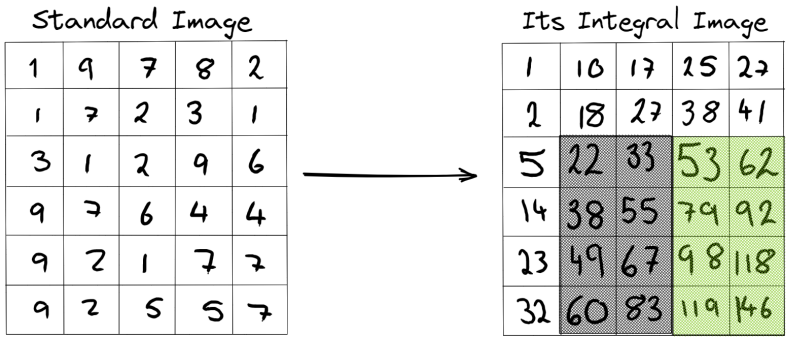


1.4.4 *Haar Features*

Haar features are a substitute to the edge detection kernels and consider the grading of the section compared to another for example a haar kernels could be:



Computing Haar Feature Extraction:
we can compute haar features in minimal calculations from an integral image rather than a standard image where you would have to perform many additions and subtractions.



to work out the intensity difference of the green section (high intensity section) and the black section (low intensity section) we do:

$$(83 - 32 - 27 + 2) - (146 - 83 - 41 + 27) = 23$$

so the difference between the sections is 23 and therefore this may represent a useful feature. We used 7 operations as opposed to 15 operations in the standard method without an integral image. The amount of operations for a comparison between 2 sections is constant but the amount of comparisons without an integral image scales with the box sizes

1.4.5 CUDA GPU Programming

In my project there will be a lot of image processing which means a lot of cycling through pixels or groups of pixels. This means that the time that it takes to run will escalate dramatically and,for example, multiplying kernels on a standard image iteratively has complexity $O\left(n^4\right)$. However, through the use of the Nvidia CUDA libraries, I can exploit GPU parallelism to make these operations run much faster.

1.4.6 GPU Architecture

The modern GPU is split into an array of streaming multiprocessors called a grid. Inside the streaming multiprocessors there are many streaming processors arranged in what is called a block. These streaming processors which we call threads in the programming design have low computational power in comparison to a CPU but there are a lot of them in each block so by splitting our computation into many small operations, we can run these in parallel and get a result much faster. The grids, blocks and threads must be considered while programming in various ways to optimise the program or to produce different results. Connected to the streaming multiprocessors is a section of shared memory called DRAM which is used to store data and in the streaming multiprocessors is a small section of shared memory called the L1 cache. When programming, we must copy the data from host memory on the CPU to device memory on the DRAM and back using the commands, CudaMalloc and CudaMemcpy.


GPUArchV2.jpg
[3]

1.4.7 CUDA Language and Structure

First, we start by programming in the host where we initialise all of the variables, set the device, allocate the memory on the GPU copying the data to the registers and set the grid, block and thread sizes. Then we program the kernel which is the code that will be run on all threads. If we are, for example, iterating through pixels in a picture, we assign a thread to each pixel or group of pixels and then locate the thread using the formula:

$$index = blockIdx.x \times blockDim.x + threadIdx.x$$

This takes the block index in the x direction and calculates the amount of individual threads by multiplying by the block size in the x direction and then adds on the current thread index in the block. The same formula is used in the y direction switching the 'x's for 'y's so $blockIdx.x \rightarrow blockIdx.y$.

1.4.8 EigenFaces

- **Principal component analysis**:

  A major issue when processing and trying to classify data from a large set of pictures is the high dimensionality of the data which makes it hard to classify data or cluster data. One method of reducing the dimensionality of a dataset whilst minimising data loss is PCA. PCA uses eigenvectors and eigenvalues extensively so the employment of PCA for facial recognition is commonly referred to as facial recognition using eigenfaces.

- **PCA steps:**

  1. vectorise the images so an $n \times n$ image turns into a $n^2 \times 1$ vector.
  2. Then store these in an $n^2 \times m$ matrix where m is the amount of images.
  3. Average all of the pixels to work out an average face
  4. Subtract the average face from all of the images which translates all of the data so the mean lies on the origin.
  5. Then we need to work out the covariance matrix which will allow us to get the eigenvectors and eigenvalues. To work out the covarience matrix, we transform our $n^2 \times m$ by its transpose so we do its transpose x the $n^2 \times m$ which gives us an $m \times m$ matrix.
  6. We then calculate the eigenvalues of all of the eigenvectors and pick the matrices with the highest eigenvalues
  7. We then store these vectors and represent the original training data as linear combinations of these vectors which we store in a weight vector where each weight corresponds to the weight applied to each of the vectors in the eigenspace when we are reconstructing the image using a weighted sum + average face.
  8. When we are trying to recognise test data, we go through the same steps as before, vectorising the image and normalising it by subtracting the mean. Then we calculate the weight vector and compare it to the weight vectors that we stored earlier by working out the difference. If the difference is below a certain threshold, the testing data is the same as that certain training data and we can classify it by the label of the training data.

- **evaluation of eigenfaces as a potential algorithm:**

  To make this work for facial detection, I need to use either a dataset of faces vs non-faces or generate a personalised dataset of faces to make a facial recognition algorithm for a certain number of people. A facial recognition algorithm would still be able to accomplish my goal as, after having been trained, it would be able to distinguish with more accuracy, certain faces from backgrounds however it would not be generalised to all faces. The best way of using eigenfaces is probably by generating average faces along with non-faces and then feeding this into the algorithm where the difference between a the input face and the average face will be less than the difference between a face and a plant for example. Another way of doing it is relying on thresholds so non-faces will not pass the threshold and therefore this will function as both a facial detection and facial recognition algorithm. In contrast, this would mean I will have to use higher thresholds which may hinder the facial recognition capabilities of this and lead to misclassifications of faces between each other.

1.4.9 Potential Data Structures

- **graphs**

  Trees will be important whether I decide to implement a neural network or follow the Viola Jones algorithm. For Viola Jones, I will use decision stumps which are trees composed of a root node connected to 2 nodes where the root node handles the computation and stores the results in the leaf nodes.

- **square arrays**

  Square arrays are vital as they allow me to do matrix multiplication efficiently with minimal memory usage as opposed to jagged arrays. They also allow you to copy and pass around data quickly rather than a jagged array which requires more computation copy. In reality, I will probably use a combination of these 2 data structures depending on the use case where, with a jagged array for example, I can quickly extract vectors/single dimensional arrays without need for iteration.

- **classes**

  I will need to use a class to preserve the matrix/square array datatype when passing data into my cuda library for matrix multiplication as there are no square arrays in c++. The class will hold the width of the square array and then hold the data in a 1 dimensional array. I will descend layers in a similar fashion to 2 dimensional arrays but will have to multiply (or divide depending on the situation) the index in the x direction by the layer to access different layers. This is more efficient than using a jagged array as it means I can loop through the array once when resetting it to a square array.

- **Abstract class**

  I will use an abstract to store the general properties of the haar feature kernels which will be their own separate classes that inherit from this abstract class, allowing me to store all of the kernels in an array.

1.5 Prototype

Project Critical Parts:

1. Grayscale conversion

   This is important for the algorithm as it simplifies the data so the amount of computation is decreased a lot and it is a lot simpler to do

2. Image scaling

   From my interview, i know that outputting as many visual representations of the data is important so I will use image scaling to show the user which image and what part of the image the model classifies as a face

3. Conversion to Integral Image

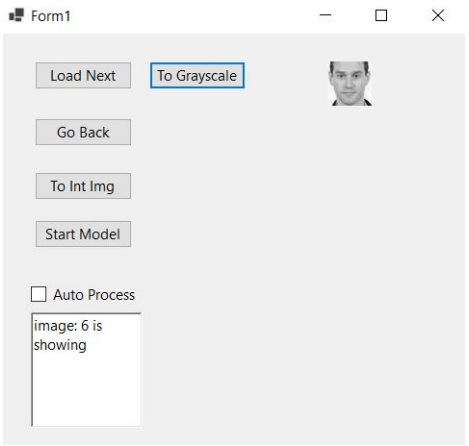
   Without integral images, the program would be a lot slower as there would be far more computation involved.

4. Finding Thresholds for Kernels

   This is important as it allows the kernels to make better classifications than just with some arbitrary threshold.

Implementation

1. Grayscale conversion

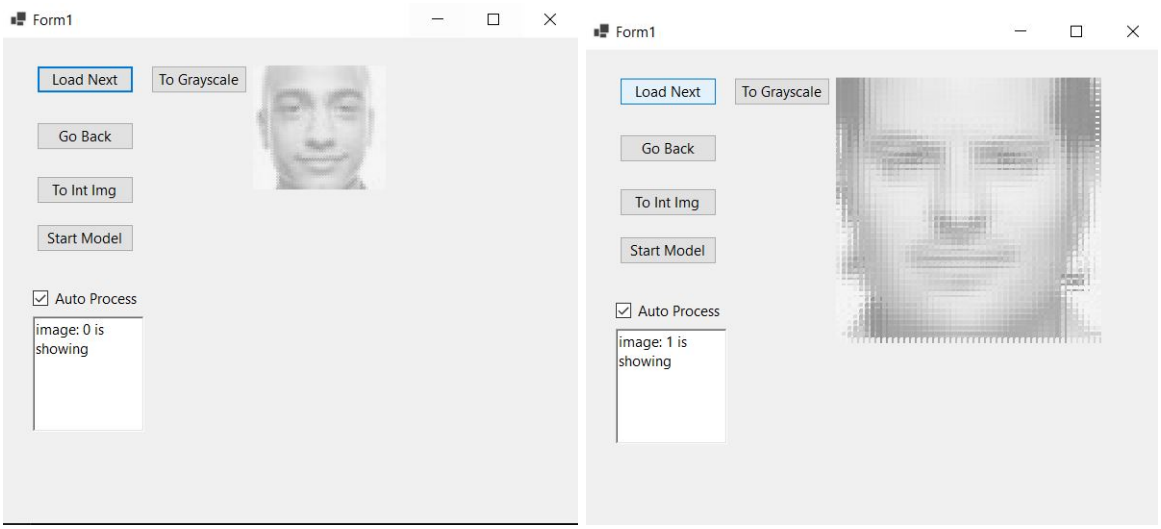



```
0 references
public void imageToGrayscale(Bitmap imageIn)
{
    for (int i = 0; i < imageIn.Width; i++)
    {
        for (int j = 0; j < imageIn.Height; j++)
        {
            //works out the average colour and then replaces the red green and
            //blue values with this so that there is no colour, only intensity
            Color col = imageIn.GetPixel(i, j);
            int avgColour = (col.R + col.G + col.B) / 3;
            imageIn.SetPixel(i, j, Color.FromArgb(col.A, avgColour, avgColour, avgColour));
        }
    }
}
```

   To convert an image to grayscale, I calculate an average value for the RGB values of each pixel and then change the colour values to be the average, preserving the alpha value.

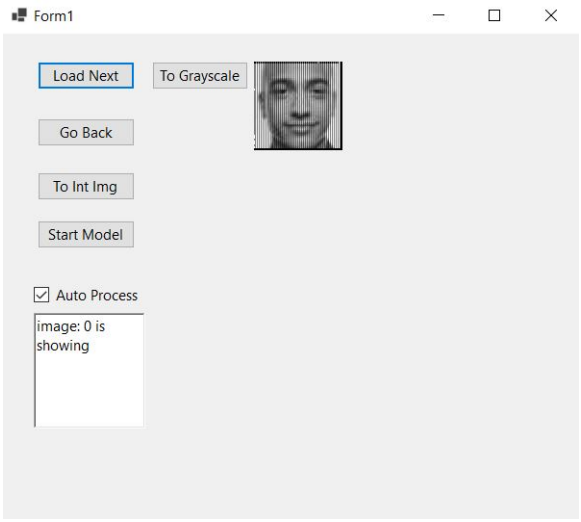
2. Image scaling

   - Nearest neighbor



nearest neighbor is an image scaling technique where we find the closest pixel on the original image to the pixel on the new image. In my program, I do this by fist creating the new image and then looping through the new image. For each pixel in the new image, I work out the local index which is the index between 0 and the scale factor. This local index is less than the scale factor as the scale factor is the distance between each pixel. If the local index is more than half of the scale factor then the pixel is closest to the next original pixel. We then do the same for the Y direction.

- Bicubic interpolation



Bicubic interpolation is an image scaling technique that creates an equation for the intensities of the pixels and then uses these equations to predict the new pixels. To do this, I used matrix methods which allowed me to solve a system of 4 simultaneous equations. 2 of the simultaneous equations considered the gradients and 2 considered the intensities. To solve this system, rather than using inverse matrices which would have been slow, I used gaussian elimination to simplify the matrices and then solve them where the last row should be simple enough to solve for the last coefficient. This can be subbed into the row above and then used to find the above coefficients.

3. Conversion to Integral Image

This is vital step for the program as it allows me to significantly reduce the number of calculations required when calculating the value of a kernel. Rather than adding up every pixel in the kernel, through integral images, I am able to just add up 3 numbers and subtract 1. To do this, I have to add in 1 more row and 1 more column of 0s to the left and top of the image. This means that I can easily move around the image in 1 go.

4. Finding Thresholds for Kernels

This is also a vital step for the program as, without it, I wouldn't be able to give classifications from the outputs of the kernels.

```
public double[][] getThresholds(ref List<windowTrainer> Data,
int amtWindows, imageProcessorClass methods)
    {
        List<double>[] thresholds = new List<double>[amtFeatures];
        Decimal[][] giniImpurity = new decimal[amtFeatures][];

        double[][] ValuesPerKernel = new double[amtFeatures][];
        for (int i = 0; i < amtFeatures; i++)
        {
            ValuesPerKernel[i] = new double[amtWindows];
            giniImpurity[i] = new decimal[amtWindows - 1];
            thresholds[i] = new List<double>();
        }

        int yesAfter = 0;
        int noAfter = 0;
        Console.WriteLine("threshold variables instantiated");
        for (int i = 0; i < Data.Count; i++)
        {
            double[] kernelVals = Data[i].getKernelValues();
            for (int j = 0; j < amtFeatures; j++)
            {
                ValuesPerKernel[j][i] = kernelVals[j];
            }

            //tallying total yes(s) and no(s) in this loop so that
            //i can save time looping later
            if (Data[i].isFace)
            {
                yesAfter++;
            }
            else
            {
                noAfter++;
            }
        }

        Console.WriteLine("line 297: yes(s) and no(s) tallied");

        for(int k = 0; k < amtFeatures; k++)
        {
            ValuesPerKernel[k] = quickSort(ValuesPerKernel[k]);
        }


        for (int i = 0; i < amtFeatures; i++)
        {
            int yesAfterTemp = yesAfter;
            int noAfterTemp = noAfter;
            int yes = 0;
            int no = 0;
            for (int j = 0; j < amtWindows - 1; j++)
            {
                thresholds[i].Add((ValuesPerKernel[i][j + 1] +
                ValuesPerKernel[i][j]) / 2);


                int indexOfData = methods.BinarySearch(ValuesPerKernel[i],
                Data[j].internalKernelValues[i]);
                if (Data[indexOfData].isFace)
                {
```

```
                    yes++;
                    yesAfterTemp--;
                }
                else
                {
                    no++;
                    noAfterTemp--;
                }

                double impurityBefore = 1 - Math.Pow(yes / (yes + no), 2) -
                Math.Pow(no / (yes + no), 2);

                double impurityAfter=1-Math.Pow(yesAfterTemp/(yesAfterTemp+noAfterTemp),2)
                - Math.Pow(noAfterTemp / (yesAfterTemp + noAfterTemp), 2);

                Decimal before = Decimal.Multiply(Decimal.Divide((yes+no),
                yes+no+yesAfter+noAfter), (Decimal)impurityBefore);

                Decimal after = Decimal.Multiply(Decimal.Divide((yesAfter+noAfter),
                yes+no+yesAfter+noAfter), (Decimal)impurityAfter);

                giniImpurity[i][j] = before + after;
            }
            Decimal min = Decimal.MaxValue;
            int index = 0;
            for (int j = 0; j < giniImpurity[i].Length; j++)
            {
                int temp = Decimal.Compare(giniImpurity[i][j], min);
                if (temp < 0)
                {
                    min = giniImpurity[i][j];
                    index = j;
                }
            }
            features[i].threshold = thresholds[i][index];
        }
        return ValuesPerKernel;
    }
```

1.6 Second Interview

aims
From my second interview I hope to get guidance on how to improve from my prototype from points to research further and then what I should improve on for my technical solution.
interview questions

1. What was successful in the prototype?
2. What could use improvement for the technical solution?
3. What type of data should I display?
4. How should I organise kernel locations?
5. What is the best way to debug my program?
6. How many types of kernels will I need?
7. What features are unnecessary?

Interview answers

1. **What was successful in the prototype?**

   I think that even though the image scaling for bicubic interpolation resulted in a few lines appearing on the screen, we can still use this in the final solution as a face is still going to produce a different output to a car if you know what I mean. Not everything has to be perfect and if you are worried about it and can't find the bug then just use the nearest neighbor which is perfect for the purposes of the algorithm.

2. **What could use improvement for the technical solution?**

   Every item apart from the image scaling works perfectly already, I would just suggest working on the scaling to be honest, it doesn't have to be perfect and it is fine as it is but would just be more pleasing to the eye if you could fix it up a bit.

3. **What type of data should I display?**

   As much as you can. Display as much as you can so that you know what you are looking at when you are debugging or tweaking the project.

   Any examples?

   You are already showing us what the faces look like so I would say, display the stumps and their associated thresholds and accuracies

4. **How should I organise kernel locations?**

   Maybe use a window and then place a bunch of kernels where you think they will work best. It doesn't really matter where they are because adaboost will take care of the ones that don't do much but obviously, a bunch of high performing kernels will perform better than a bunch of bad ones, adaboost or no adaboost. Then just store these in an array with their vertices' coordinates for ease.

5. **What is the best way to debug my program?**

   Re-read the code a lot and make sure to check that every calculation outputs the right values. A good way is to start at a high level and zero-in on anything you think is weird.

6. **How many types of kernels will I need?**

   You can do a lot with a little, just use the square and rectangle features, no need to do lines and stuff. You can combine edge features to find lines.

7. **What features are unnecessary?**

   Line features, point features, anything complex really, no need for diagonals and so on.

1.7 Further Research

1.7.1 Image Scaling

From my interview and prototype I have realised the importance of scaling of images and normalising the data. I need to ensure that all of my images are of about the same size so that I can use the same kernels on them reducing the chance of overfitting.

- **Nearest neighbour**

  The nearest neighbour algorithm is a potential approach. The advantage of this algorithm is that we wont be adding any data to the picture so it preserves the data and edges which would be lost with other approaches in large scale factor scaling. Also, this algorithm is very fast as there are little calculations involved, this means it wont impact the already relatively slow run-time of the whole application. The disadvantage of this is that when scaling, it will result in some pixellation which will hinder the accuracy of the kernels when detecting edges if the pixels are too large.

- **Bicubic interpolation**

  This is another approach which seeks to reduce the pixellation of the scaled up image by working out the colour of the new pixels based on the distance between the original pixels around it. This is an improved version of bilinear interpolation which only takes an average of the product of the distances between the pixels and the pixel values of the original image. The advantage of using such an algorithm is that it may be able to increase the size of the image without losing the edges which are significantly blurred in bilinear interpolation and without making the image too pixellated. However, the downside to this approach is the increased complexity and the increased impact on the run-time of the application.

In conclusion, I will attempt to use a combination of both as I only have nearest neighbour in my prototype so far which handles small scale factor increases well but, I think, for large scale factor increases it is necessary to use bicubic interpolation.

Nearest Neighbour

Nearest neighbour steps:

1. find scale factor to fit whatever new container you want to put it in
2. make a new blank image of the old image's dimensions multiplied by the scale factor
3. iterate through the new image
4. divide the index of each pixel by the scale factor to get where it would have been on the old image
5. subtract the difference between the pixel to the right and to the left. Store the x-index of the old pixel that has the smallest difference
6. subtract the difference between the pixel above and below. Store the y-index of the old pixel that has the smallest difference.
7. then set the new pixel colour to be that of the pixel that is at the indexes that we calculated in the last two steps.
8. then iterate to the next pixel and repeat. This works for all pixels including those that map directly to the new image as they will have the smallest difference.

Bicubic interpolation

1. Find your scale factor and round it to the nearest natural number.
2. Get the gradient and colour values for the pixels and their neighbours in the X direction.
3. Use the differentiated and standard cubic equation to generate a system of 4 simultaneous equations with 4 unknowns
4. Use Gaussian elimination to solve these equations, the d and b coefficients in the right hand side vector and the coefficient matrix need to be swapped so that we can make this into row echelon form.
5. Use these Generated values to predict missing pixel values based on the X gradients.
6. Store predicted values in a predicted X matrix.
7. Do all of these steps again but in the Y direction and store it in a predicted Y matrix
8. Average the matrices and convert this back into an image.

1.7.2 Creating Stumps

One element I had overlooked in my initial research was the method in which I would get the threshold values for my decision stumps which is the first preliminary step that facilitates the whole algorithm.
To get my threshold values, I will have to use Gini Impurities to work out which threshold value is the best at separating positive classifications from negative ones.

1. First work out the mid-points between the kernel values returned by each feature.
2. Then tally the amount of true and false classifications before and after the mid-point in question.
3. Then plug the amount of true and false classifications into the equation:

$$impurity = 1 - \left(\frac{true}{true + false}\right)^2 - \left(\frac{false}{true + false}\right)^2$$

4. The gini impurity is the weighted average of the impurity before and after:

$$gini\ \ impurity = \frac{classifications\ \ before}{total\ \ classifications} + \frac{classifications\ \ after}{total\ \ classifications}$$

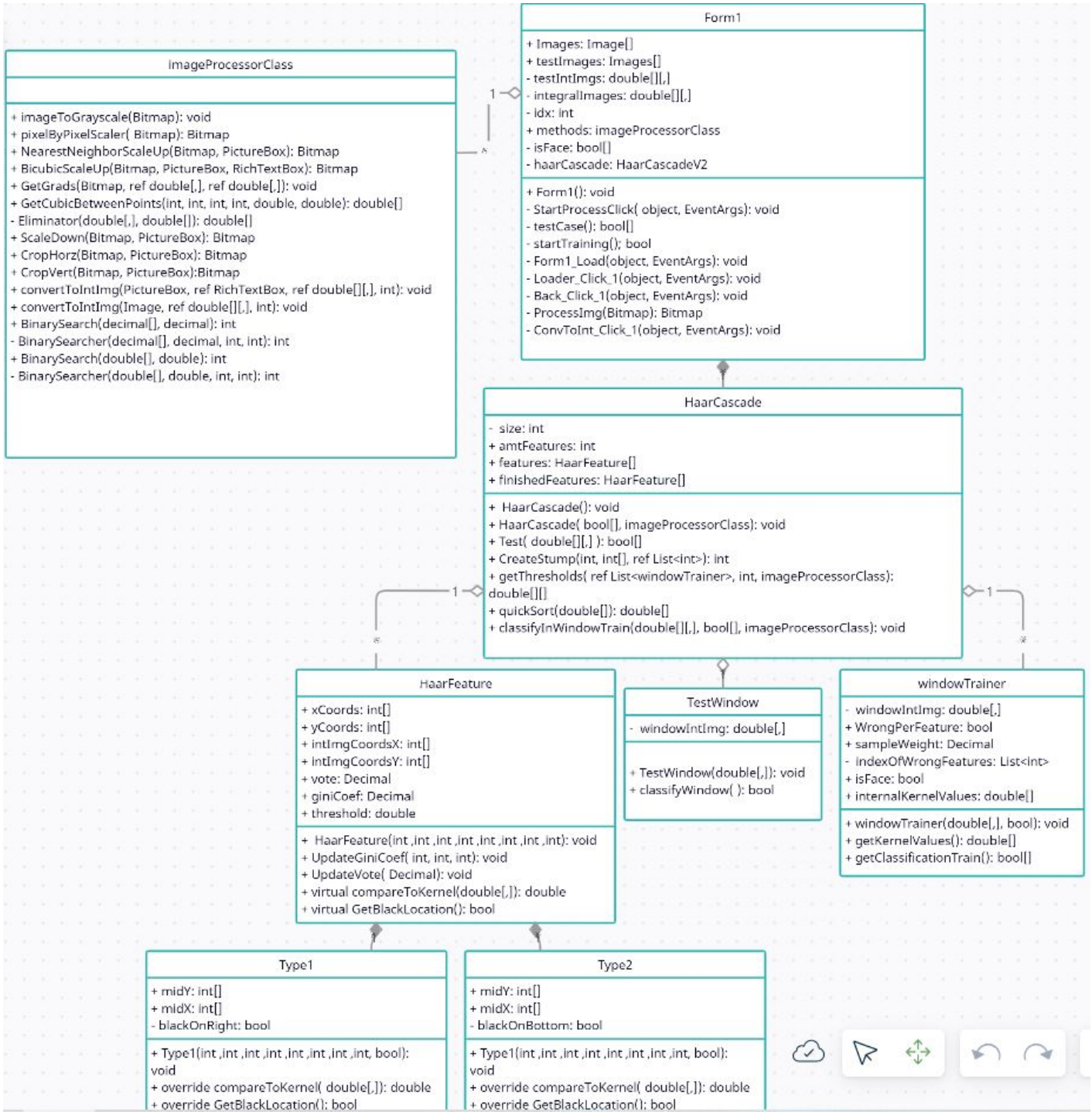
5. Then choose the mid-point with the lowest gini impurity to be the threshold value.

1.8 Objectives

1. The user should be able to choose to train the model or run a previous version of the model.
    (a) Thresholds should be stored in a text file so that the end model can read from this and classify data without having to be trained again.
    (b) at least 2 versions should be backed up and then the system overwrites the oldest version every time.
2. The user should be able to input a picture
    (a) The user should should be able to choose their image in file explorer by clicking on a button
    (b) The picture will be converted to a bitmap
    (c) Image will be converted to grayscale
    (d) Image will be scaled to that they are all of similar dimensions
        i. image will be scaled with nearest neighbour which will preserve the data and wont add anything in terms of smoothing or blurring.
        ii. The images will not be stretched or squashed to be square but the middlemost section of the image will be used
3. The program should use no 3rd party software/programs.
    (a) I will build my own model
        i. Custom Haar Feature Cascade
            A. Abstract class of haar features
            B. Boolean to store the orientation of kernels
            C. features will be of any dimensions either rectangle or square
            D. features will be divided either horizontally or vertically
        ii. Conversion to integral image
        iii. AdaBoosting
            A. Has a class for a decision stump
            B. Should be able to work out the best thresholds for each stump
            C. Has a train subroutine that iterates through a dataset
            D. Has a test function that returns a binary classification of the object.
4. It must output graphical representations of the data and pictures
    (a) Use a scatter graph to show the clusters of data and how the algorithm separates faces and not faces in both the training and test data. This will be a graph with axes: total vote for and total vote against being a face.
    (b) Use a bar chart of frequency against time to show how the algorithm performs on different data and calculate a mean time to process and classify 1 image.
    (c) Plot a line graph of size vs time for the matrix methods on different dimension matrices.
    (d) Display the processes applied to the image in a picture box.
    (e) faces picked out of the image must be output at a larger size than the kernel

1.9 Modelling Of Problem

1.9.1 class diagram for project:



1.10 simultaneous equation solving with matrices

to solve simultaneous equations on a computer, we need to either find the inverse of the matrix and multiply or use gaussian elimination to reduce the equations to row echelon form from which they can be solved procedurally.

1.10.1 Gaussian elimination

Steps for gaussian elimination:

1. Set up the simultaneous equations in an augmented matrix:

$$
\begin{aligned}
A: 2w + 5x + 4y + 3z &= 10\\
B: w + 5y + 6z &= 5\\
C: 8w + 5x + 2y + 3z &= 2\\
D: 4w + x + 6z &= 8
\end{aligned}
$$

$$
\Rightarrow \left(\begin{array}{cccc|c} 2 & 5 & 4 & 3 & 10 \\ 1 & 0 & 5 & 6 & 5 \\ 8 & 5 & 2 & 3 & 2 \\ 4 & 1 & 0 & 6 & 8 \end{array}\right)
$$

2. Switch the rows so that there is a non-zero value in every slot diagonally:

$$
\Rightarrow \begin{array}{c} R_1 \\ R_2 \\ R_3 \\ R_4 \end{array} \left(\begin{array}{cccc|c} 2 & 5 & 4 & 3 & 10 \\ 8 & 5 & 2 & 3 & 2 \\ 1 & 0 & 5 & 6 & 5 \\ 4 & 1 & 0 & 6 & 8 \end{array}\right)
$$

3. divide each value in the row by the 1st entry:

$$
\text{Want} \left(\begin{array}{cccc|c} 1 & a & b & c & g \\ 0 & 1 & d & e & h \\ 0 & 0 & 1 & f & i \\ 0 & 0 & 0 & 1 & j \end{array}\right)
$$

So we divide R_1 by 2

$$
\left(\begin{array}{cccc|c} 1 & 5/2 & 2 & 3/2 & 5 \\ 8 & 5 & 2 & 3 & 2 \\ 1 & 0 & 5 & 6 & 5 \\ 4 & 1 & 0 & 6 & 8 \end{array}\right)
$$

4. Then subtract each row by the scaled values in the 1st row so that the 1st entry in each row is 0:

Now subtract all rows by R_1

$$
\left(\begin{array}{cccc|c} 1 & 5/2 & 2 & 3/2 & 5 \\ 0 & -15 & -14 & -9 & -38 \\ 0 & -5/2 & 3 & 9/2 & 0 \\ 0 & -9 & -8 & 0 & -12 \end{array}\right)
$$

2 design

3 Testing

4 Evaluation

5 Technical Solution

6 References

References

[1] Paul Viola and Michael Jones' paper: Rapid Object Detection Using A Boosted Cascade Of Simple Features

[2] Definition of stump and equations for adaboost (which are standard accross all applications of this algorithm) taken from Josh Starmer's video on AdaBoost, used to learn the steps for AdaBoost:

https://www.youtube.com/watch?v=LsK-xG1cLYA

[3] https://www.researchgate.net/figure/High-level-overview-of-a-CUDA-GPU-architecture_fig2_274653719

[4] Site used to learn adaboost:

https://medium.com/@sampathtunuguntla13/math-behind-adaboost-algorithm-in-3-steps-477745399553