Mask Detection

Project Report by Garima Maheshwari and Hailey Schauman

Table of Contents

Table of Contents 1

Problem Statement 2

Objectives 2

Algorithms 2

- Edge Detection 2
- Object Recognition 3
- Image Cropping 3
- Transformation Space 3
- Divide and Conquer 3
- Ratio at Each Transformation 4
- The Bound 5

Constraints 5

- Divide and Conquer Translation 5
- The Bound 6

Results 7

- Positive Testing 7
- Negative Testing 8
- Incorrect Results 9

Conclusion 10

References 11

Problem Statement
When given a search image with a person in it, detect if that person is wearing a mask.

Objectives
Due to the complexity of the problem, we identified constraints before establishing our objectives. First, due to the variety of masks that a person can wear (e.g., n95, duckbill masks, cotton masks) and how the masks when worn can take any form, we focused specifically on front-view perspectives with cotton masks. Furthermore, when performing positive testing, it was guaranteed to have only one person in the image who was facing forward.

Therefore, using a front-view cotton mask, our main objectives became the following:

- Create a transformation space
- Perform divide and conquer on a transformation space
- Perform divide and conquer on translations
- Have successful positive and negative matches with a person wearing a mask

These are the additional objectives we had established but were unable to achieve:

- Have successful positive and negative matches with side-view perspectives of the cotton mask
- Test against n95 masks with front-view and side-view perspectives
- Perform lip detection
- Given a search image with multiple people, detect if they were wearing masks

After our initial testing, we realized that the exemplar mask we were working with constrained us to test masks with similar texture and color. For this reason, we mainly focused on detecting positive and negative matches with a person who was either wearing or not wearing a darker colored mask. Later in our constraints section, we explain some of the constraints that we had which prevented us from accomplishing the additional objectives as stated above.

Algorithms
To provide an efficient solution, we divided the problem into two parts: (1) perform edge detection on the exemplar and search image and (2) perform object recognition. The second part was divided further into five subparts: (1) crop image, (2) create transformation space, (3) divide and conquer, (4) receive the ratio of edges over size of the image at a specific transformation, and (5) determine the bound.

Edge Detection
The purpose of performing edge detection was to remove the color and texture of the mask, but still keep the shape of it, so that it could be used against the exemplar. To do this successfully, the images were first converted to gray-scale images, then had one iteration of smoothing with a Gaussian filter, and lastly were computed with edge detection using the Canny function. We used OpenCV calls to perform these operations on the images.

While the main objective was to remove color and texture, it was imperative to come up with a minimum and maximum threshold that was specific to each search image, as each image needed their own threshold bounds. After research was conducted, it was found that by computing the average amount of gray in the image, this value could then be used to obtain a threshold range applicable to an individual image (Wong). Through testing, it was found that subtracting 60 from the average of gray and adding 30 from the average gray computed a threshold range that depicted the mask outline while also removing unnecessary edges.

Object Recognition

The purpose of this part was to perform object recognition using an edge-detected exemplar and search image.

Image Cropping

An algorithm was developed to crop an image after edge detection. This was done for computational performance as it reduced time and space complexity due to removing spaces where no edges existed.

Transformation Space

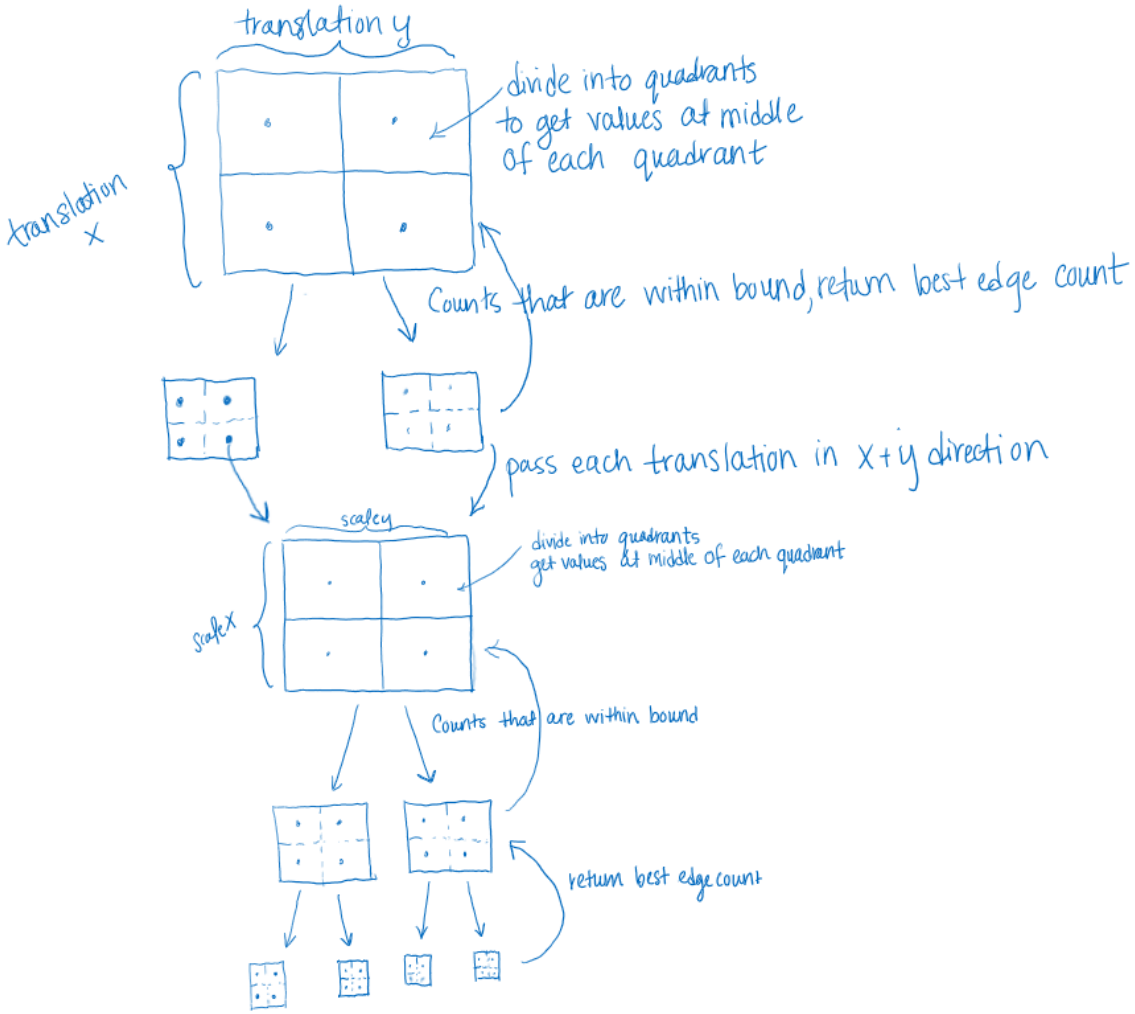
To handle different sizes of an exemplar image against a search image, a 3D transformation space was created. Within this space, combinations of scales and rotations were stored. Therefore, at each translation of a search image, the transformation space provided the ability to transform the exemplar image with different scale and rotation values. By doing so, the program would have the ability to detect a variety of front-view perspective cotton masks.

Divide and Conquer

The divide and conquer algorithm was used for the transformation space and the translations of a search image. The objective of the algorithm was to find the best match with a reduced runtime for the program. To accomplish this, the algorithm would divide the space into four quadrants. At each quadrant, the values in the middle would be used to determine the transformation values for the exemplar. Then, with these values, the algorithm would calculate whether that transformed exemplar was a successful match at that translation. If it was, it would divide further.

Refer to Figure 1, which shows a visual representation of the algorithm for the translations and transformation space.

Figure 1: A visual representation of the Divide and Conquer algorithm



Ratio at Each Transformation

To calculate the number of edge matches at a specific transformation, we created an algorithm that would use the transformation variables (e.g., scales, rotation, and translation) to compute where the exemplar should be in relation to the search image. Iterating through the exemplar, we kept track of the number of edge matches it had with the search image and the total number of edges.

After iterating through the image, we returned edge matches and total number of edges. Using these variables, we calculated the ratio to determine how successful the match was. For example, if an image had 50 matched edges with a total of 100 edges at that specific transformation, there was a 50% match.

The Bound

For our bound, we took an approach that would allow us to adapt to the edge ratio within the search image and the number of times we had already divided and conquered. First, we take in the ratio that was computed with the transformation. Then, we used the edge ratio within the search image we were iterating through to decide if we were going to create a larger or smaller bound for the current quadrant ratio. We also used another variable that was known as the levelOfDivide, which was the number of times that the divide and conquer had already occured. The higher the level of divide, the stricter the bound we wanted.

If the search image ratio was above 0.06 and the transformation ratio we were checking was greater than 0.20 multiplied by the level of divide, we would return true, meaning that the ratio was within the bound. If this was not the case, we would return false for the checkBounds function. Another way that the bound could pass is if the search image did not have a lot of pixels, hence it's edge ratio was smaller, and we would keep a less stricter bound. For this condition, we multiplied by 0.15 by the level of divide to get a number that would allow more ratios to return true for the checkBounds function.

With less edges in the search image, we figured that we would need to search through more of the image before determining whether or not the image did contain the exemplar or not.

Constraints

The complexity of the mask detection program was high due to the different types of images that were being passed through. For example, after performing edge detection, the search image could contain few edges compared to its overall size, resulting in a small ratio. However, if the search image was small and had many edges, it resulted in a high ratio compared to edges and image size.

Due to our divide and conquer method being dependent on the bound, which used ratio size to determine if a quadrant was good enough, it led to inaccurate results when testing. Therefore, we established constraints as listed below.

Divide and Conquer Translation

The Divide and Conquer method for translation was used to determine if a quadrant with a x translation and y translation was a good enough quadrant for a recursive divide and conquer given the ratio that was calculated from the transformation space. However, after testing, it led to inaccurate results because some images had low amounts of edges compared to the overall size.

For example, our first test used was to test the exemplar against itself. With the image being cropped, the edges were formed around the border. Thus, when performing divide and conquer on the first iteration, the edge ratio was significantly small due to the translations being in the middle of the image. This resulted in a negative match because the exemplar image never reached translation (0,0).

To combat this, we put a constraint on the divide and conquer translation. When calling the match function (used to detect if there was a match), it computed the search image's ratio of edges and image size. If the ratio was higher than 0.05, that indicated the image had a high amount of edges; whereas, if the image's ratio was lower, it would iterate through a for loop at iterations of 25. This then allowed the program to detect multiple translations on an image with a small ratio.

However, we acknowledge that this constraint would not be needed if the bound algorithm was accurate. Please refer to the next section, which discusses our constraints with the bound method.

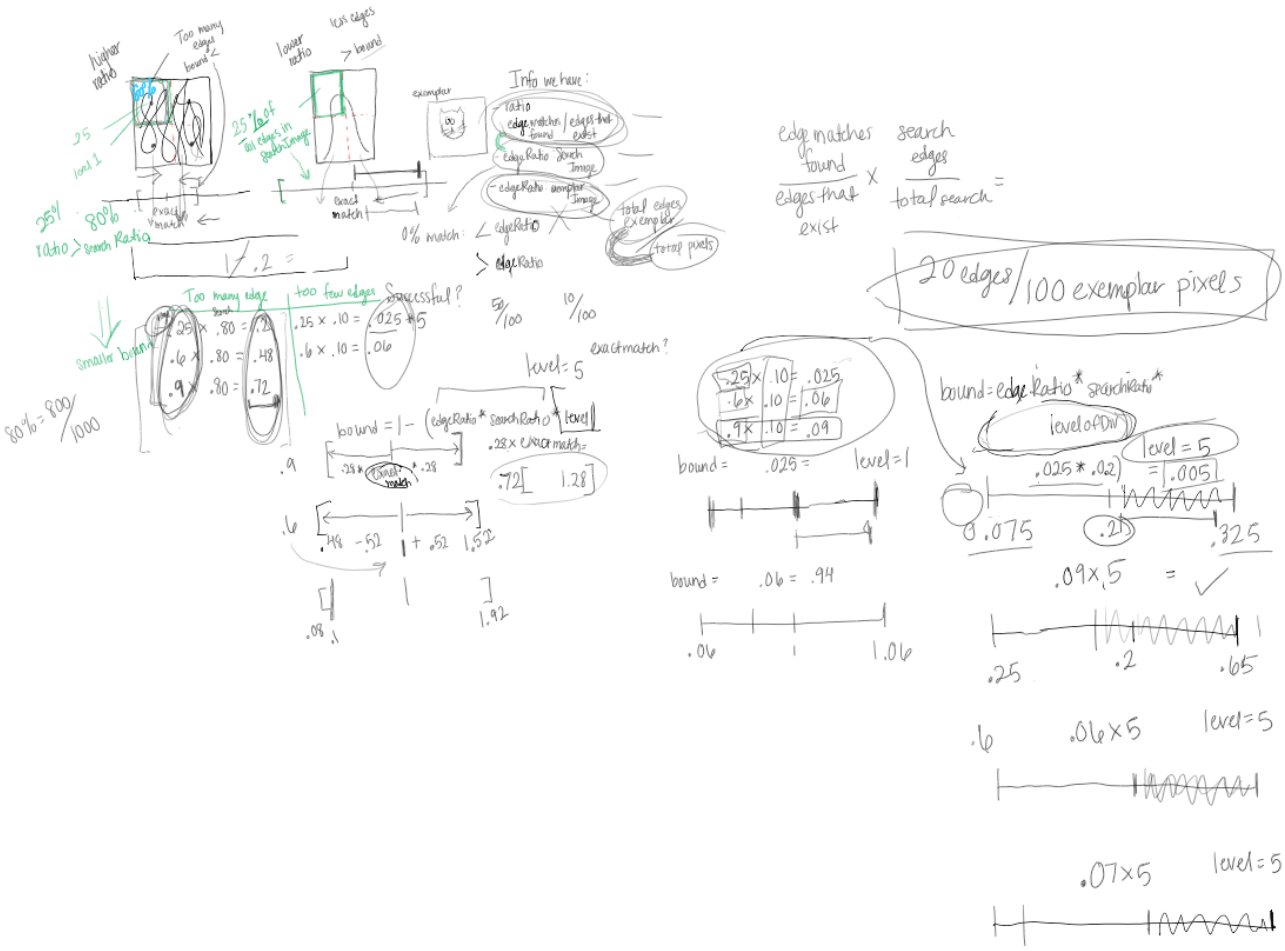
The Bound

When determining the bound that we would use for our divide and conquer theorem we tested a lot of different methods. Through trial and error we determined an equation with a few hard coded values(constants) that we felt would give us the most accurate results and also be efficient.

To start off we tried to keep our bound a constant number. After testing, we realized that this number would need to change and become stricter as we divide and conquer more into the transformation space. For this reason, we implemented a variable which would keep track of the level of divide. Using this variable, our bound was multiplied by this, which resulted in a time decrease that was spent looking through the transformation space. Although this was helpful, we were still not able to get counts that were high enough to say if a person was wearing a mask. We also noticed that we were getting abnormally higher results for our test images where a person was not wearing a mask. We determined that this was due to the fact that the ratio of edges to total number of pixels within the search image would result in a higher percentage of edge matches being found, which would lead us to return a higher match percentage despite there being no mask in the picture and too many edges.

After all the different methods, we ended up using the algorithm stated in the Algorithm section for the bound. Through trial and error, we realized that since we were testing a face mask that was considerably different from the mask that was found in our search Image, it would cause a greater chance of error when doing object recognition with edges. Below in Figure 2 are some of the calculations we performed when trying to come up with an equation that would generalize the edge density within the search Image. We were unfortunately unable to determine an equation, but did end up finding good hard-coded values for the exemplar we were testing.

Figure 2: Some of the calculations we made when trying to find an equation for the bound



The figure above contains notes on how we were trying to figure out a way to determine an equation for find a proper bound given: (1) the ratio of edges matched and the total number of edges within the exemplar, (2) the ratio of the number of edges and pixels found in the searchImage, and (3) the ratio of the edges within our exemplar to the number of pixels in the exemplar.

On the far left of the image you can see that we are trying to generalize the bound so that divide and conquer would work accurately for searchImages that contained a higher ratio of edges. We figured we would need a smaller bound for an image that had more edges within the searchImage. We tried to test out an equation where we would multiply the searchImage ratio to the ratio of edges matched and the level of divide and subtract the entire number from 1.

After trying to implement this in our code, we realized hard coded values would provide better results due to this solution not allowing us to generalize the ratio of edges within the searchImage.

Results

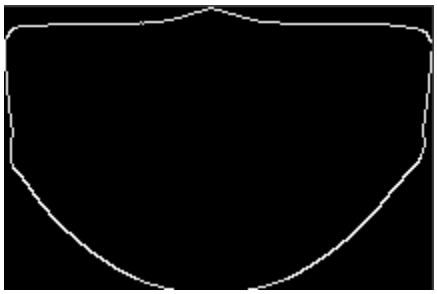
See the following sections, which discuss our results involving positive testing, negative testing, and incorrect results.

Positive Testing

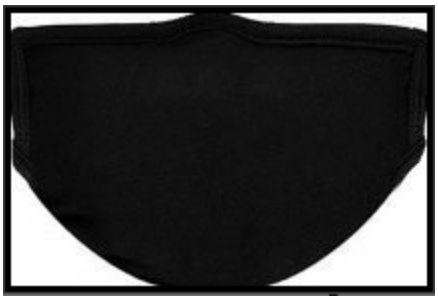
For our positive testing, we used a front-view perspective of a cotton mask as an exemplar and search image. This was one of the first tests used to determine if our algorithms were working correctly. As seen in Figure 3, a positive match was detected, which is outlined with a black border of the original, cropped search image.

Figure 3: Results after performing object detection on the exemplar image.

Exemplar/Search Image after Edge Detection



Results



After completing object recognition on the search image in Figure 3, the highest ratio of matches was 100.00%, which was expected with an exact match.

Negative Testing

For the negative testing, we used the same exemplar as seen in Figure 3. The two search images used contained one person who was not wearing a mask. As seen in Figure 4 and Figure 5, the results were successful as the exemplar was not found in the images.

Figure 4: Results after performing object detection with the exemplar image and a search image.

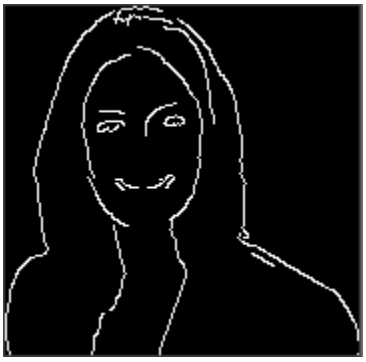
Search Image after Edge Detection

Results

After completing object recognition on the search image in Figure 4, the highest ratio of matches was 40.8895%.

Figure 5: Results after performing object detection with the exemplar image and a search image.

Search Image after Edge Detection



Results

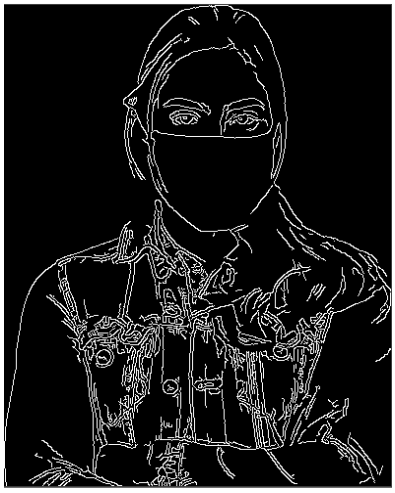
After completing object recognition on the search image in Figure 4, the highest ratio of matches was 19.2156%.

Incorrect Results

After performing object detection on other images, it showed that there false positives, false negatives, and incorrect out images. See Figure 6, 7, and 8, which highlights these results through search images.

Figure 6: Results after performing object detection on an image that had positive results, but had an incorrect output.

Search Image after Edge Detection



Results

After completing object recognition on the search image in Figure 6, the highest ratio of matches was 79.1798%. While this had a high amount of accuracy, the output image showed that the match was inaccurate. Instead of detecting the mask, it detected the person's hair, which contained a lot of edges.

Figure 7: Results after performing object detection with the exemplar image and a search image. This resulted in a false positive.

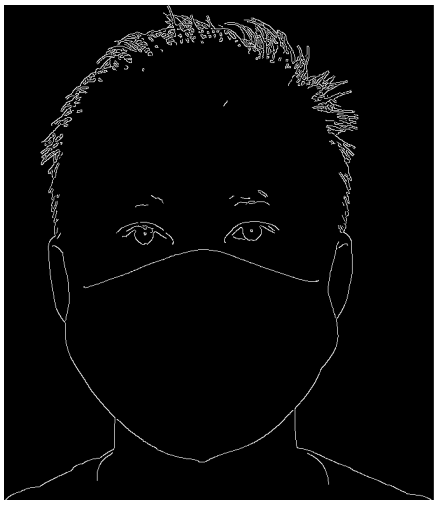
Search Image after Edge Detection

Results

After completing object recognition on the search image in Figure 7, the highest ratio of matches was 84.3849%.

Figure 8: Results after performing object detection with the exemplar image and a search image. This resulted in a false negative.

Search Image after Edge Detection



Results

After completing object recognition on the search image in Figure 7, the highest ratio of matches was 56.1514%.

Conclusion

In conclusion, we believe we were able to create an algorithm for object detection that could work on detecting masks that were either exactly the same or similar to the exemplar image. We believe that given more time, we could also test against color histograms to account for factors of the mask that include color and texture, which would result in better matches on the image. We also learned that in order to generalize the different masks that could be detected by our program we would need to train our program to test against many exemplar images, instead of just the one that we had. Furthermore, we believe that if we were able to create an algorithm for the bound given any search image, this would allow an increased success rate.

Our main takeaway for this entire project was understanding how to create a transformation space given an exemplar and develop a divide and conquer algorithm that focused on computational efficiency, which allowed us to detect an exemplar image taking different transformations. Furthermore, we now understand that the scope of the project is complex and is dependent on many factors, such as the type of mask, color, and the form the mask has taken when being worn. Unfortunately, we mainly focused on one shape of a mask (outside of scaling), which is why our object detection was not versatile with images.

How to Run the program

Please compile using openCV and once set up. Place runTest.bat in the folder containing cpp files. This is the bat file that must be run in the command line.

References

Wong, Kerry. “Canny Edge Detection Auto Thresholding.” *KerryWong*, 7 May 2009, www.kerrywong.com/2009/05/07/canny-edge-detection-auto-thresholding/. Accessed 30 Nov. 2020.