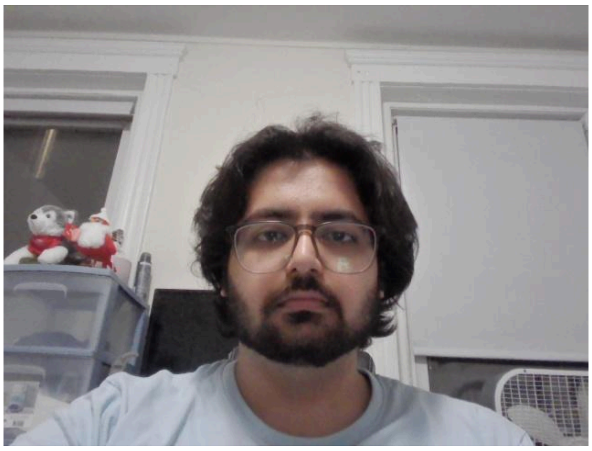
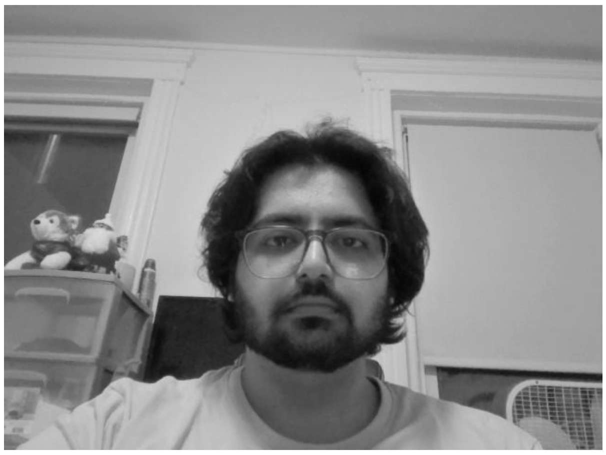
Ronak Harish Bhanushali
CS 5330 Project 1
Github - https://github.com/ronakhb/cs5330 (The repo is private but I can give access to TAs if needed)

Project 1

This project deals primarily with basic OpenCV functions and data types. The main focus of this project was implementing different types of filters. A filter essentially manipulates the image data and transforms it into something meaningful that can be applied for specific use cases. One example was seen in this project where we implement a grayscale filter and we use that grayscale image to detect faces using Haar Cascade. Other examples of filters giving useful information are Sobel X, Sobel Y and gradient magnitude. Other filters like blur, sepia and vignette just provide visual effects. Key learnings from this project were working with cv::Mats (manual pixel manipulation), applying convolution, reading writing images and video feeds, working with opencv datatypes.

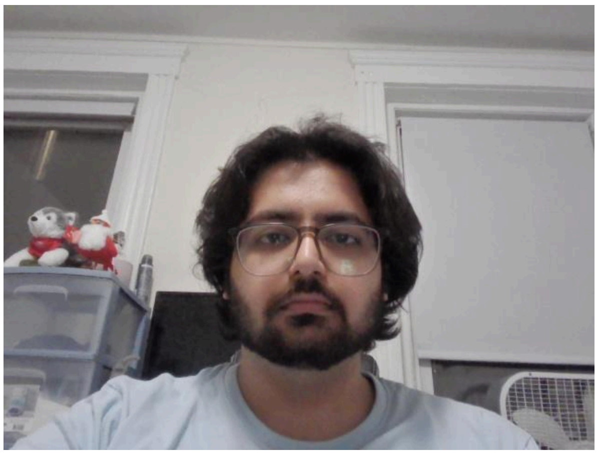
Grayscale

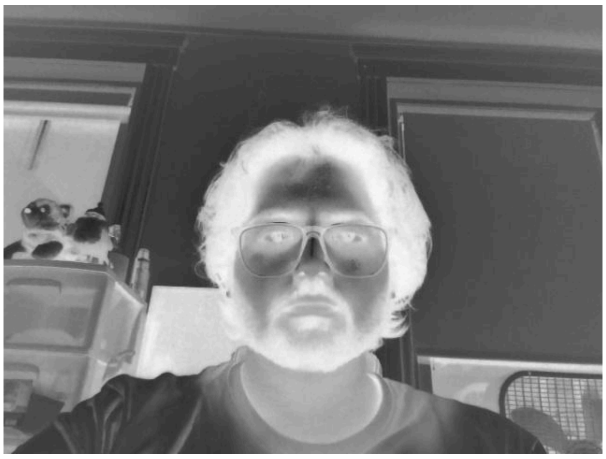




The image on the left is the original image and the image on the right is generated using the in built openCV function cvtColor(). This function can be used for conversions other than grayscale too.

Custom Grayscale



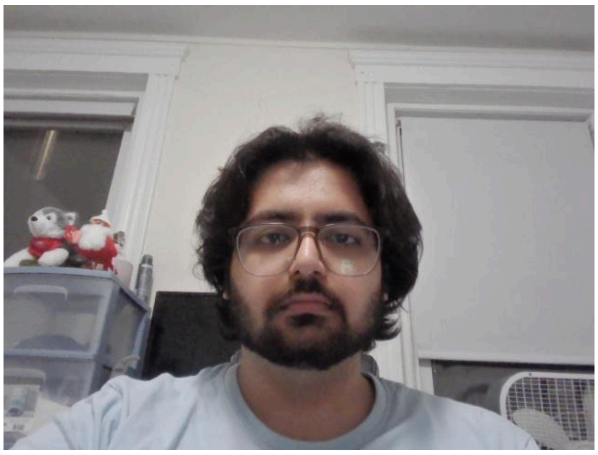


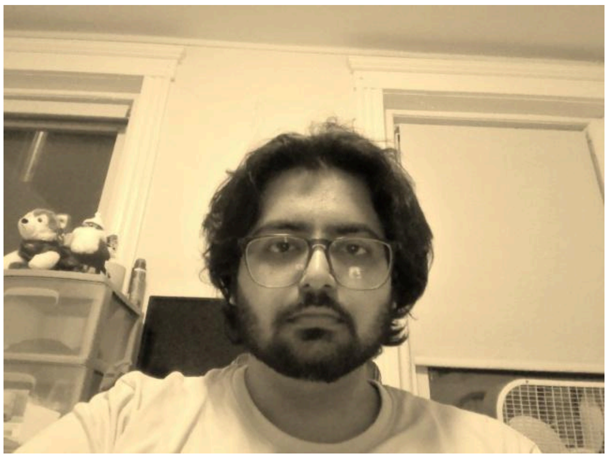
This filter takes a pixel and changes its value to 255 - ((r+g+b)/3). This is essentially taking the negative value of average value of all three color channels and using that value for all channels

The main difference apart from the negation is the weightage of the three channels. On the opencv documentation, we see that the grayscale value is obtained as follows -

$$\text{RGB[A] to Gray:} Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

Sepia Filter





Since Sepia alters the RGB values at the pixel level, I saved the B, G, R values in three variables and calculated the new B, G, R values using those before writing the new values to the dst matrix.

Extension 1
I saw that the sepia filter could easily be written as a (3x3) matrix and I could use matrix multuplication with the (3,1) pixel vector to directly get the new BGR values instead of having to manually perform the matrix multiplication. The objective of this extension was to go more in depth in openCV functions and try to find a way to not use so many for loops. This extension also extends into other filters too. However the first thing I learned from trying this was that data types are extremely important when dealing with cv::Mats. I was not able to get my code to run for a long time as the matrix multiplication would give me an error due to data type. Even after that, the output was not right. The solution to this was to cast the data types at every step and convert the cv::Mat before starting to process. Second thing I learned from this is that openCV matrix multiplications are not efficient. I get a very laggy feed when I use this implementation instead of manual manipulation. (This implementation is named sepia2 in the filters.cpp file). However despite these issues, I get the same output with the sepia2 filter and I was able to implement what I had planned.

5x5 Blur

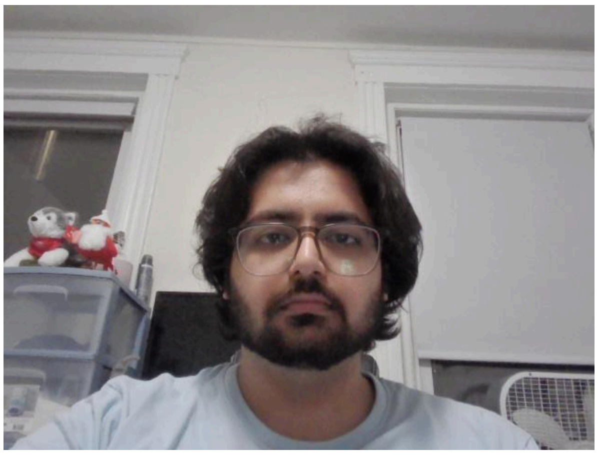
I have made these images a bit larger to show the effect of the blur. Since my laptop could only handle 640 x 480 images, the original resolution is low and its a bit difficult to see the blur but it is definitely there.
The seperable filters show a visible improvement when running the code. It can also be seen in the timeBlur output -

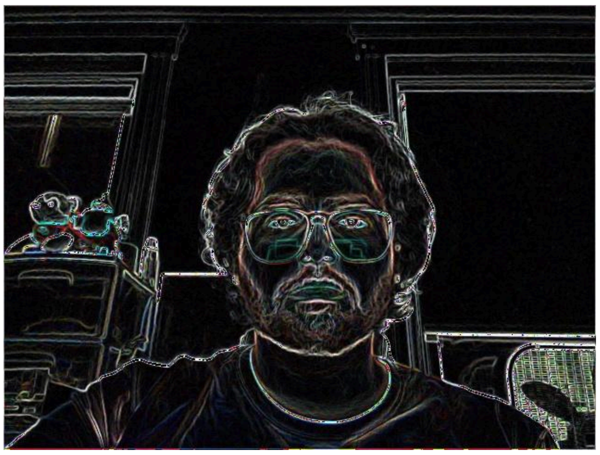
```
Time per image (1): 0.2174 seconds
Time per image (2): 0.1458 seconds
```

Extension 2
The purpose of this extension was similar to the first one. I wanted to go deeper and learn more about how matrix slicing and summing works in opencv. I made a similar kernel in this but now instead of pixel wise multiplication, I had a slice of the image and I had to use that. I faced similar datatype problems but mainly I learnt about how opencv handles data.

cv::Mats are smart pointers and are always passed as pointers, however this was also true when I was working with slices and individual pixels. I had to make sure to clone everything I was using to make sure I did not alter the original image as the convolution kernel needs to reuse the same pixel at multiple places. I got it to work and it was again slower than using for loops.

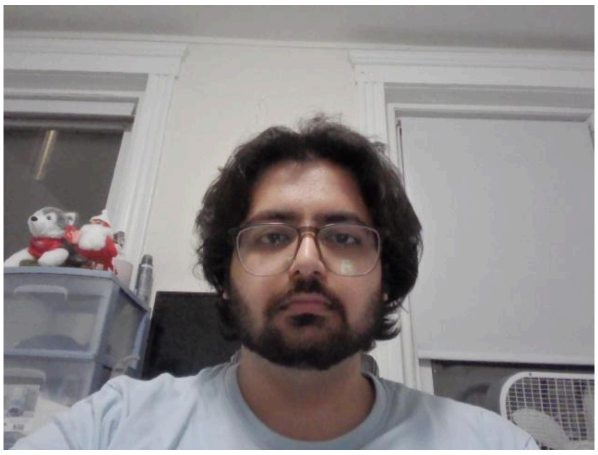
Gradient Magnitude

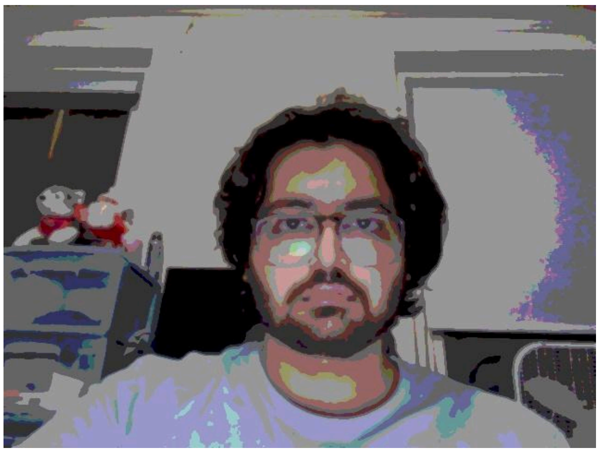




The gradient magnitude shows all the edges in the image

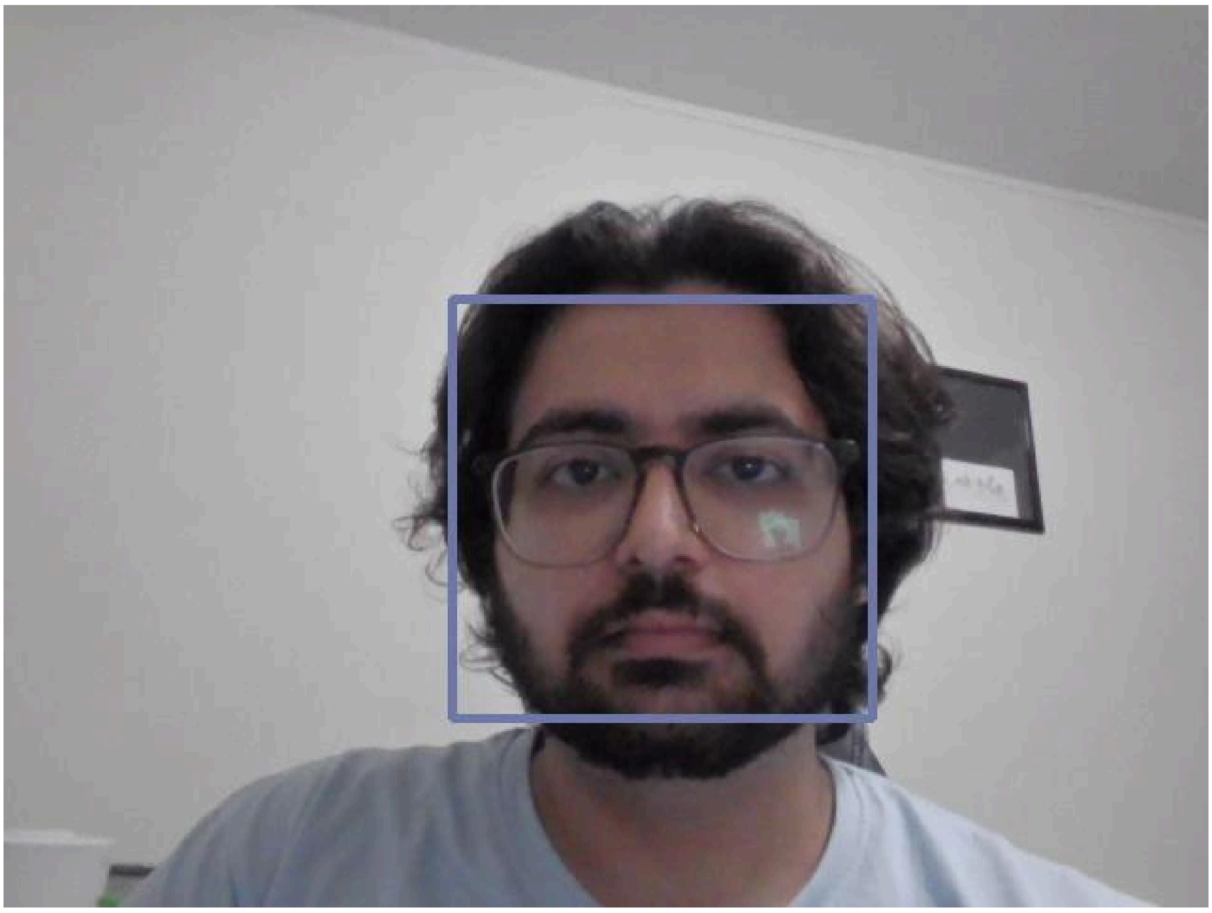
Blur/Quantized





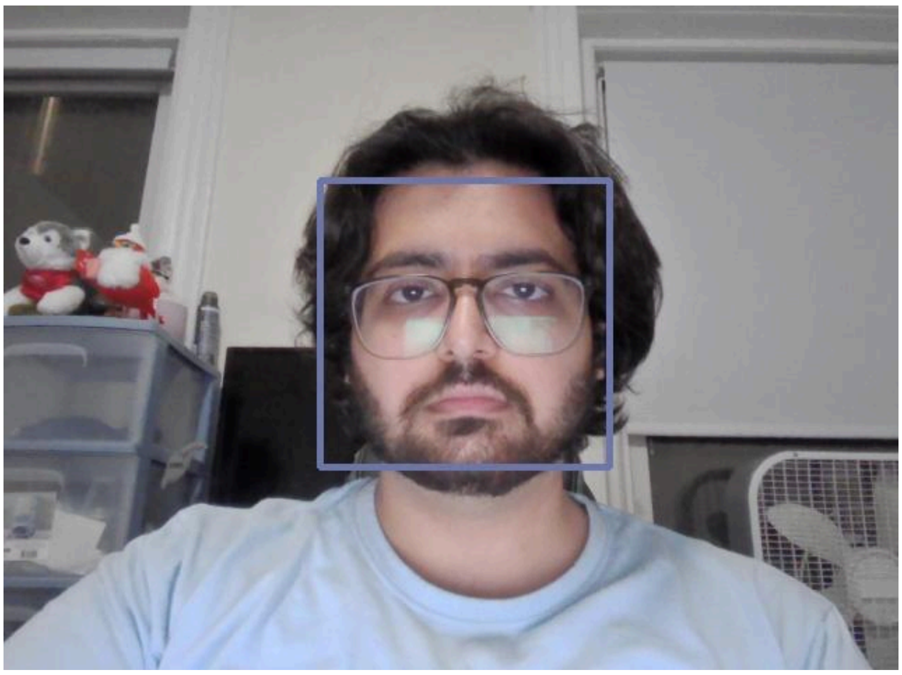
The level used to generate this image was 5

Face Detection

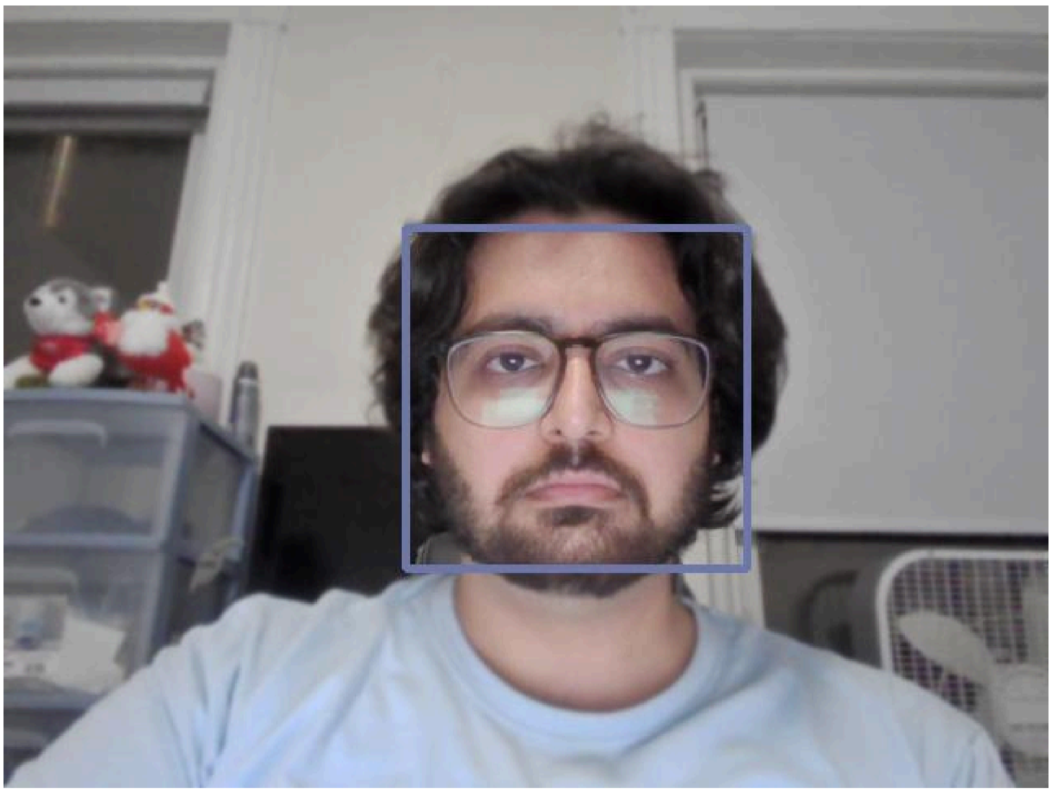


Three More Effects

1. Blur Background



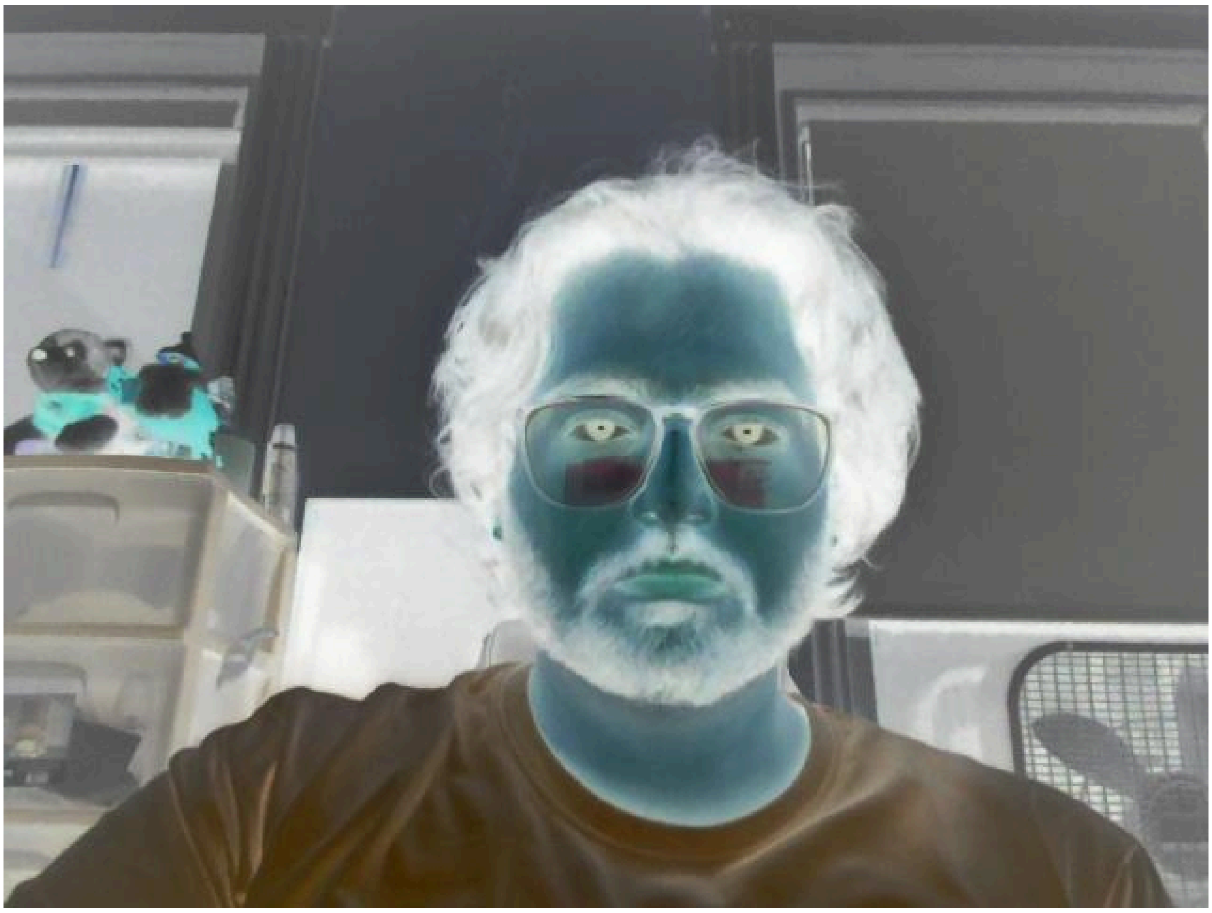
Original Image



Background Blurred

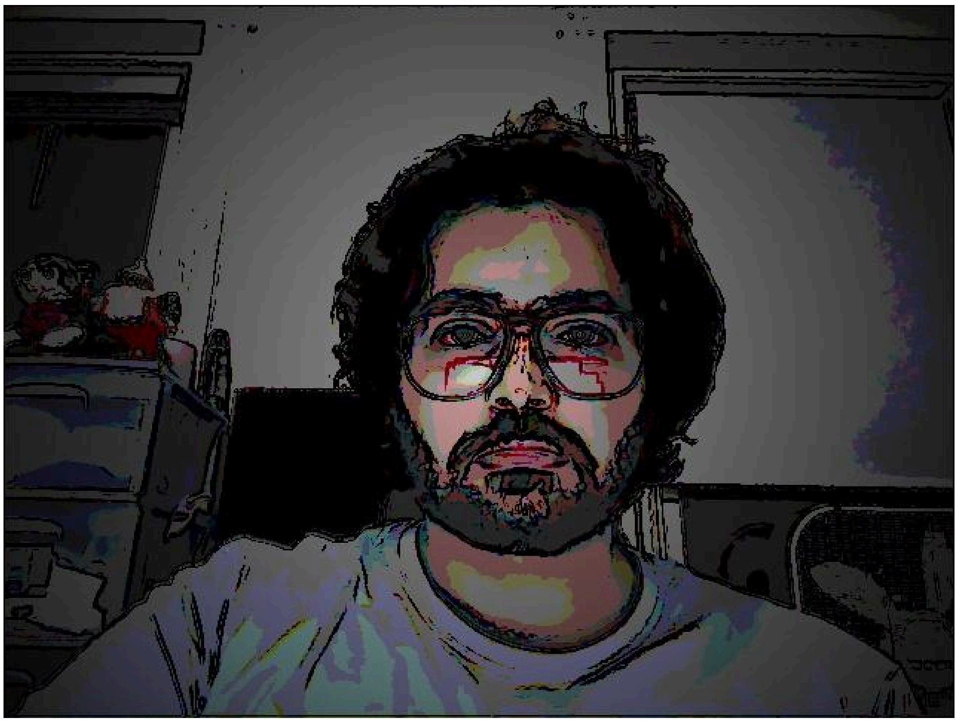
In the second image, we can see the husky in the background looks blurred.

2. Negative Image



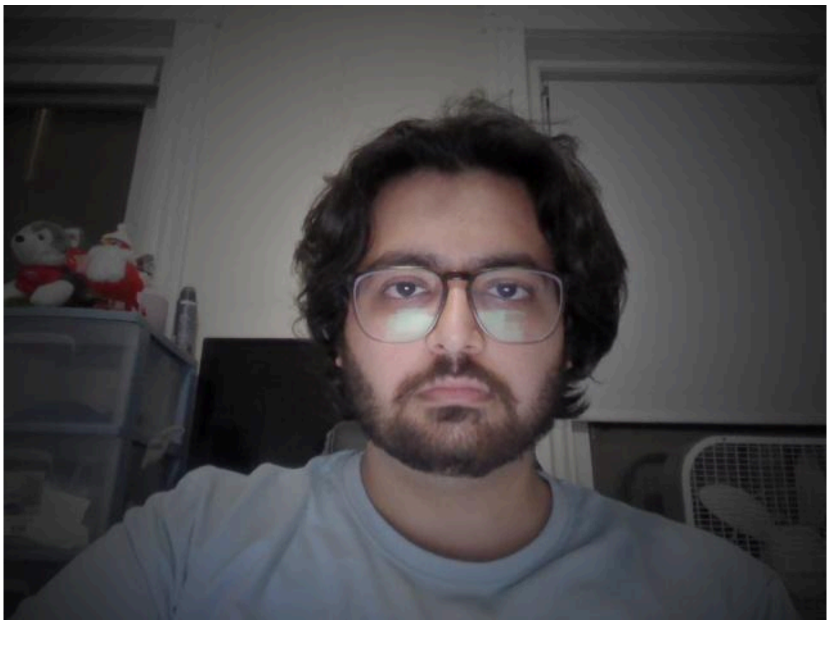
This image is similar to my custom grayscale but you can see the colors in this one whereas in the other one is fully grayscale

3. Cartoonization



Extensions 3 and 4

For one of the extensions, I added a feature to add a vignette effect on top of any filter or on its own. The vignette filter essentially changes the intensity of pixels as you move farther from the center. The amount of intensity change depends on the strength of the filter.



Even the cartoon filter above has the vignette enabled.

For the last extension, I added the functionality to record video with any filters enabled. Pressing ‘r’ starts the recording and it can be ended by pressing ‘r’ again. Link to a recorded video is in the readme file.

Learnings and Takeaways

From the given tasks I learned the following-

1. Reading and writing images
2. Accessing pixels and individual channels
3. Applying filters - in built and custom
4. Convolution - 1D and 2D
5. Introduction to Haar cascade

From the extension I learned the following -

1. OpenCV data types
2. OpenCV dot and cross products and their efficiency
3. OpenCV slicing
4. Right way to handle cv Mats
5. How to make a vignette filter
6. Video writing

Acknowledgements-

1. OpenCV documentation - https://docs.opencv.org/4.5.1/
2. Stack Overflow for troubleshooting