Danny Chinn (NUID: 001699303) and Priyanshu Ranka (NUID: 002305396)

23 January 2025

CS5330 Spring 2025

Professor Bruce Maxwell

Project 1 Report

1. Project Overview

- This project focuses on developing image and video processing applications using OpenCV. The primary tasks include loading and displaying images, live video processing with grayscale conversion, applying filters like blur, Sobel, and sepia tone, and implementing advanced features such as face detection and depth estimation using ONNX Runtime. The goal is to understand real-time processing, optimization techniques, and creative customization for various effects.

2. Task Demonstration and Results

2.1. Read an image from a file and display it

- This task involves loading an image file from disk and displaying it using OpenCV. It demonstrates basic file I/O operations in OpenCV, allowing the user to view the loaded image in a dedicated window. This forms the foundation for subsequent image-processing tasks.

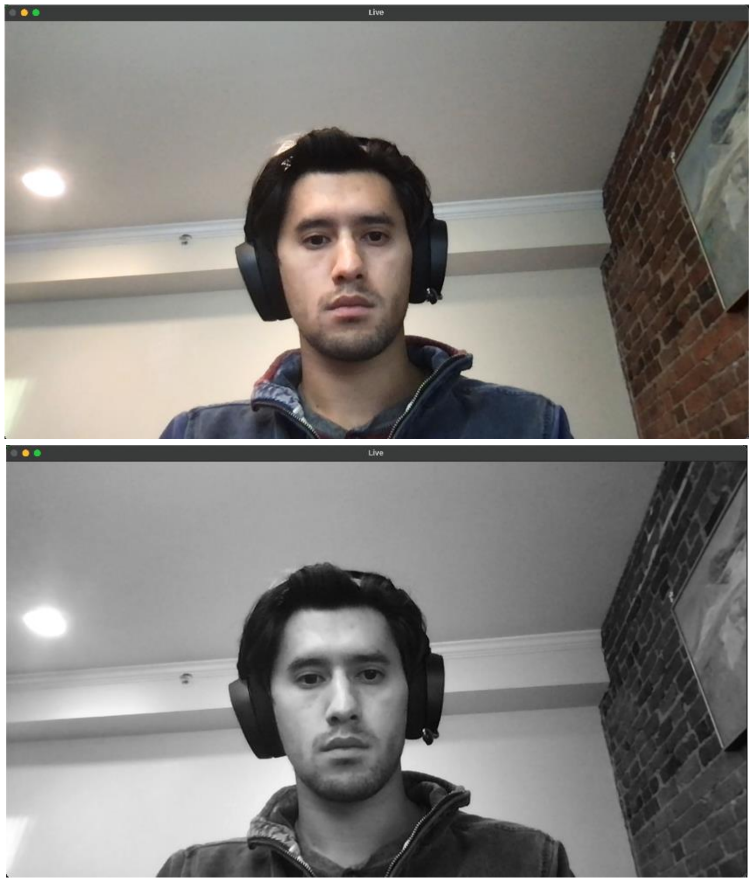
-

2.2. Display live video

- This task focuses on capturing video frames from a webcam and displaying them in real time with function “defaultGreyscale”. It demonstrates live video streaming, forming the groundwork for real-time processing. There are also additional features – pressing ‘q’ quits the program and ‘s’ saves the frame at that instant.

2.3. Display greyscale live video

- This task converts a video stream, through function “greyscale”, into greyscale using OpenCV’s built-in ‘ctvColor’ function. If the user types 'g' it displays a greyscale version of the video stream instead of color. Pressing ‘g’ key switches it back to color stream. This uses the OpenCV’s ctvColor function to do so.

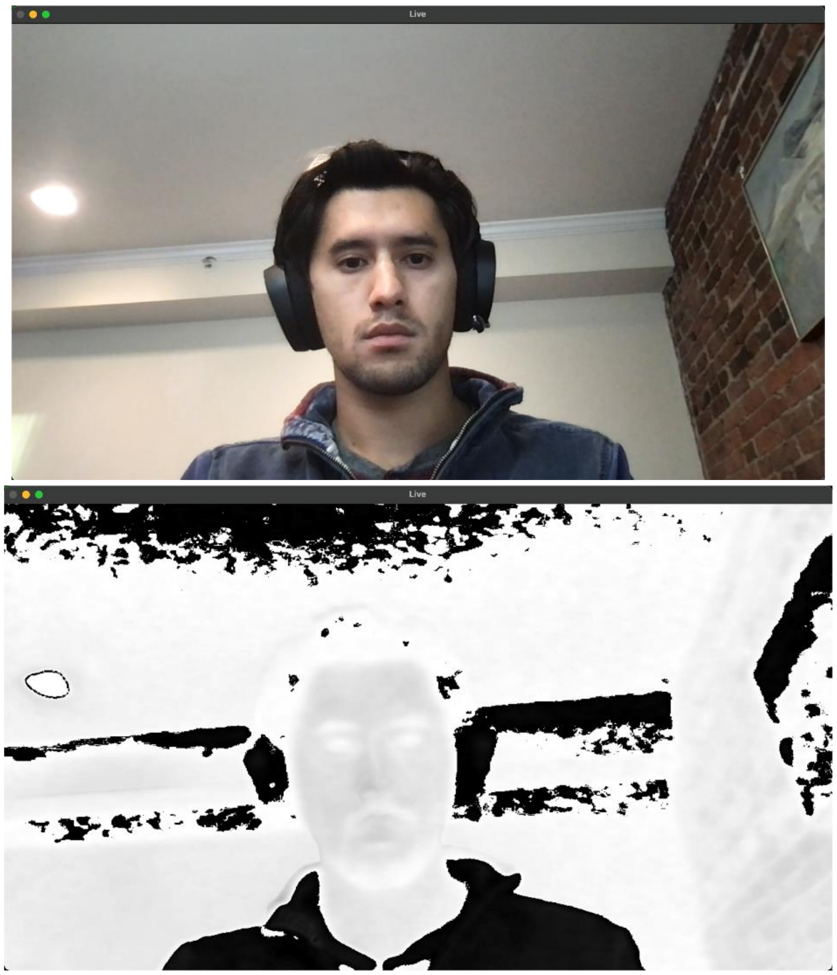


Required Image 1: Original v/s Grey-scaled using ctvColor function

2.4. Display alternative greyscale live video

- A custom implementation of greyscale conversion without OpenCV’s built-in functions. This task reinforces pixel-wise operations and the understanding of the greyscale formula. The formula used in this function is

$$grey = (blue\_tmp + green\_tmp) / 2 - red\_tmp$$



Required Image 2: Original v/s Grey-scaled using a custom function

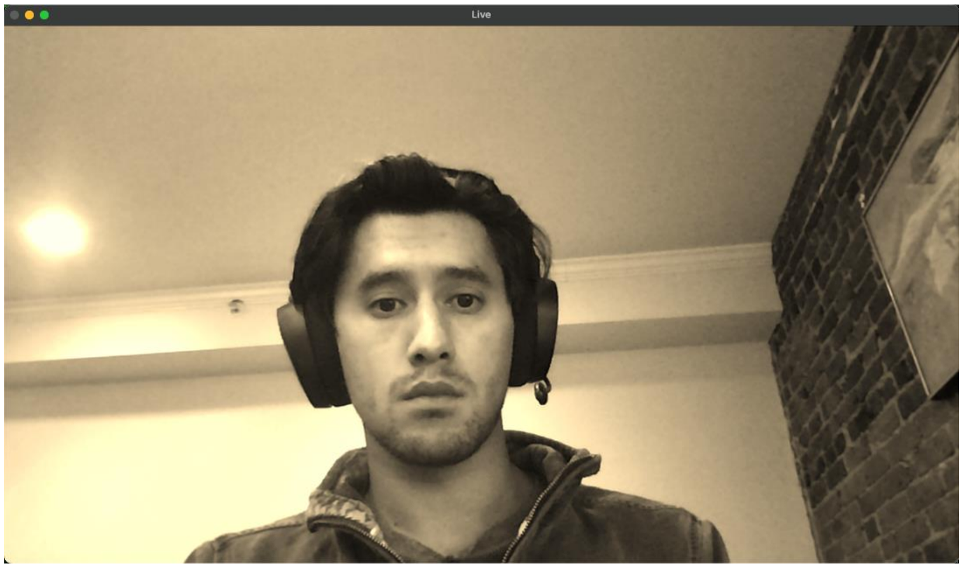
Aspect	ctvColor	Custom Formula
Weighting	Weighted sum: 0.299R, 0.578G, 0.114B	Equal for G and B, less for R
Accuracy	Mimics human brightness perception	Stylish or Customized
Effect on Red Channel	Red adds brightness	Red areas are dimer/darker

Table: Differences between Default (Task 2.3) and Custom Greyscale (Task 2.4) functions

2.5. Implement a Sepia tone filter

- This task applies a sepia filter to the live video with function “filterSepia”. By manipulating color values through a transformation matrix, it gives frames a warm, vintage appearance. The new color values are according to the matrix:

```
[ 0.272, 0.534, 0.131    // Blue coefficients
  0.349, 0.686, 0.168    // Green coefficients
  0.393, 0.769, 0.189 ]  // Red coefficients
```



Required Image 2: Application of Sepia tone Filter

- Ensuring Use of Original RGB Values

In the sepia transformation, the original RGB values are directly used in the computation because:

1. The sepia kernel explicitly relies on a linear combination of the original R, G, and B values.
2. Each channel is computed independently using the kernel, ensuring no unintended overwriting or loss of the original data.

2.6. Implement a 5x5 blur filter

Part A. This task applies a basic blur effect using a convolution operation over a kernel in a single nested loop, within function “blur5x5_1”. The filter used here is

Gaussian [1 2 4 2 1; 2 4 8 4 2; 4 8 16 8 4; 2 4 8 4 2; 1 2 4 2 1]

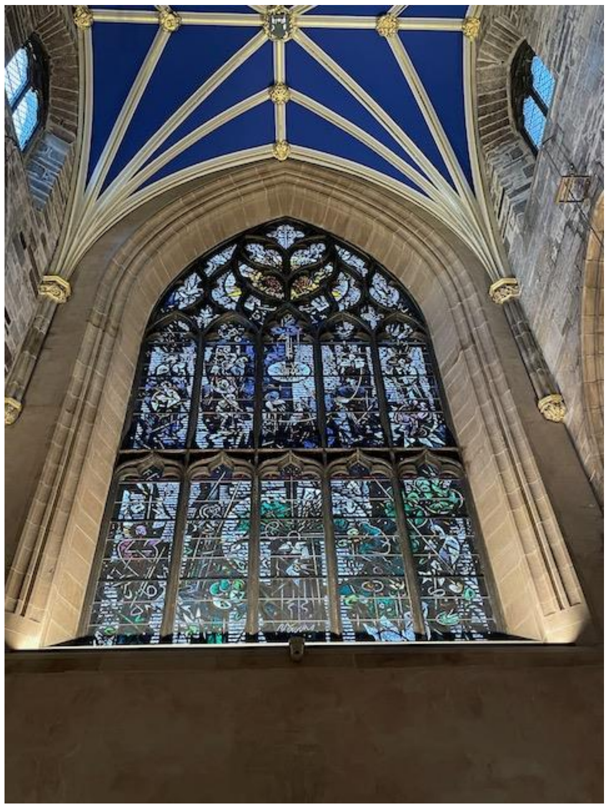
In this we are implementing *same convolution* that is we are not changing the size of source video/input. Keypress ‘u’ is used to utilize this filter.

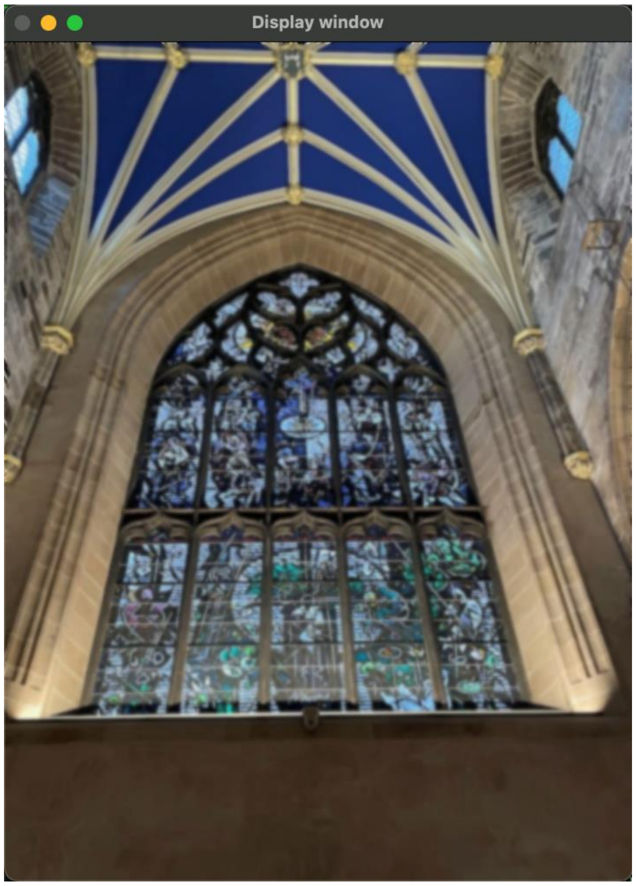
Using the timeBlur program, this naive implementation took .1229 seconds on the cathedral.jpeg image displayed below.

Part B. For this part, the same filter is applied. However, instead of a single 5x5, two separable filters are used: a horizontal and vertical [1, 2, 4, 2, 1] filter.

Keypress ‘b’ is used to utilize this filter.

Using the timeBlur program, this naive implementation took .0655 seconds on the same image. This is nearly twice as fast as the naive implementation, without using pointers.





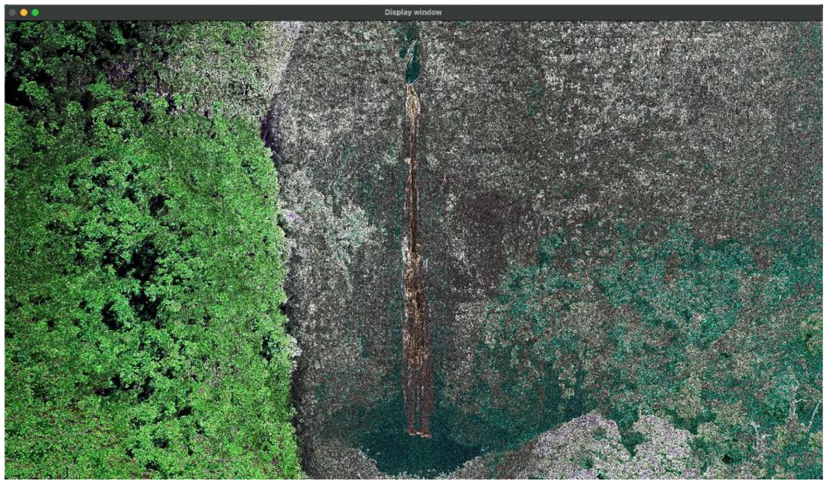
Required Image 3: Output of blur5x5_2

2.7. Implement a 3x3 Sobel X and 3x3 Sobel Y filter as separable 1x3 filters

- Building upon the naive blur, this task optimizes the blurring process by separating the kernel into two 1D operations. It significantly reduces computational complexity. Keypress ‘x’ is used for SobelX filter and Keypress ‘y’ is used for SobelY filter.

2.8. Implement a function that generates a gradient magnitude image from the X and Y Sobel images

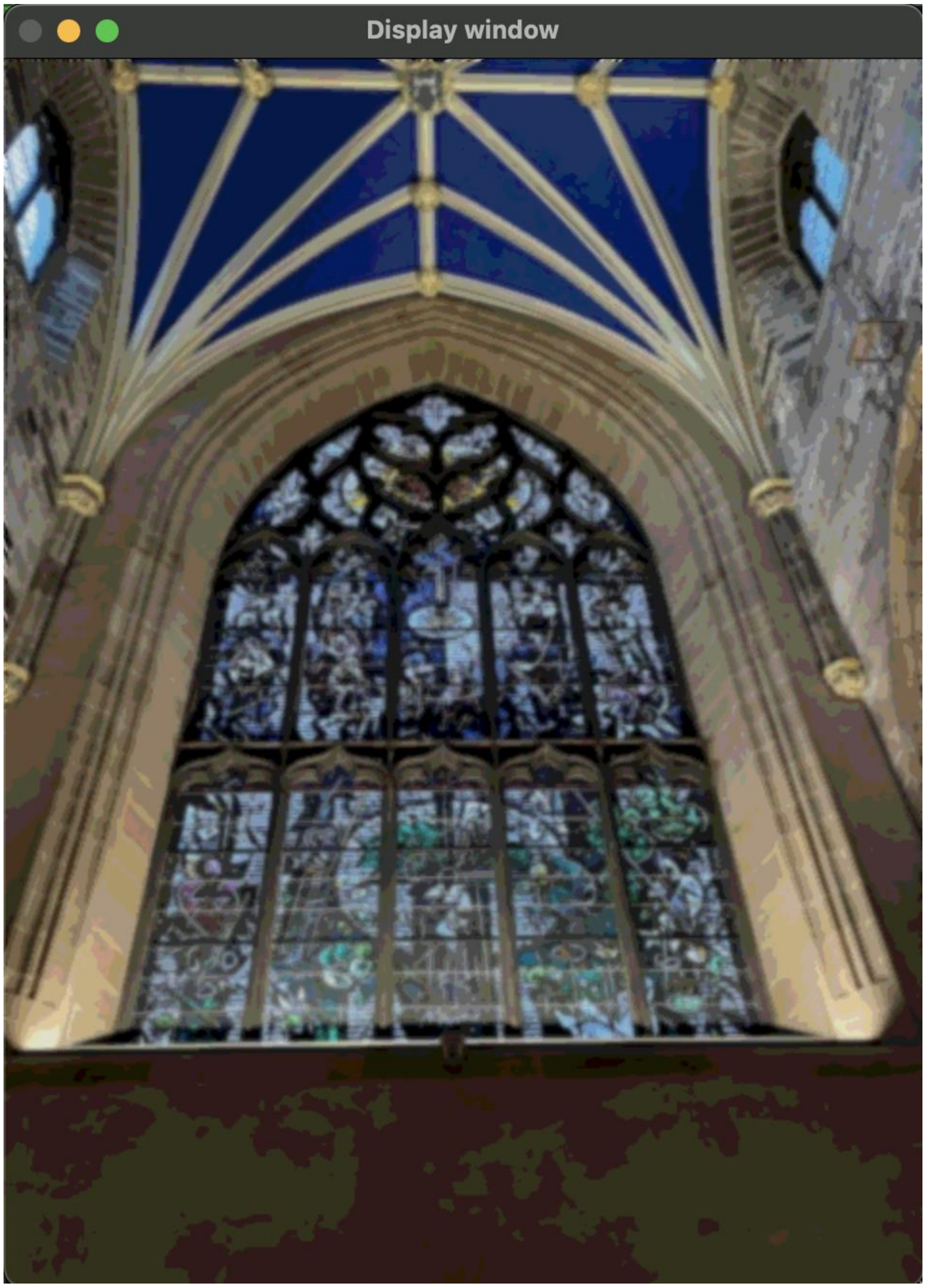
- This task applies Sobel filters to detect horizontal and vertical edges in an image, calculating the gradient magnitude to highlight areas of intensity change, crucial for edge detection.



Required Image 4: Output of magnitude filter

2.9. Implement a function that blurs and quantizes a color image

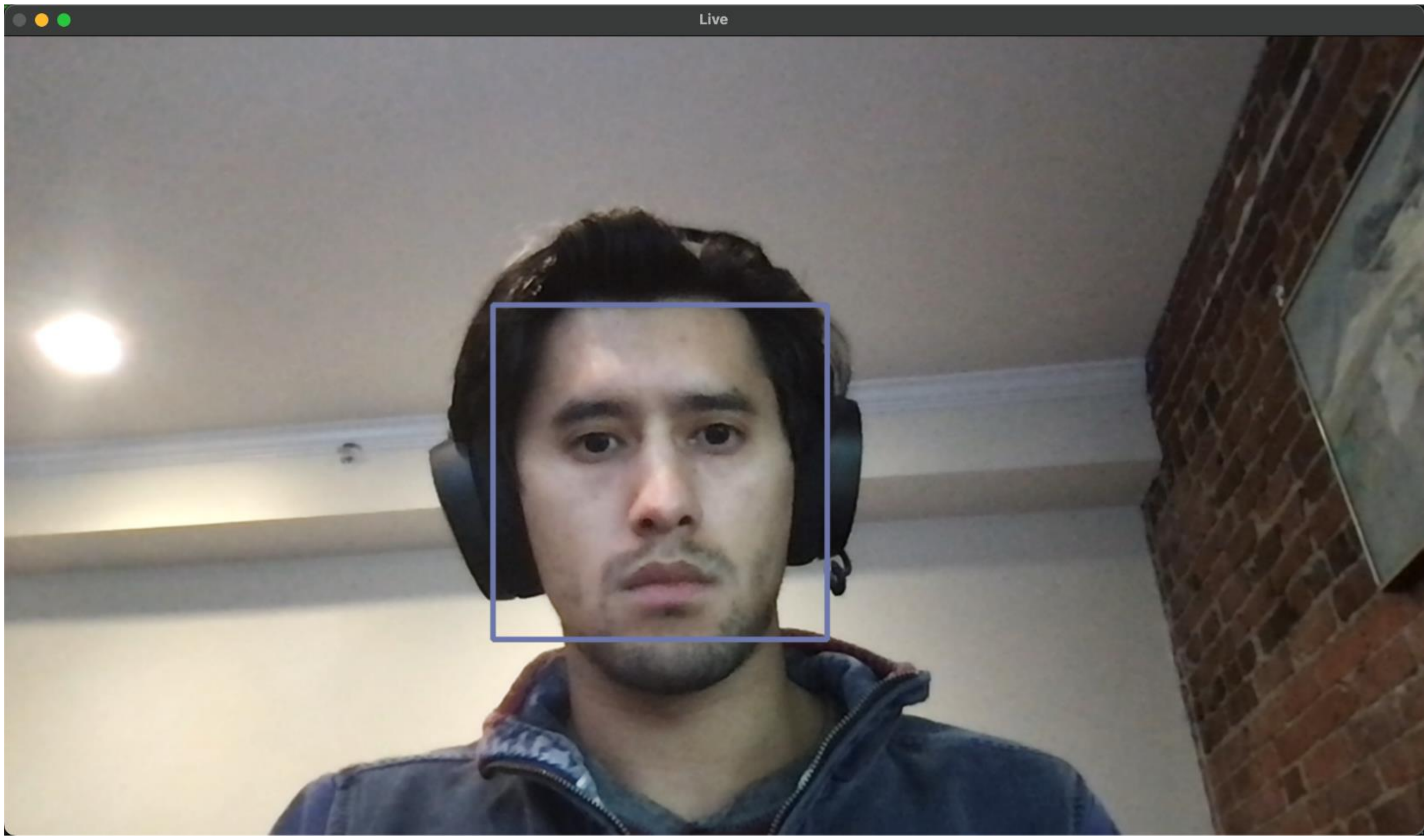
- Combines the second blur filter with quantization, reducing the number of intensity levels in the image. This results in an abstract, artistic effect while demonstrating image simplification.



Required Image 5: Blurred + Quantized Image (original same as cathedral

2.10. Detect faces in an image

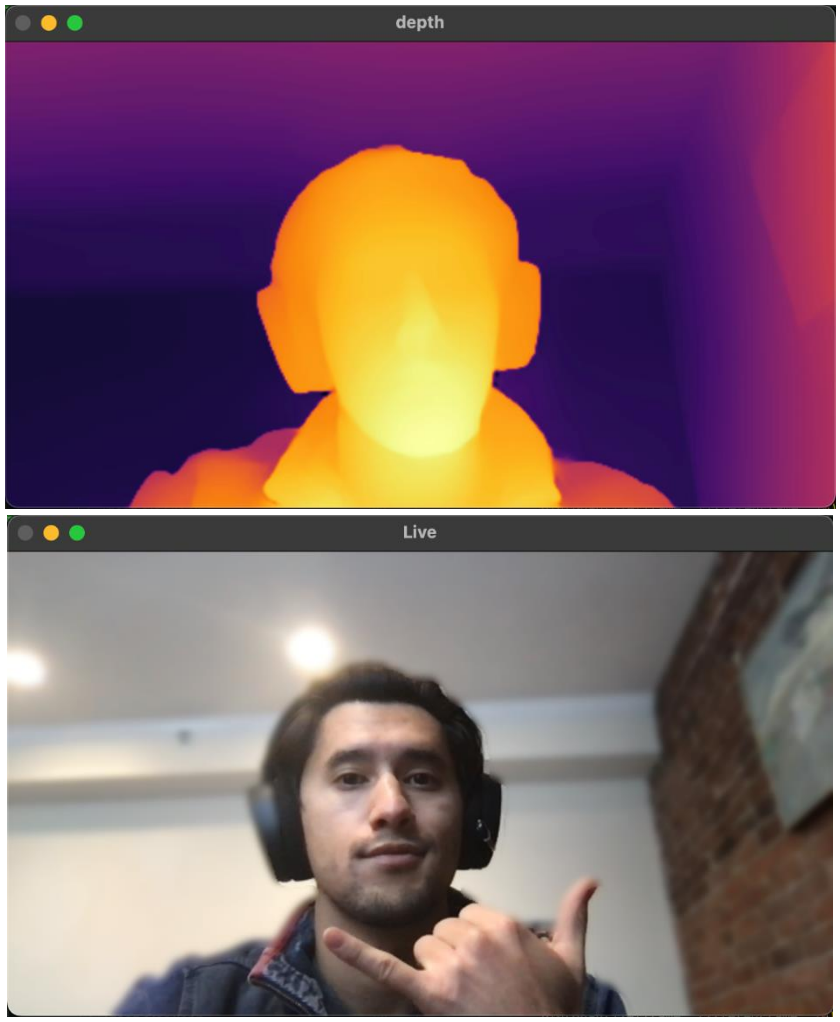
- Using a pre-trained face detection model, this task detects faces in images or live video. It demonstrates machine learning integration for real-time object detection. It draws a localization for all ROIs detected.



Required Image 6: Face detection

2.11. Use the Depth Anything V2 network to get depth estimates on your video feed

- This task involves estimating depth information from an image using a pre-trained ONNX model. It highlights 3D perception, useful for applications like augmented reality or robotics. We implemented a custom filter using the DA2 output values to blur the background (any depth values over a certain threshold). We smooth the depth map values and apply a 15x15 Gaussian blur filter. We use OpenCV's bitwise operator functions to combine the different masks.

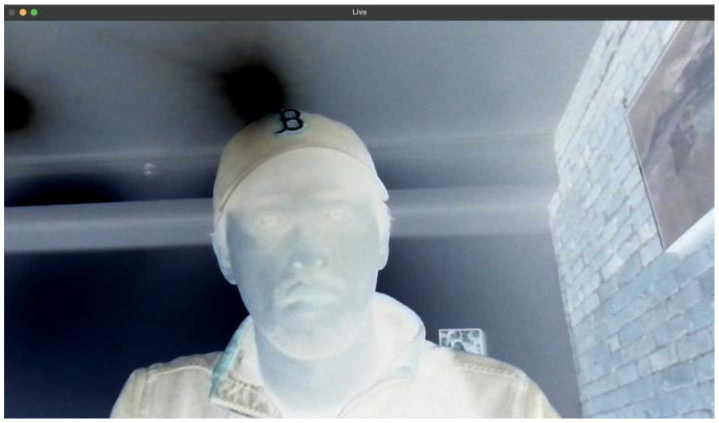


Required Image 7-8: Depth example and custom background blur

2.12. Implement three more effects on your video

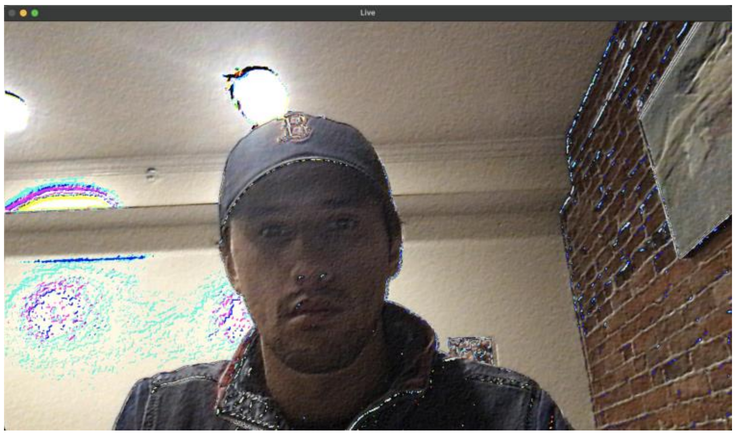
- An optional task that involves creating and implementing a unique filter or effect. The filters implemented are as follows:

A. *negativeFilter()* – This is a pixel-wise filter that creates the negative of the live video stream. This simply subtracts the current channel values from 255 to do so. Keypress '1' is used for this filter.



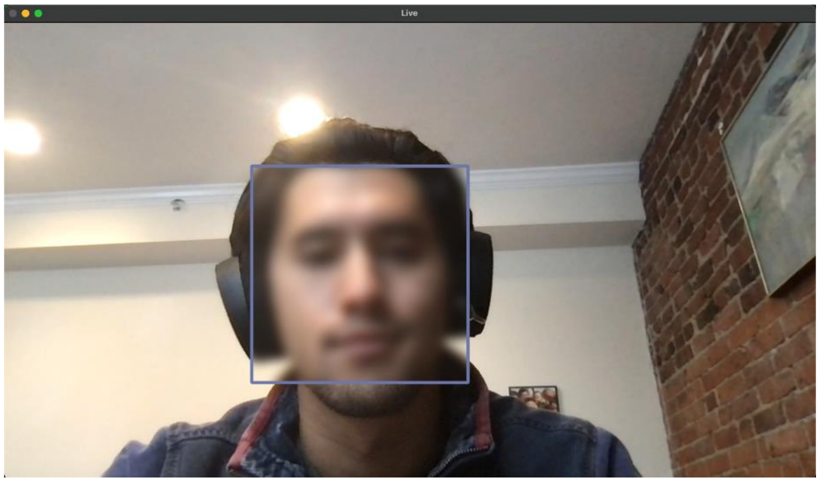
Required Image 9: Negative Filter Output

B. *embossFilter()* - Creates a 3D-like effect by highlighting the edges in a way that mimics raised surfaces. This is a kernel-based filter. The kernel used is [-2, -1 , 0; -1, 1, 1; 0, 1, 2]. This kernel highlights edges by creating light and dark areas that simulate depth. Keypress '2' is used for this filter.



Required Image 10: Emboss Filter Output

C. faceRedactor*()* - Uses Cascade Classifier to identify faces (from provided example), then applies a 51x51 Gaussian blur filter to those ROIs. Keypress '3' is used for this filter.



Required Image 11: Face redactor

2.13 EXTENSIONS

A. All filters can be applied to both images and the video stream using the same key presses. Pressing the same key twice will undo the filter and revert to the source image.
B. Video stream can be recorded using OpenCV's VideoWriter class. Filters can also be applied/removed during the recording process.
C. Loading and saving images, as well as saving video recordings, are now handled via tinyfiledialog. This removes the tedious requirement of specifying file paths via CLI and contributes to a more interactive user experience.
D. An additional blur5x5 filter added for even smoother performance during video streaming. Blur time for cathedral.jpeg image: .0026 seconds. Additional warm and cool filter, and a median filter were added as part of an extension.

A link to videos demonstrating the recording and loading/saving extensions can be found in the readme.txt file.

3. Reflection/ Learning

The learnings from this project can be summarized as under-

- Learned how to *manipulate* and *process images* using OpenCV, including basic operations like image loading, resizing, and applying filters.

- Gained hands-on experience in implementing filters using *mathematical kernels* and performing matrix transformations for effects like *sepia* and *grayscale*.
- Explored *OpenCV functions* like ‘cv::transform’, ‘convertScaleAbs’ and others, understanding their role in optimizing performance for image processing tasks.
- Learned how to debug common issues like *type mismatches* (e.g., float vs. integer matrices) and out-of-bound pixel values while implementing filters.
- Applied mathematical concepts such as *convolution*, *linear transformations*, and *pixel intensity manipulation* to real-world problems.
- Recognized the importance of writing *modular and reusable code* by encapsulating filter operations into separate, well-defined functions.
- Deepened my understanding of how the RGB channels interact and contribute to different visual effects, such as *color inversion, grayscale, and sepia*.
- Understood the significance of choosing appropriate *data types* (e.g., float vs. integer) and optimizing matrix operations to improve *runtime performance*.
- Realized how these concepts could be applied in fields like photography, augmented reality, and video editing for *creating unique visual effects*.
- Enhanced my *problem-solving skills* by experimenting with custom filter implementations and creatively combining mathematical operations for desired effects.

4. Acknowledgements

This project used resources such as OpenCV documentation, ONNX Runtime tutorials, and class lectures. It also utilized the help from ChatGPT and Google search engine to solve some unforeseen errors and code snippets where the problem couldn’t be solved from other sources available at that time. Special thanks to Prof. Bruce A. Maxwell and TAs for their guidance.

5. References