# Facial Detection in Color Images

## CPSC 4310 – Image Processing

**William Beninger**

**2015-04-18**

# Introduction

## Abstract

Facial detection is a feature that is readily implemented in most cameras and is useful for a variety of tasks. Once a face has been identified, it may be useful to focus a picture on that focal point or to perform additional processing. At its simplest, facial detection is the process by which an algorithm seeks to identify where human faces may exist within an image. Most software packages, including OpenCV, include pre-built solutions for this task and have an excellent accuracy with minimal false positives. In this paper, I will be describing a process for facial detection that I have implemented in C++ based on several published papers. I will discuss the solution that this paper describes and the method to which I have implemented it. Further, I will step through the processes and provide intermediate data to illustrate exactly what my program is doing to achieve its end goal of detecting faces. In this process, I will discuss the advantages of my program as well as what improvements could be made to increase accuracy and precision. Finally, I will include my experimental data and reveal the results of exactly how my testing was done.

## Motivation

This project was of immediate interest to me when I had made my decision to register for this class. As I reviewed the course material, I realized that it fell slightly out of scope and took it upon myself to learn more about it for this project. Originally in my research I looked into quite a few avenues for facial detection but the majority required additional libraries (*eigenfaces/eigenvectors/openCV/etc*) and I elected to pursue a route that involved only the STL libraries and NetPBM. My belief was that I would learn the most from the process and I would have a deep understanding of the subject matter. Speaking now that I have finished the project, I can confirm this original feeling as I have a much greater appreciation for these algorithms and the code that goes into implementing them.

Beyond simply a deeper understanding of the subject matter, facial detection is of particular interest to me as it involves teaching a computer to do steps in learned identification that humans take for granted. Humans are extraordinarily gifted at identifying faces, so much so that we often experience pareidolia; or rather the phenomenon where we believe we see a human face in something entirely insignificant. This learned behavior is hardwired in and the idea of teaching some of this methodology to a machine is extremely interesting. I believe that this type of knowledge transference can be extended and is an extremely valuable attribute to possess.

## Problem

How do you find a proverbial needle in a haystack? Finding a human face in an image is a very difficult task to attempt to transfer over to a machine. Images are themselves composed of a variety of individual points arranged in a grid format with each point containing a mixture of three colors. In itself, there is nothing extremely obvious to tell a machine to look for and we must instead abstract upon the concept of how exactly to identify a human face.

Some papers elect to pursue the route that involves identifying shapes that appear to generally take on that of a human head and attempt to further refine afterwards while others attempt to find features that commonly exist within a face such as the placement of eyes, nose, and mouth in relation to each other.  Unfortunately, both of these attempts can very quickly throw out a lot of potentially useful data and is extremely complex to implement. Both methodology requires that the face exist in such a way that is matches a preset number of shapes. If the head is turned providing a side profile or if the face is partially obscured, this approach fails entirely. As well, lighting conditions may influence algorithms that attempt to define lines to create shapes causing even more data to be thrown out. Figure 1 attempts to illustrate this concept further.

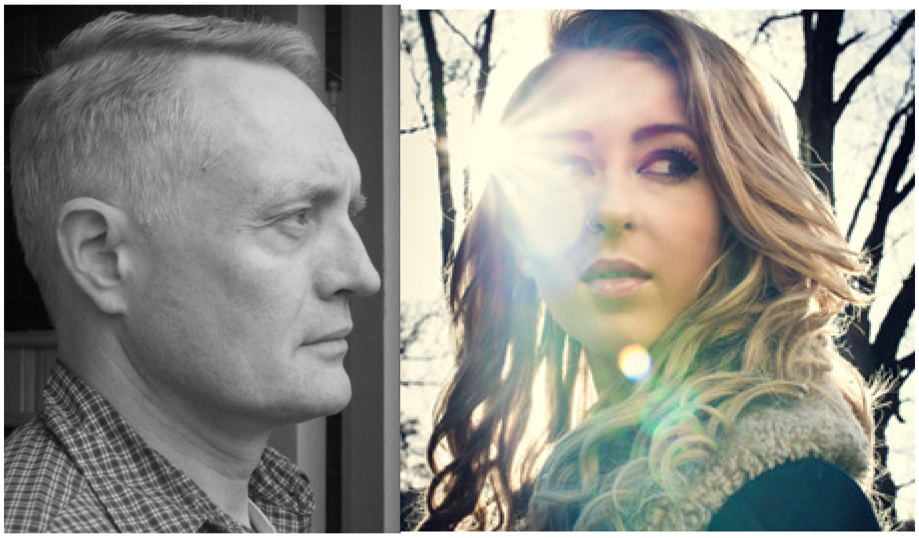


**Figure 1:** *A side profile picture paired with a drastic lighting change to illustrate the problem.*

This idea of casting a very wide net and attempting to further refine is excellent in theory but if this is our intention, we must be sure that this initial action is computationally insignificant and does in fact improve our accuracy. So the question returns to, what is one of the simplest things that a machine can do with the given data in its present format to quickly return a localization of where a face may be in an image?

## Proposed Solution

The solution that I have studied for this project has attempted to propose a different way of localizing and detecting faces in images that is computationally insignificant but still provides a high degree of accuracy. The paper that I have targeted for my project was co-authored by Hongming Zhang and Debin Zhao from the Harbin Institute of Technology. Zhang et al. have put forward the idea of using skin color as well as learned skin pattern matching in order to better localize regions of an image to further process and look for faces. Spatial pattern matching is achieved through spatial histograms and further refinement is done through the usage of Support Vector Machine, both of which will be discussed later in this paper.

## My Implementation

I have created a C++ program using the NetPBM library that we have been using in class in addition to the standard template libraries in order to implement my solution. No other exterior libraries are used in the solution as it currently exists. All compiling has been done on g++ 4.9.1 on a Ubuntu machine. The machine specifics are fairly insignificant as the operations that will be described here are not intensive and will not strain even the most budget machine.

The program itself from a very high level begins and requests the program operator to select whether to create a new database of pattern features based on files that exist within a database folder or to load from a previously saved value. These images are used to create a standard template for later histogram matching and the resulting template is saved in a text file automatically for later usage. Afterwards, the program operator is prompted to input a .ppm image file to be processed and the operator is prompted again for a destination file to print out the same image with each of the faces highlighted in a white square. Overlapping squares of the same size are removed but different sized squares are kept as they may provide indications into why a particular area was selected.

All database sample pictures have been taken from the publicly available headshots on the dating website http://www.plentyoffish.com/. They have been cropped and sized on a case by case basis and were selected based on photo quality and apparent lack for any additional photo filter being used. In total there were 50 face pictures used to create the database with 25 males and 25 females. For each gender, I attempted to roughly collect 25% Caucasian, 25% African American, 25% Asian, and 25% Indian faces in order to have the color distribution to be fairly evenly distributed. Additional features which I have considered are glasses and facial hair which are also included in my fairly small sample size. See Figure 2 for an example of my database.

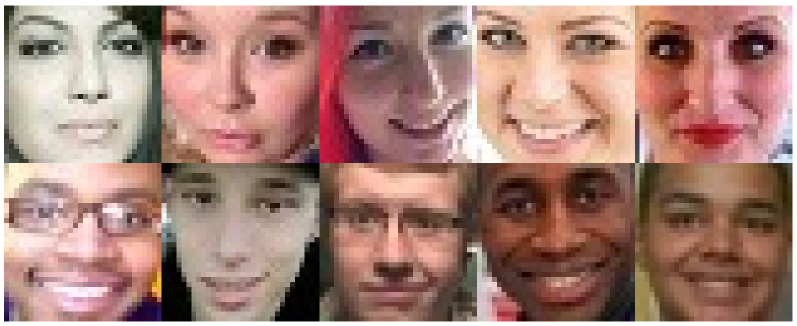


**Figure 2:** *Examples from my facial database with 5 female faces and 5 male faces.*

All experimental test data has also been pulled from http://www.google.com image search and consists of a variety of group pictures, portrait photos, and a variety of additional images without human faces to test for false positives. For my experiments, I have run the program on the unmodified version of the photo and then again on an "optimized" version of the photo which attempts to put the faces in a scale to which the program operates best. I will be sharing my results with both and explaining the reasoning as to why this is warranted and what shortcomings my program may have as a result later in my paper.

## Detailed Description of Solution

### High Level Approach

From a very high level, this implementation attempts to use a series of training data to compare against certain spatial segments of an image. In order to create both the training data and the rendering the image into the correct format, we must first apply a variety of operations to the image itself. Despite having not implemented the Support Vector Machine portion of the program that prerequisites a good deal of this data, I have performed all of the necessary calculations so that it could be easily added into the program if this step was necessary. The support vector machine is, from my understanding and research, a methodology of attempting to group similar datasets in a multidimensional meaningful with a clear distinction to identify what is not considered a part of this dataset. This is accomplished through the usage of supervised learning where the machine is told what data is to be considered similar and what data is to be considered as negative. From this learning, the machine is able to plot future data and determine its likelihood to be a part of a particular group. In this program, all of the data is in a format that would be readily accepted by an SVM machine but it carried the prerequisite of including an additional library and was deemed to be out of scope.

For the proceeding sections, the steps necessary to render an image or a portion of an image into this necessary format will be described individually with a final description of how all of the pieces fit together.

### Introduction of YUV Color Space

The first step of this process involves taking the image in its present Red/Green/Blue color space and converting the values into the YUV space. YUV color space is composed of Y which carries the luminance information of a pixel (essentially brightness) and two color components UV. This color space is what is considered to be the most similar to what human perception of color actually is and so is used when necessity of viewing color in this way is required. Further, in this color space the hue is defined as the angle $\theta$ of U and V.

There are a variety of conversion formulas for a variety of different YUV formats however the one that was implemented in this paper is the SDTV BT.601 version and the formula exist as described in figure 3.

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ U &= -0.147R - 0.289G + 0.436B \\ V &= 0.615R - 0.515G - 0.100B \end{aligned}$$

$$\theta = \arctan(\frac{V}{U})$$

**Figure 3:** *Formulas that describe the RGB->Conversion*

The completed table of measurements consists of Y, R, G, B, and $\theta$. All are used in the following sections in order to render the image into the appropriate format however they are all operated on independently.

## Histogram Equalization

In this step, each individual measurement is equalized via histogram. In the effort for completeness, a histogram is a measure of an image or a segment of an image which is entirely a 1 dimensional sum of how many of each value exists. In the case of the measurements that we are working with, a histogram would exist as 4 separate arrays of size 256 and a final array of size 360 for the angle measurement. It is calculated simply by iterating through the image capturing the recorded value at each so it operates in O(NxM) time with N being height and M being width.

Histogram equalization is a method through which each of these sums is divided into a probability by dividing the total sum by the number of pixels in the image. Afterwards, each intensity probability is then multiplied by the largest value and added to the proceeding intensity value. The formula implemented in my solution is taken from Digital Image Processing by Gonzalez. (Figure 4)

$$\begin{aligned} s_k = T(r_k) &= (L-1)\sum_{j=0}^{k} p_r(r_j) \\ &= \frac{(L-1)}{MN}\sum_{j=0}^{k} n_j \qquad k = 0, 1, 2, \ldots, L-1 \end{aligned}$$

**Figure 4:** *Formula showing the implementation of histogram equalization*

This methodology results in an image that attempts to have the colors all around the same level. The before and after effects of applying this process on an image on the individualized RGB components is displayed here in figure 5. The in detail histograms of each measurement are displayed below in Figure 6.

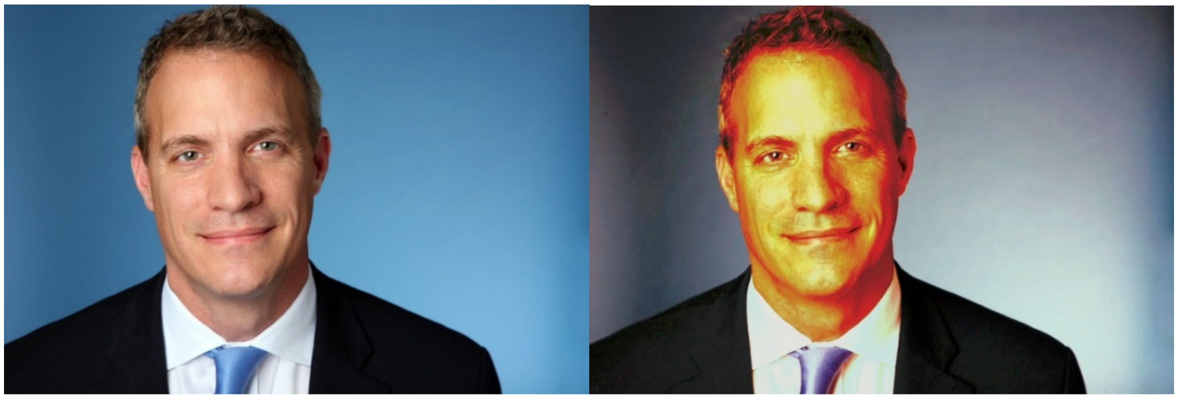
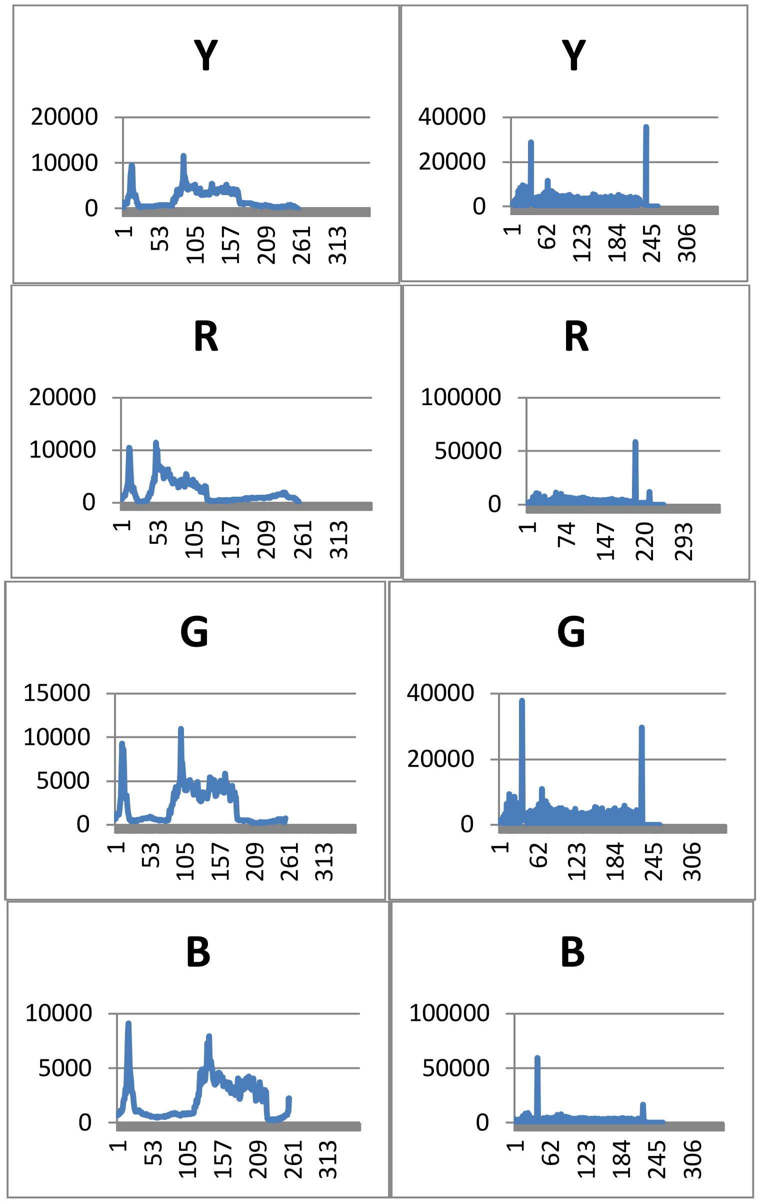


**Figure 5:** *Results of applying histogram equalization across the RGB spectrum on each measurement independently.*



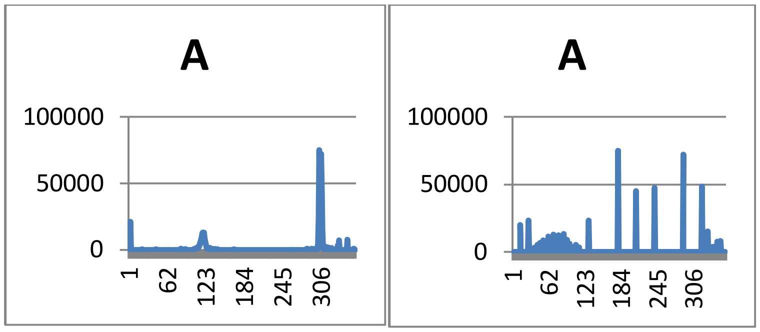


**Figure 6:** *Charted histogram results of the five measurements with before on the left and after on the right*

## Local Binary Pattern (LBP) Operation

This step is borrowed from another paper that makes suggestions on rotationally invariant texture classifications authored by Pietikainen et al. In this classification, a 3x3 neighborhood is shifted around the image performing a thresholding operation with the middle pixel on all surrounding pixels. With this binary value, they are multiplied by a set of preset values for each neighbor with each preset value being a power of two. The resulting neighborhood is then summed to produce a single value which is stored in that pixel and the neighborhood moves on. Figure 7, taken from the paper, illustrates this concept with a neighborhood of sample values.

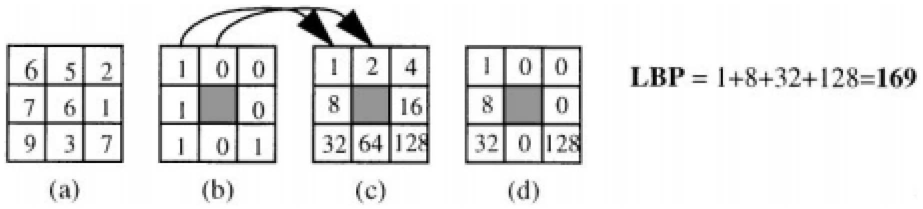


**Figure 7:** *Example of the LBP operation being performed on a 3x3 neighborhood.*

In the above example, we can see that the middle value of 6 is used to threshold all of the values in the neighborhood and those greater or equal to the value are given a binary value of 1 while those less than this threshold values are given 0. The binary values are multiplied by the powers of 2 which cascade from $2^0$ in the top left to $2^7$ in the bottom right. The summed value is placed in the pixel and we are given a pattern format. A visualization of what the process does to the RGB color space is given below in Figure 8. A charted histogram analysis of each converted measurement is presented in Figure 9.

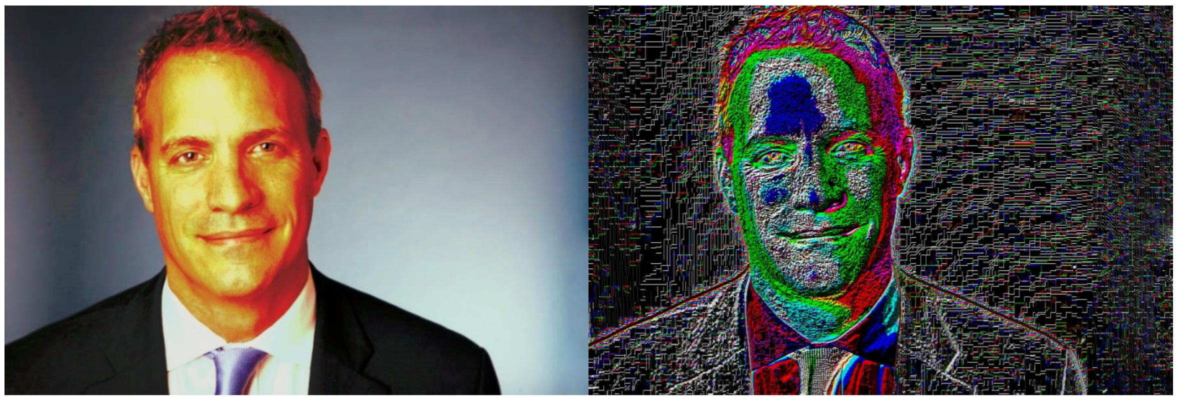


**Figure 8:** *Visualization of an equalized image being converted into the LBP space.*

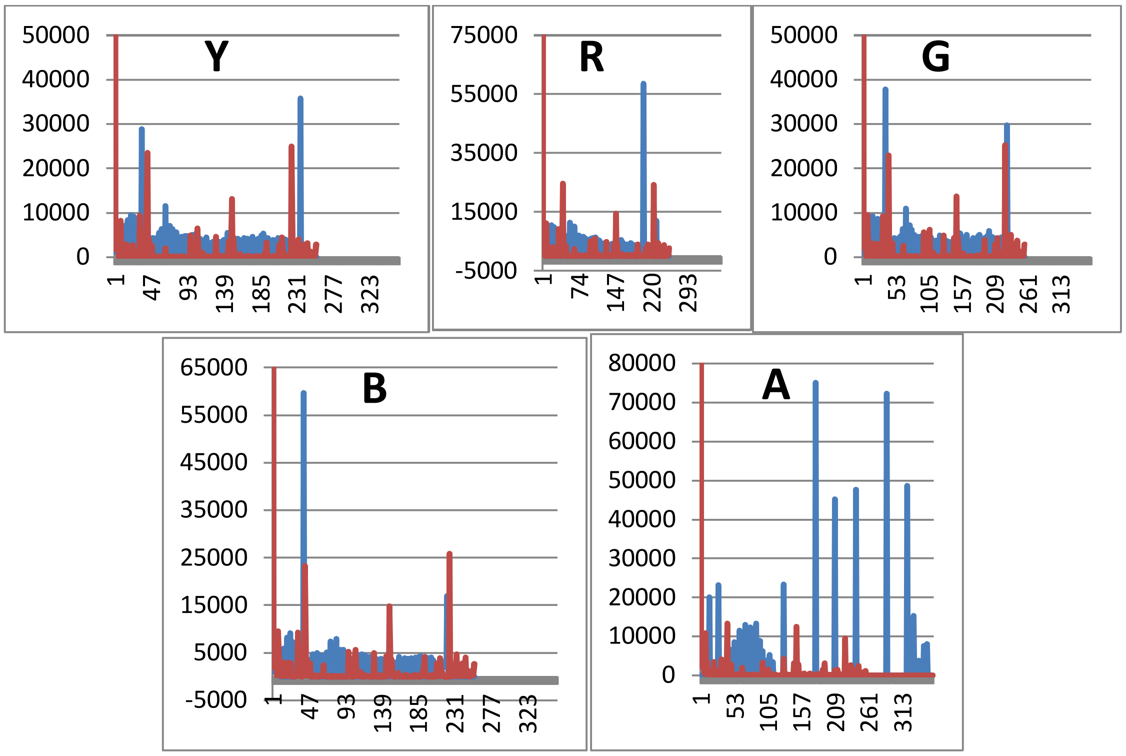


**Figure 9:** *Histogram representation of the change with original being represented in blue and the change in red*

## Spatial Histograms

Thus far we've discussed histograms "*ad nauseum*", however these histograms are incredibly limited by their 1 dimensional descriptive ability. Prior to explaining the next step in this process, it is important to understand how Zhang et al. have attempted to add a spatially defining feature to this. Through the usage of specialized spatial masks, the captured areas are further defined and patterns of textures must conform more exactly to the training data set of facial histogram features. These masks are placed on the image and slid around to collect a more specialized histogram that will later be used for matching. In their paper, Zhang and Zhao have shown 23 spatial templates that have the potential to preserve facial shape features however only actually implement three. For the full list of templates, please see Figure 10.

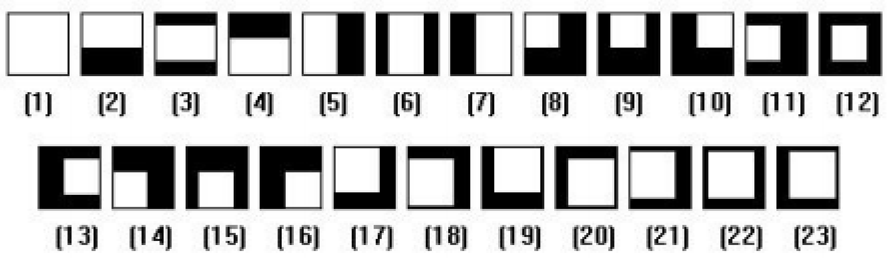


**Figure 10:** *Full listing of 23 spatial templates recommended by the paper.*

However, during the researcher's implementation they only made use of templates 1, 2, and 6. In my personal experimentation I found that 1, 3, and 6 to be the most effecting spatial templates however that will be explained further in my experimentation. These masks are binary with white being 1 and black being 0. Each mask is 32x32 in size and is directly multiplied with the existing pixel values beneath them.

## Training Data

In order to proceed any further, we need to take a side track in order to create the next element in this paper. Before we are able to compare the initial image against anything we must first create the training set of data which is supposed to hold the standard for what a face is actually supposed to look like. In the case of the training data, I make use of the cropped faces of size 32x32 and apply all of the above steps individually to each image including the spatial templates to form spatial histograms for each image. At this point, each set of training data image contains 5 measurements with 3 spatial histograms for each which means that each training image provides 15 dimensional plotted data for what an image should look like. At this point however, I should mention that for the purposes of the implemented paper that only the 1st measurement of Y is actually going to be used as the researchers determined it was precise enough and through my own trials of the various measurements I have found that it is more precise than the others. This leads to each image posting only 3 dimensional data points but for the immediate purposes this is plenty.

With 50 images in my training database and 3 dimensions each we are left with the task of combining all of these images into 3 averaged spatial histograms. The research paper makes the suggestion of calculating the sum of each histogram being that adding the first intensity value from all images with the same spatial template and dividing the resulting sum by the number of images for each intensity value in the histogram. The resulting histograms look like Figure 11.

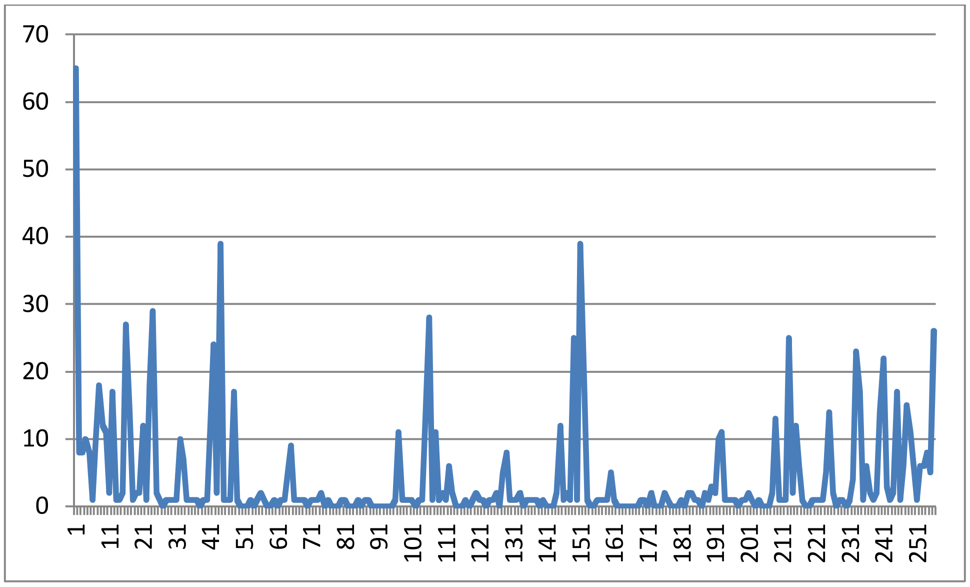


**Figure 11:** *Histogram showing the completed merged database of training data for measurement 1 and the first spatial template.*

## Histogram Differences

With everything above, we have three training database sets of spatial histograms and we have an image that is prepared to be converted to the same histograms based on some 32x32 segments. Unfortunately, the paper did not go into their details as how this segment was implemented beyond comparing the differences between a 32x32 segment of the original image with the training data. In my implementation, I elected to create a window that would be slid over the entirety of the image gradually to attempt to pull up as much data as possible.

In my trials, I initially attempted to slide the window over the original image by moving it one pixel to the right and moving one down when I had completed a row. This was obviously incredibly computationally expensive. Instead, I elected to use a ratio of 1/4 of the spatial template size to shuffle the window around the image. It is still not as fast as potentially moving the window by the entire template size but I have found that it is much more accurate and tends to miss less.

After completely scanning the image, in its original resolution I decided to scale the image by a factor of two. That is, I decided to shrink the image by half and rerun my window over the same image. All resizing of the image was done using bilinear interpolation and implemented in much the same way as the Image Processing by Gonzalez Textbook describes. My spatial template size did not change so I theoretically used a 32x32 template on my first pass, a 64x64 on the second pass, 128x128 on the third, and so on. I made sure to capture both the location of a face and what scale it was detected at for future processing.

Finally, what is a face? Given that I am able to create 3 spatial histograms from my moving window over the image and I have an additional 3 spatial templates from my training data, how do I compare them? The research paper makes reference to a very computationally inexpensive and stable method of calculating the intersection of two histograms. Figure 12 shows in detail what this method is.

$$D(SH_1, SH_2) = \sum_{i=1}^{k} \min(SH_1^i, SH_2^i)$$

**Figure 12:** *Formula showing the difference between two spatial histograms described as the sum of the minimum value of each of the intensity values divided by the number of pixels present in each spatial template that are not masked to zero.*

The resulting difference is a value between [0,1] and for the purposes of this paper can be described as the similarity percentage between the two. Based on this percentage and a threshold value we can now determine if something is a face or not. The paper recommends a threshold of 0.7 as this anecdotally allows for 99.6% face detection rate with a false alarm rate of 30.2%. Up to this point, I have been collecting each and every face value in a vector to later be used to draw on the image to show visually where each of these images are.

## Drawing the Faces

This step is another that is not covered by the paper and so was implemented in a way that I found to be the most useful. Initially from my vector of faces and their respective sizes, I have a large number of overlapping faces as the spatial template is slid around the image moving only 8 pixels at a time. Due to this, drawing over the image placing squares around faces can render the entire image to a white color if there are that many detected faces. Instead, I choose to cycle through the faces in the vector and to delete any that overlap of the same size from left to right. In this way, I can see that quite a few small potentially detected faces may amount to one larger face after scaling had been done.

In order to determine if two rectangles were overlapping, I used 4 conditions that necessitate when an overlapping exists. These four conditions are shown in Figure 13 below.

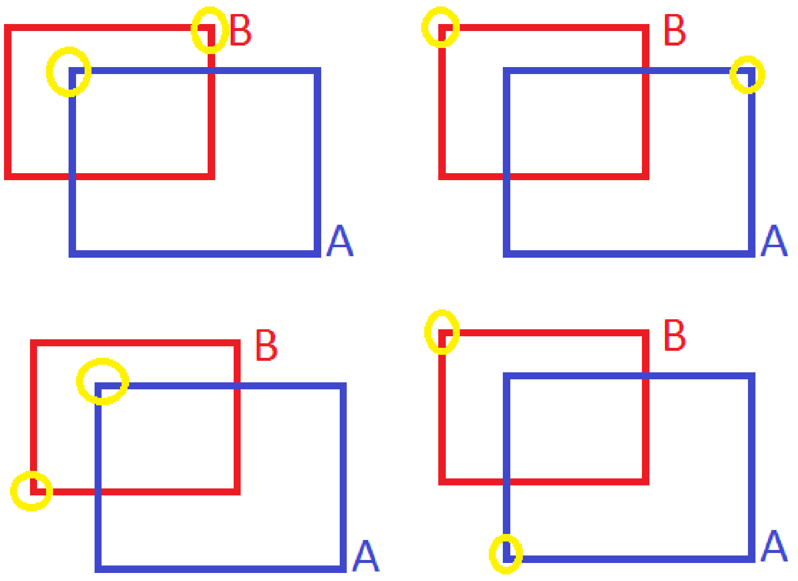


**Figure 13:** *Displaying the four necessary conditions such that failing any one indicates that an overlap exists which in this case is failed for all four. Top-Left: A's Left Edge needs to be to the right of B's Right Edge. Top-Right: A's Right Edge needs to be to the right of B's Left Edge. Bottom-Left: A's top edge needs to be below B's bottom edge. Bottom-Right: A's Bottom Edge needs to be above B's Top Edge.*

Once all of the overlapping same size squares have been removed, I proceed to "drawing" them. In this case, for the size of the square I set all of the RGB values on the edge of the square to 255 causing a white box to surround the identified face. The scaling of the square is done through calculating where the origin of each of the squares would have been on the original image by multiplying the scaling factor by the captured x and y coordinates. The rectangle size is calculated by multiplying the original rectangle size of 32 by the scaling factor and subtracting the scaling factor. An example of the completed image is provided here in Figure 14.

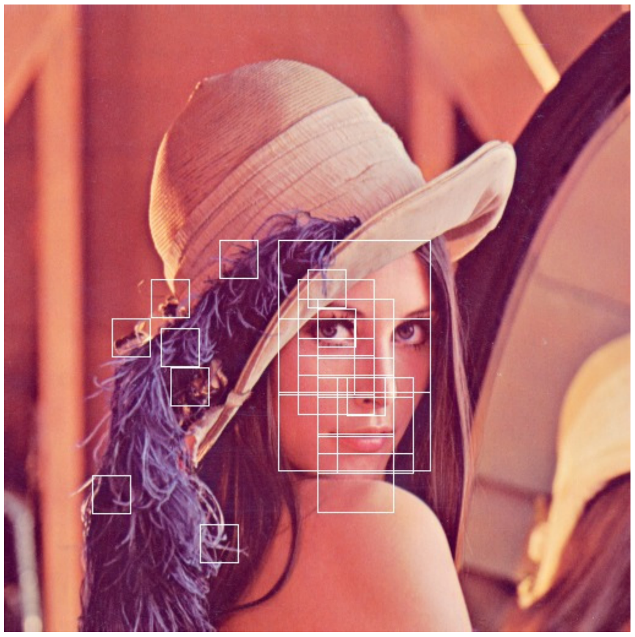


**Figure 14:** *The classic Lena image that has had squares drawn where the algorithm has detected facial features. As can be seen, some false positives are visible in the image however they are fairly minor. The over lapping boxes can be seen are a variety of sizes indicating that this area strongly matches the provided training data.*

## Experiments

### Introduction

Experiments in this paper will center around a few key deliverables. It is my goal to test out the implemented algorithm but also attempt to find where potential weaknesses may exist that could potentially be improved or create doubt that this is an effective algorithm.

First and foremost, I intend to investigate the threshold level and determine whether a higher or lower threshold may reduce false positives while still retaining the high degree of detection.

Second, I intend to test the program on portrait photos where the face in question is fairly obvious and the program is likely to find it. In this test, I will determine if the program does find the face and count the number of false positives that are generated as a result.

Third, I intend to test the program on group photos in an attempt to locate all the faces within the image. In this test false positives will also be recorded however all found and not found images will attempted to be rationalized as to why certain faces may appear while others don't.

Finally, I intend to test the program on various photos which do not in fact contain any faces at all and will count the false positives that are generated.

## Testing

This test will make use of 3 types of images; 1 portrait, 1 group photo, and 1 negative facial photos. Scoring of the success of the algorithm on each image type will be graded as follows: the portrait image must have the majority of the face identified as a facial region and minimal false positives. The group photo must identify all of the faces in the image and minimize the false positives and finally the photos with no faces in them will strictly be counted against how many there are. The highest level at which the first condition holds while the second condition is minimized will be deemed successful.

The thresholding in the paper was established to be 0.7 based on a feature value distribution constructed by testing both the images of faces and of non-faces against the database. The similarity between the tested histogram and the training data produced the curve shown in Figure 14. Based on this graph, choosing anything below 0.7 would be foolish however potentially tweaking the threshold between 0.7 and 0.8 may prove to contain interesting results. If the left distribution of the faces is found to contain fairly poor samples then it is possible that the curve may in fact exist slightly further to the right.

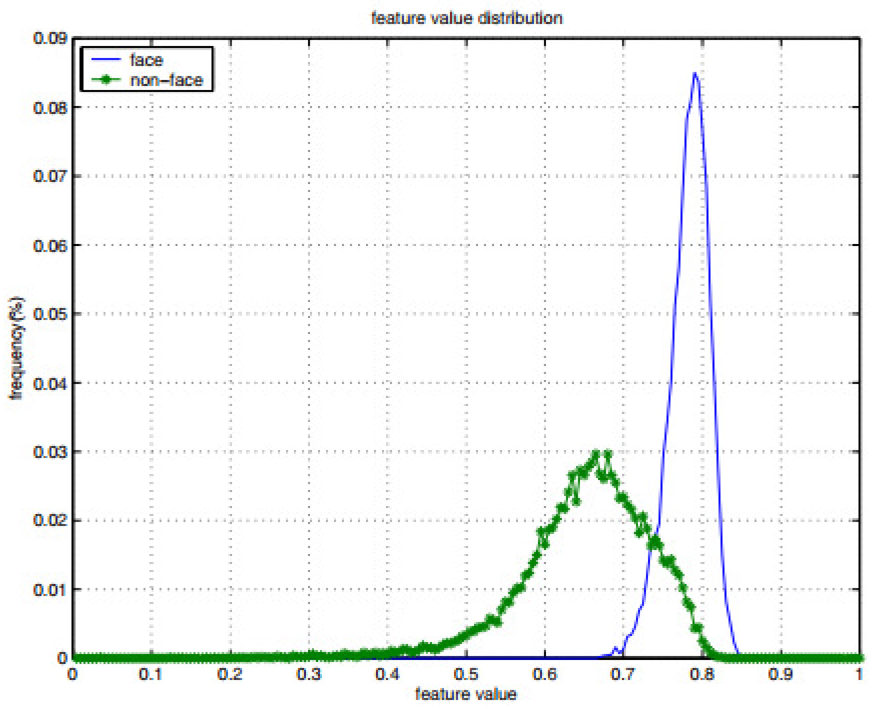


**Figure 14:** *A feature value distribution which explains as to why the thresholding value of 0.7 was chosen. It shows that for this selection, 99.6% of faces are kept with only a 0.4% loss and there are only approximately 30.2% non-faces which will meet this criteria.*

I first attempted to manipulate the threshold using the Lena picture and received results as shown in Figure 15 for changing the threshold in increments of 0.01 from 0.70 to 0.76. At 0.76, no faces were found so it will not be included. In this experiment, it can be seen that by incrementing the threshold from 0.7 that we immediately lose the larges bounding box around the face which could arguably by considered the most correct. 0.71 & 0.72 both do decrease the false positives however do lose a substantial portion of the right side of her face. In this test, increasing the threshold was completely undesirable.

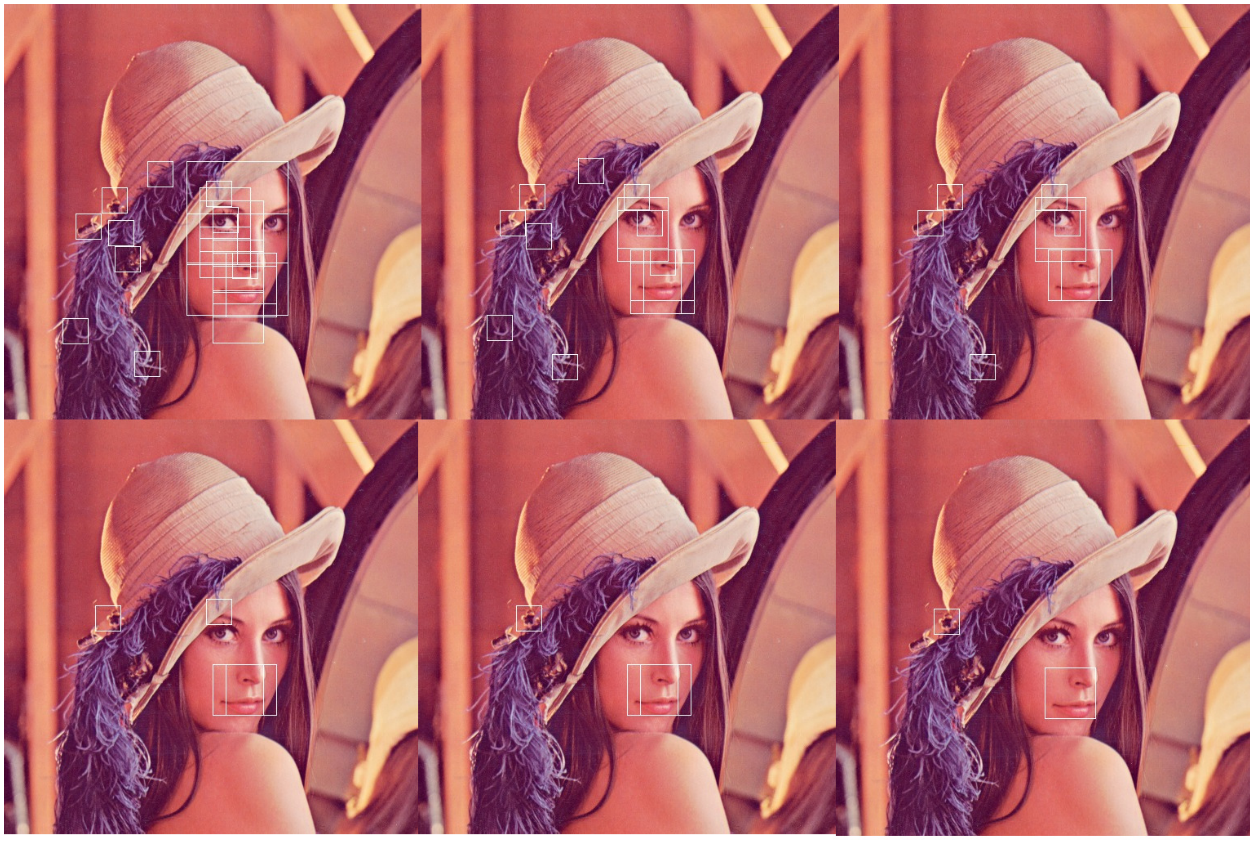


**Figure 15:** *In this figure, I show the effect of increasing the threshold value by increments of 0.01 from left to right top to bottom beginning at 0.70 and ending at 0.75.*

In my second test on groups, I downloaded a few random group photographs from google images and attempted to run the algorithm directly on the images before any preprocessing was done. Please find in Figure 16 the results. As you can see, in the first image a variety of odd shapes were detected in jeans that had some slight wrinkling in the knees. With this theory in mind, I tested is against other images of jeans with wrinkles in the knees and found this to be an incredibly consistent false positive. It is my belief that the slight wrinkling creates a sufficient pattern to be sufficiently similar to the training data.

Additionally, in the second image nearly every face is correctly detected with only 2 that do not have a centered square. It is possible that in removing the overlapping squares that they were correctly identified and then deleted in the effort to keep the image uncluttered. In this test, the second image was incredibly successful with only a potential false positive being some of their clothing. In the first image however, four faces were missed entirely. I then grabbed and cropped each of the faces to a 32x32 image and tried running it manually and found that it did in fact work and that the problem existed more so with the shifting of the window. Despite only advancing 8 pixels at a time, potentially this number would need to be decreased to increase the accuracy.

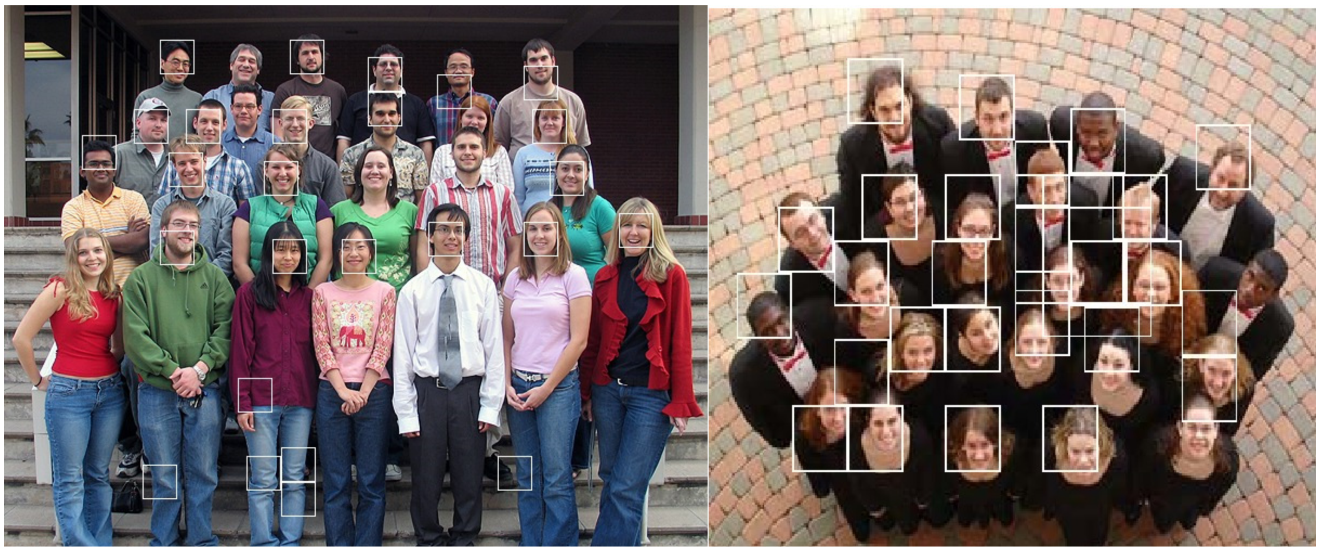


**Figure 16:** *Results from running the facial detection algorithm on random google group images.*

For my false positives test, I initially had been planning on using random color wheels and seeing if a particular combination turned out to produce false positives with a certain set of reliability. However during my testing I discovered how frequently wrinkles and light patterns in jeans can also trip up the algorithm. With this knowledge, I attempted to create the ultimate false positive test as illustrated in Figure 17. While this may seem to be a joke, further analysis of what exactly is causing this effect is very apparent. We are only using one measurement (lumen) to identify a pattern. Based on the light changes within the acid wash of some jeans and how wrinkles create potential features, these make an excellent source of potential false positive material as illustrated in the below figure.

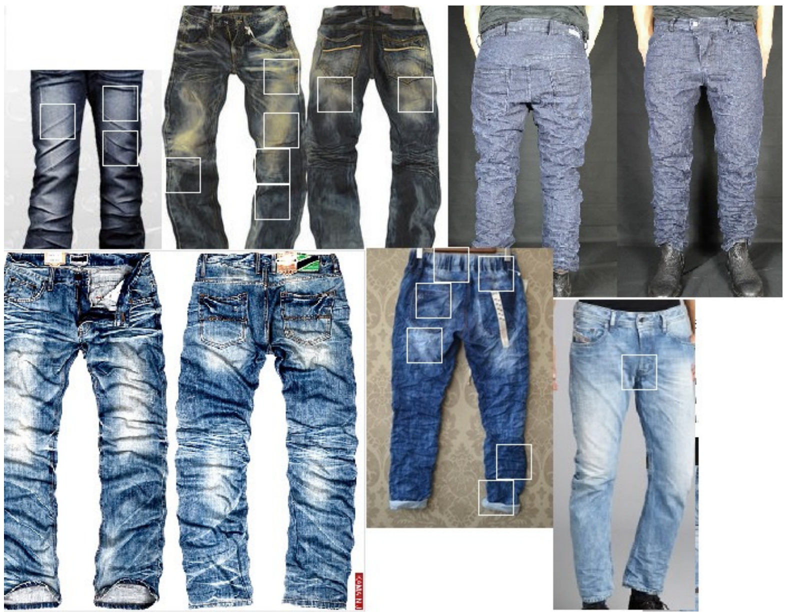


**Figure 17:** *A false positive test attempting to use what we have learned thus far about the frequent false positives to force their appearance. As we can see, there are several regions which are clearly more prone and typically have to do with a lightened pattern among a darker background or a dark area with features centered.*

With all of the above in mind and my earlier defined metric, I found 30 test images that were broken into 10 portrait pictures, 10 group pictures, and 10 false positives. I ran each of them through the program and have created the following table with the results. Unfortunately for the group photos, the earlier issue with jeans as some backgrounds causes a huge number of false positives and it does largely depend on the scaling of an image. In Figure 18, I have the same image run through the program first in its native size and then in a cropped scaled version to focus on the faces. The differences between the two are night and day.

| **Image Type** | Portrait | Group | False Positive |
|---|---|---|---|
| **Correct Identification** | 9/10 | 75/86 | N/A |
| **Incorrect Identification** | 75 | Too many to count | 135 |

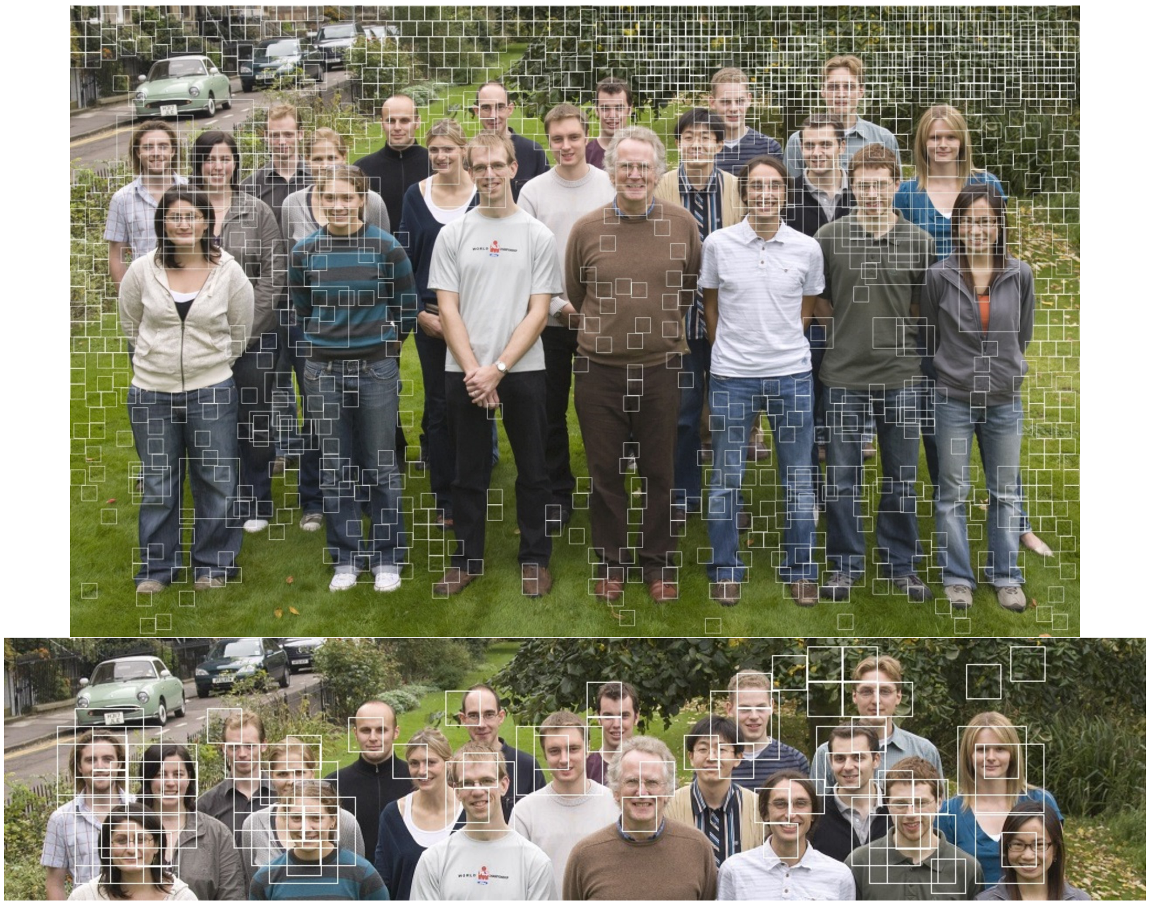


**Figure 18:** *Two group pictures the first which is unedited and full of false positives and the second which has been cropped and scaled.*

As we can see, the first group image produced a face box over almost the entire green background and anything that wasn't the group. The irony being that it didn't actually cover a good number of the faces in the picture. This issue brings up a problem that this algorithm has with scaling. The paper that I have implemented the solution from was written in 2004 and hasn't aged in the best way. 32x32 pixels with some of the cameras that have a ridiculous number of megapixels simply is not practical. For the pure amount of detail that is contained in these images it is no wonder that false positives are this abundant. Only upon scaling to a size that is on par with how the training images were constructed can come close to finding the correct answer here. In this case, the remedy is one of scaling. Either all of the training data needs to be scaled up to a larger size to better accommodate these newer larger images, or instead the images need to first be scaled down by the program in order to accommodate.

## Conclusion

In this paper I have discussed the problem facing the industry of image detection and have shown a proposed solution suggested by researchers. Further, I have implemented said solution and verified the claims that it makes in regards to its accuracy. In my implementation however I did not implement the SVM module so my test results still contain the high degree of false positives that the researchers were able to weed out using the SVM.

The localization process achieved by the algorithm is very effective in itself and is very computationally insignificant. I believe that it casts a sufficiently wide net in order to catch all of the potential regions to minimize the usage of SVM. In this sense, it is extremely efficient and does an excellent job at reducing the overall processing time.

Future work would include expanding the initial image database from the 50 images. While this is a good start, it does not nearly encapsulate all of the variables necessary to accurately describe the color pattern of a person's face. Further, I would like to implement the SVM engine and verify the remainder of the papers claims now that facial regions have been narrowed down. A large pitfall in my program is that in the effort to keep the resulting image as uncluttered as possible overlapping facial regions are deleted. A future implementation would contain a method for creating a bounding box to contain and certain all identified facial squares. This would allow the square to actually center around the person's face instead of creating multiple boxes that make up the face. Finally, as mentioned in the previous section, scaling is a huge issue with this algorithm in the world of thousands of pixels.

## References

Gonzalez, R. C., & Woods, R. E. (2002). Digital image processing.

Ojala, T., Pietikäinen, M., & Harwood, D. (1996). A comparative study of texture measures with classification based on featured distributions. Pattern recognition, 29(1), 51-59.

Pietikäinen, M., Ojala, T., & Xu, Z. (2000). Rotation-invariant texture classification using feature distributions. Pattern Recognition, 33(1), 43-52.

Swain, M. J., & Ballard, D. H. (1991). Color indexing. International journal of computer vision, 7(1), 11-32.

Zhang, H., & Zhao, D. (2005). Spatial histogram features for face detection in color images. In Advances in Multimedia Information Processing-PCM 2004(pp. 377-384). Springer Berlin Heidelberg.