Project Final Report

Automated Face Blurring Software

Course: COMP 4102
Date: April 19, 2021
Student Name: Adrian Ong
Student Number: 101032077

Table of Contents

Abstract:

For this project, the tool I created is an auto face blurring software. The purpose of this software is to be able to recognize and blur faces both in still images, and live video taken from a webcam. This problem was challenging for me personally as I have never completed a project like this before, and I knew that the program would have to deal with a range of different faces, and different facial angles. This report will delve into my journey into what I learned, and how I created this tool.

Introduction:

As stated in the “Abstract” section above, this tool is an auto face blurring software that is meant to recognize and blur faces in both images and in live video. This would use the concept of first being able to recognize faces within a photo, as well as the concept of manipulating the pixels where the faces are located, to make them unrecognizable. For this tool to work, one would need to either upload a photo as input, resulting in the software outputting the blurred version, or have a working webcam for the “live” functionality of the tool.

The image below is an example of what I hoped to achieve with this software.

The first part of the problem was ensuring that my program would recognize faces. One of the reasons why this problem is challenging is because the software would need to be able to recognize a diverse set of faces. If the training data is too narrow, then the software might struggle with recognizing scars, glasses, different ethnicities, smiles, etc. Another challenge would be dealing with the angles of a face within a photo. If the person is looking to the side, and the software sees a very asymmetrical face, ideally it should still be able to detect that person’s face. Skin tone can also be a challenge depending on the background of the image. For example, it may be challenging to detect a fairer skinned person in a bright background, and a darker skinned person in a dark background, because the face blending into the background would make it more difficult to detect the edges of the face.

The second part of the problem is ensuring that the blur strength is sufficient so that it would be difficult to recognize the face being blurred. If the blurring isn't strong enough, then the tool is essentially useless, so ideally this outcome is avoided.

When I embarked on this project, I had the goal of learning how to create a reliable Computer Vision software that can identify a desired object, and track it as it moves, as this could be used in a plethora of applications that can assist people, and change how we interact with the world on a day-to-day basis.

Background:

In this section, I will be citing relevant papers, software packages, etc.

Mittal, Aditya. "Haar Cascades, Explained." Medium, Analytics Vidhya, 21 Dec. 2020, medium.com/analytics-vidhya/haar-cascades-explained-38210e57970d#:~:text=Haar%20cascades%20are%20machine%20learning,combination%20of%20%E2%80%9Cweak%E2%80%9D%20learners.

"Opencv-Python." PyPI, pypi.org/project/opencv-python/.

"Smoothing Images¶." OpenCV, opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html.

"Cascade Classifier." OpenCV, docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html.

This background material relates to my chosen problem through my use of Haar cascades for my face detection problem, and Gaussian blurring in when blurring the detected faces. The first and last sources cited pertain to the Haar cascades, and the math behind them. However, the Haar cascade I used for my tool is called `haarcascade_frontalface_default.xml` as it was sufficient for my use case, and worked very well during testing. The second source cited delves into OpenCV's cv2 library, which is the most important library used to complete this project. Some of the functions I used from this library include `CascadeClassifier`, `GaussianBlur`, and `VideoCapture`. Finally, the third source cited delves into Gaussian blurring, and the theory behind it.

Approach:

As mentioned in the Background section, the most important software library I used is `cv2`. Much of the functionality of the software required `cv2` functions such as `CascadeClassifier`, `GaussianBlur`, and `VideoCapture`. When I first thought of this project, I wasn't sure how I was going to go about training the software to recognize faces. Thankfully, the TA who looked over my proposal suggested I could use a pre-trained cascade model, which is provided by OpenCV. I chose to go with this approach as it had the best balance of effectiveness, and quickness to implement. Haar cascades are object detection algorithms that are based on machine learning. They use Haar features to figure out the likelihood of a certain point being a part of a certain object. To get a Haar feature, calculations are performed on adjacent rectangular regions at a specific location in a detection window. This involves getting the sum of the pixel intensities for each region, then afterwards, calculating the differences between the sums. Here are some examples of Haar features below.

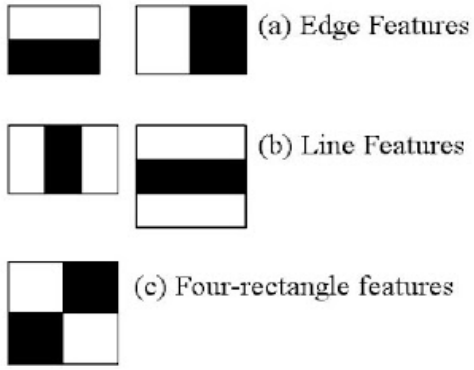

Fig. 1. Mittal, Aditya. Types of Haar features.

Afterwards, what happens is the creation of integral images. Their purpose is to speed up the calculation of Haar features by creating sub-rectangles and creating array references for each sub-rectangle, rather than computing at every single pixel. These sub-rectangles are then used to compute the Haar features. The image below demonstrates how an integral image works.

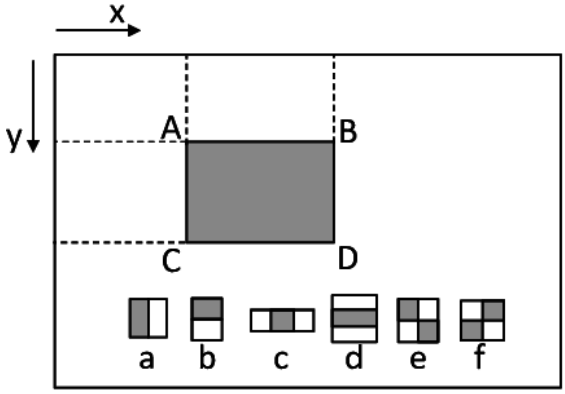

Fig. 2. Mittal, Aditya. Illustration for how an integral image works.

For the next step, Adaboost training is used. Adaboost is used to choose the best features, and trains the classifiers to use them. Combining the weak classifiers, it creates a strong classifier that the algorithm uses to detect objects. This process is called cascading classifiers.

The next important aspect of my project is Gaussian blurring. In this project, Gaussian blurring was implemented using `cv2.GaussianBlur`. One parameter being passed into this function is the rectangular area that we want to apply the filter to, which includes x and y coordinates, as well as the width and height of our desired rectangle. In the case of this project, each rectangular area is what the Haar cascade already labeled as face. The next parameter is a tuple which represents the kernel size. When setting this parameter, I ensured that the kernel dimensions were odd integers, which is necessary to ensure that the kernel can be placed at a central (x, y) coordinate of the input image. This was done by taking our width and height values, dividing each of them by some factor, then checking if the resulting value is even using the mod operator. If we see that the value mod 2 equals 0, then we subtract 1 from the kernel width or kernel height parameter. The purpose of our factor is to control the size of our kernel. Since we are dividing by the factor, a higher factor value means a higher denominator, which results in a smaller kernel size and less blurring power. This means that a lower factor value results in a higher blurring power. The value 2 was chosen as the factor for this tool in order to ensure a more anonymous blur.

Results:

The application was successful! Since the Haar cascade used was trained very well, the challenging scenarios mentioned (skin colour, background contrast, head angle, etc.) were not a significant problem, and the faces were still correctly detected the vast majority of the time. The only cases where the tool fails are when the head is tilted sideways at an extreme angle, and when the head is turned all the way to the side, so that only a side profile can be seen. False positives were not at all an issue. Below are some blurred images generated by this tool.

Fig. 3. Ong, Adrian. Blurred Image of Abba using a black and white input image.

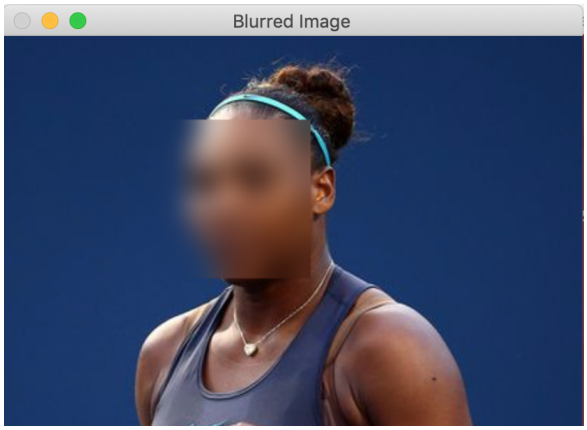

Fig. 4. Ong, Adrian. Serena Williams blurred.

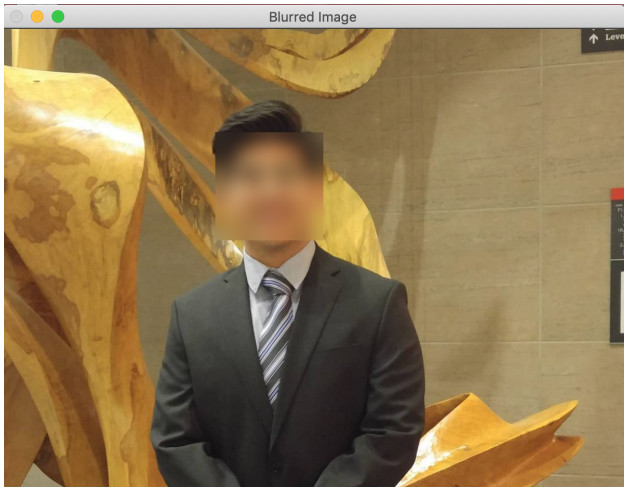

Fig. 5. Ong, Adrian. A blurred picture of myself.

As we can see from the results, each image was handled correctly despite the different skin tones in each one. In *Fig. 5.*, my face, the wooden sculpture in Richcraft Hall (formerly River Building), and the background are all similar in colour, and the tool still correctly blurred my face. In *Fig. 3.*, it is also demonstrated that the tool can handle multiple faces, even when they are very close together.

Important Notes:

- The demonstration of the live webcam portion of this tool will be demonstrated in the video presentation.
- Sources used to write this report have been cited in the Background section of this report.
- This was a solo project.
- The link to the GitHub page is: https://github.com/adrianong1/comp4102FinalProject.