CS 5330: Pattern Recognition and Computer Vision - Project 1

(Piyush Goel)

Short Description

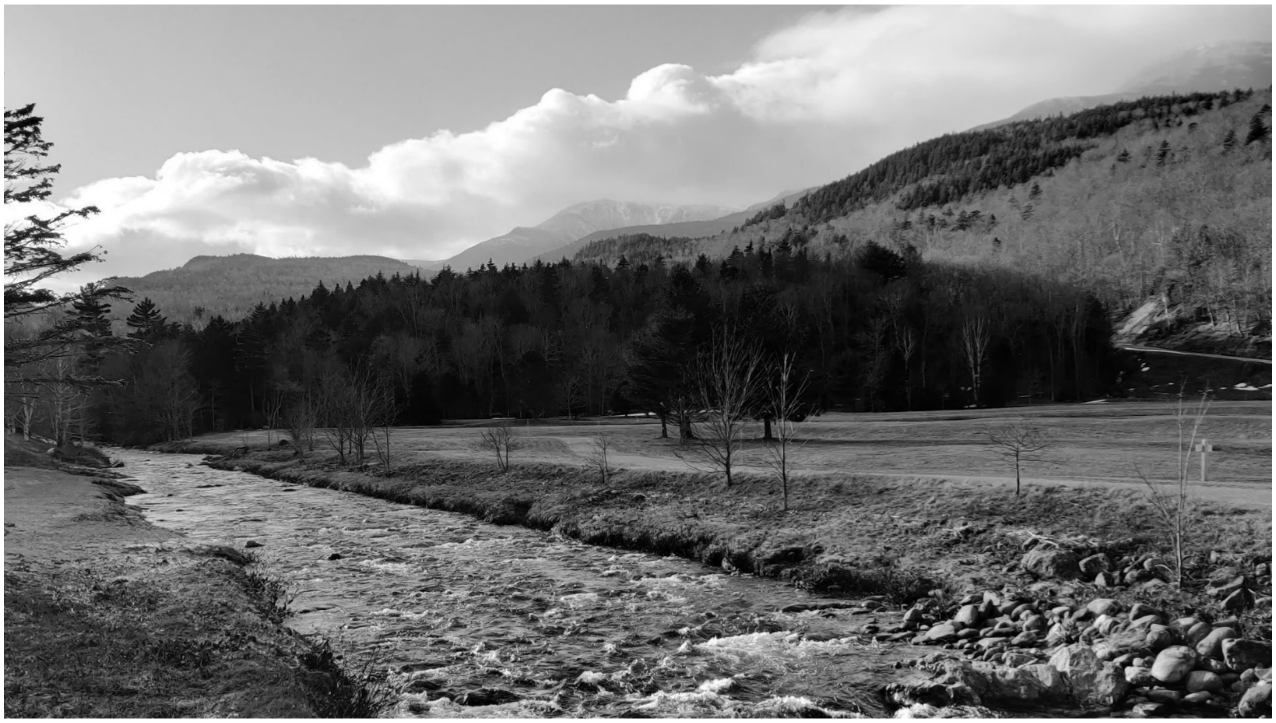
In this project I worked on implementing various image filters and effects in C++ from scratch using just the basic classes and functions from the opencv framework. Then I also worked on giving the ability for the user to be able to apply all these features on live video from the webcam by pressing specific keys. The user can also combine multiple filters on top of each other by toggling on multiple filters at the same time without turning off the previous ones. Though this might heavily lower the fps of the video.

All the filters which require convolution use the wrapping technique for the border pixels.

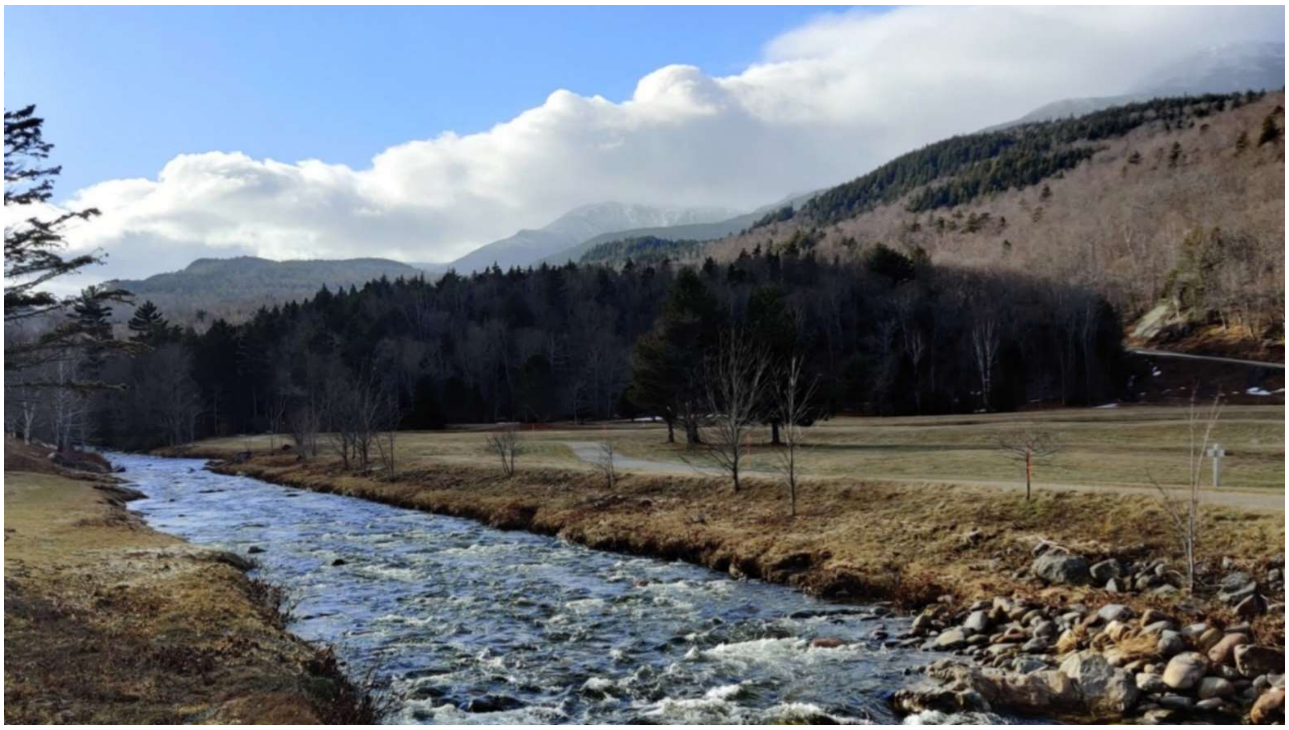
Required Images

1. ORIGINAL IMAGE (This is the image is the one on which all the effects are being applied.)

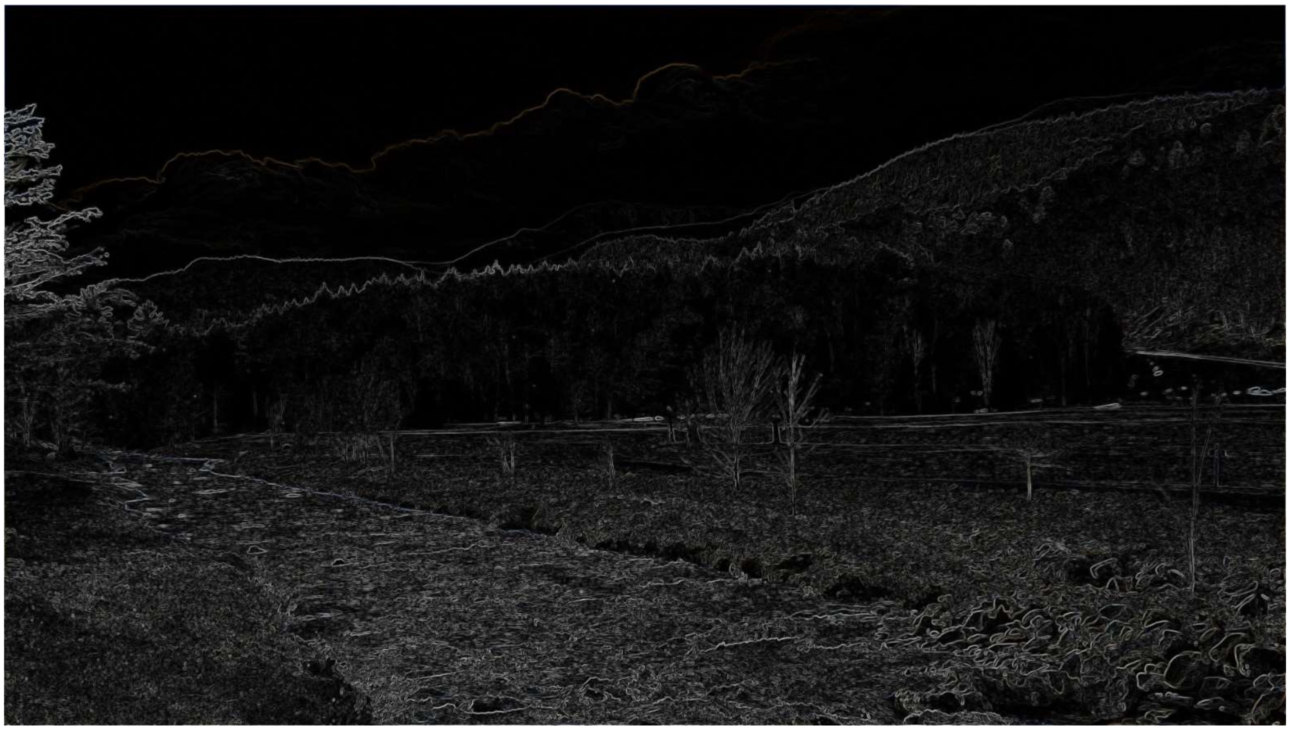
2. REQUIRED IMAGE 1 - GREYSCALE IMAGE



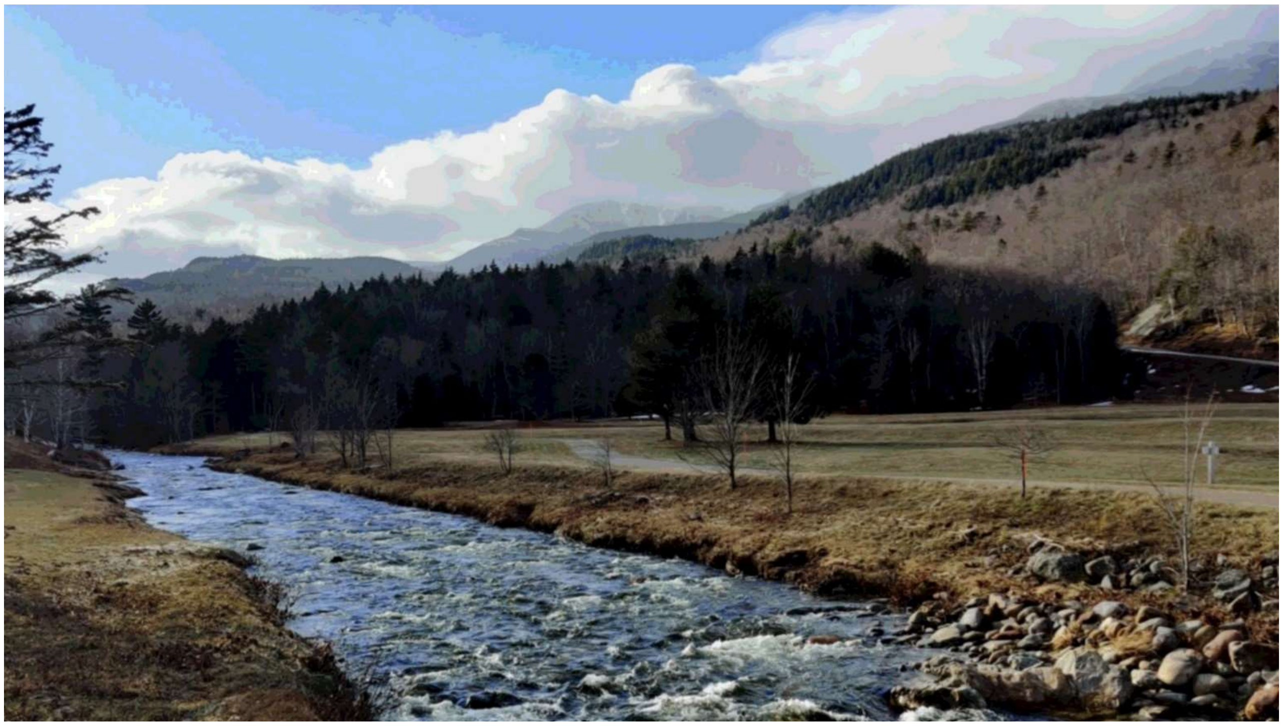
3. REQUIRED IMAGE 2 - BLURRED IMAGE (Since the image here is shrunk down to a smaller size to fit on the page, the blurriness is not clearly visible, but you can see some hints of it in the detail in the grass or the water. But to be able to see the difference clearly see the my_image.jpg and blurred.jpg files within the SavedImages folder.)



4. REQUIRED IMAGE 3 - SOBEL GRADIENT IMAGE

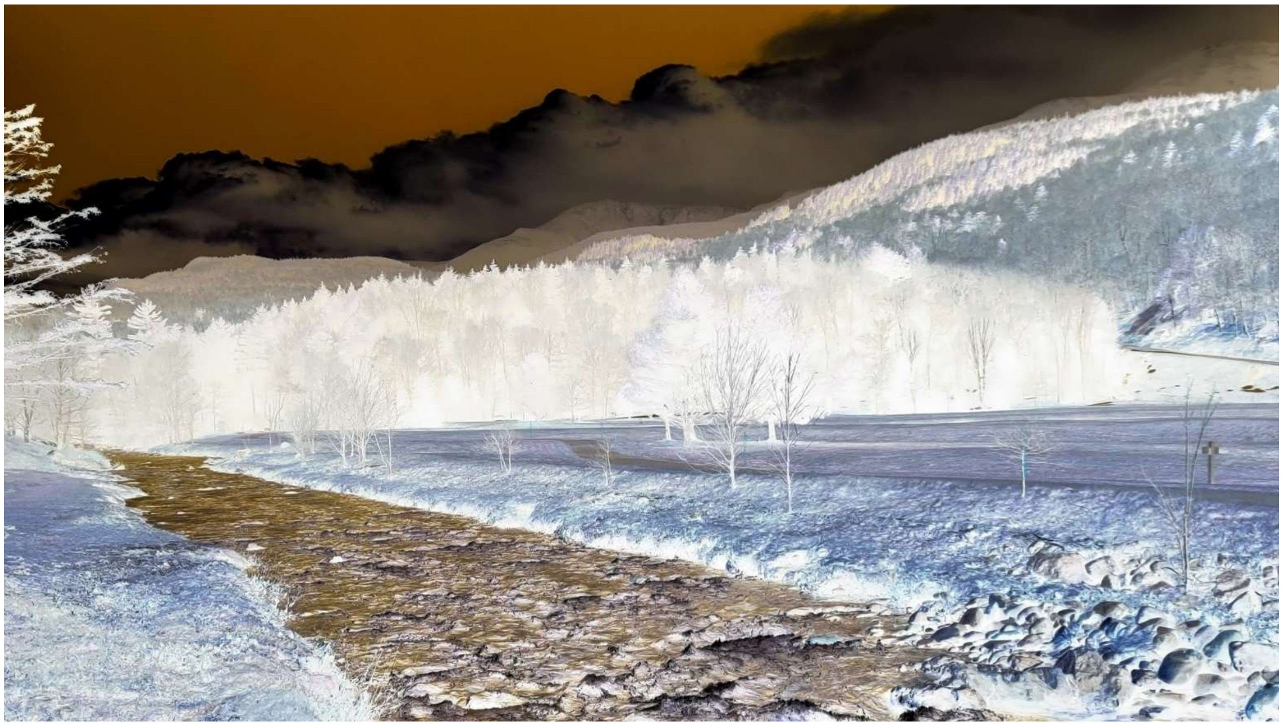


5. REQUIRED IMAGE 4 - BLUR QUANTIZE IMAGE (with the number of levels as 15. Again as was the case with the blurring effect to fully appreciate the difference open up the blurquantize_15.jpg file, but here we can see a major difference due to the effect in the clouds and the sky.)



6. REQUIRED IMAGE 5 - CARTOON IMAGE (with 15 as the blur quantize level and 20 as the Sobel magnitude threshold)

7. REQUIRED IMAGE 6 - NEGATIVE IMAGE (every color (RGB) of every pixel is replaced with 255 minus it's original value.)

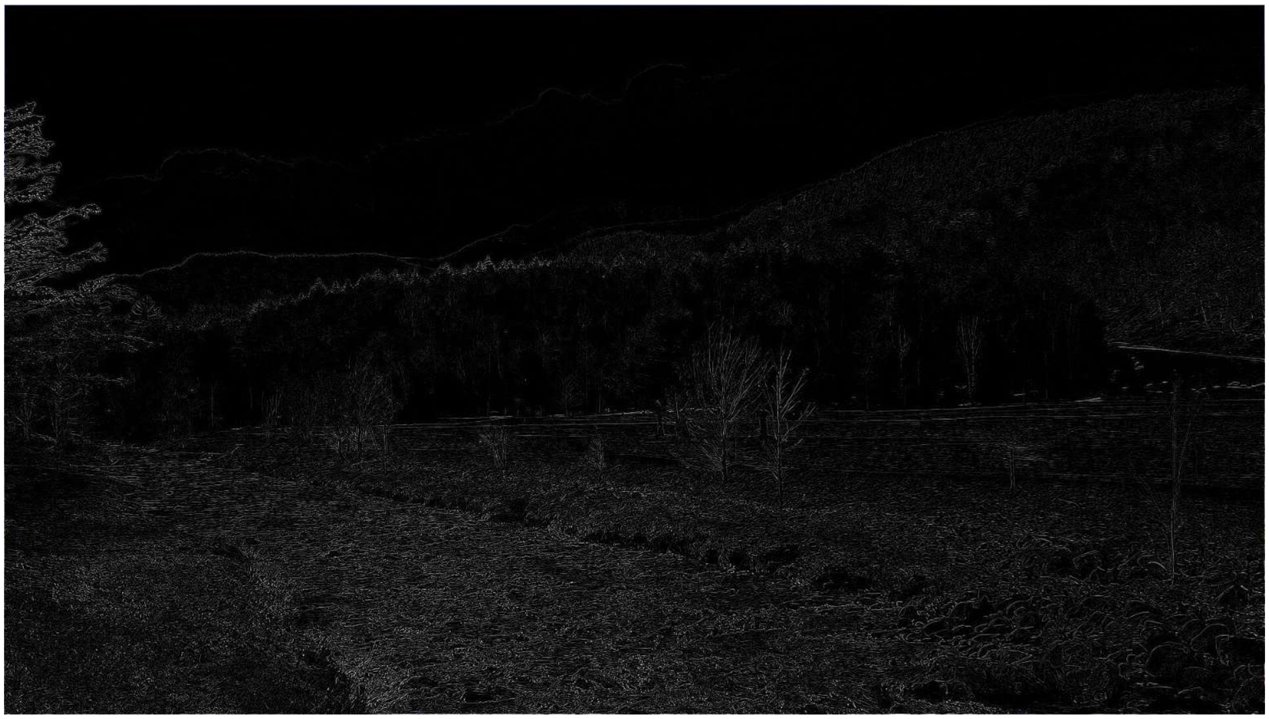


Extensions

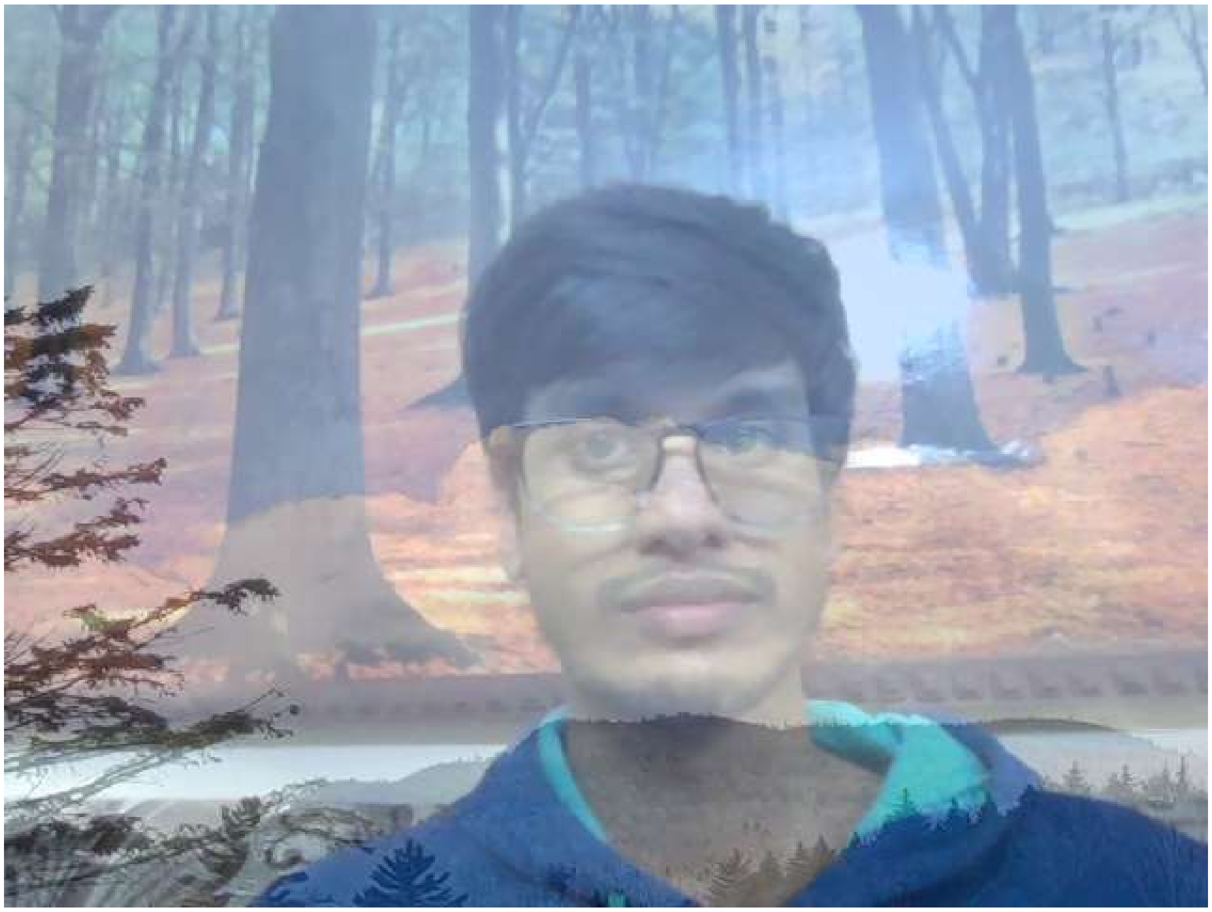
1. The ability to negate any color combination out of RGB, i.e. the ability to apply the **p' = 255 – p** computation to any set of colors the user choses. Here are some examples

2. Ability to adjust the brightness and contrast of the image. The contrast and brightness of an image are adjusted by applying the following equation on every color (RGB) of all the pixels of the image **p' = a*p + b** where p' is the color value of the new pixel, p is the old color value, a is the parameter which helps us adjust the contrast, b is the parameter which helps us adjust the brightness. Example image with a = 3.0 and b = 100.

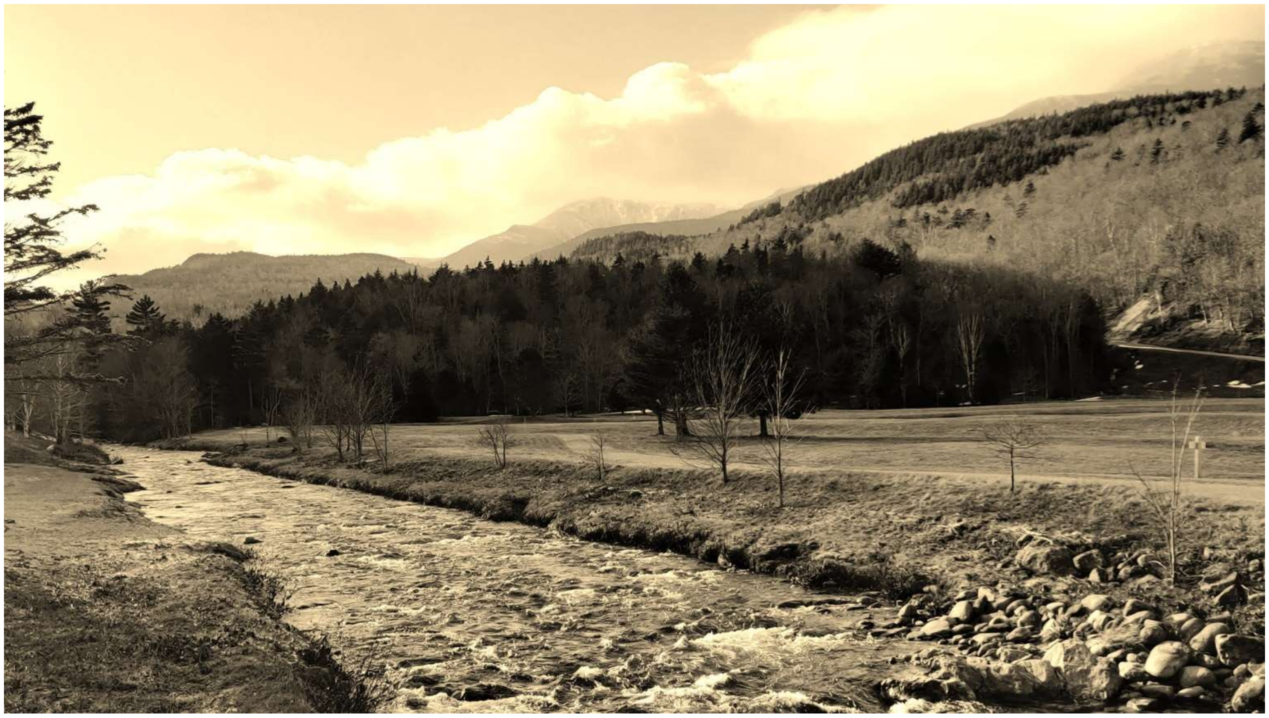
3. 3x3 Laplace Filter (from scratch) with the matrix [[0, -1, 0], [-1, 4, -1], [0, -1, 0]] as the filter.



4. An effect to combine/mix multiple images in any ratio from 0.0 to 1.0. Example image with the original image from above mixed with my photo from the webcam.

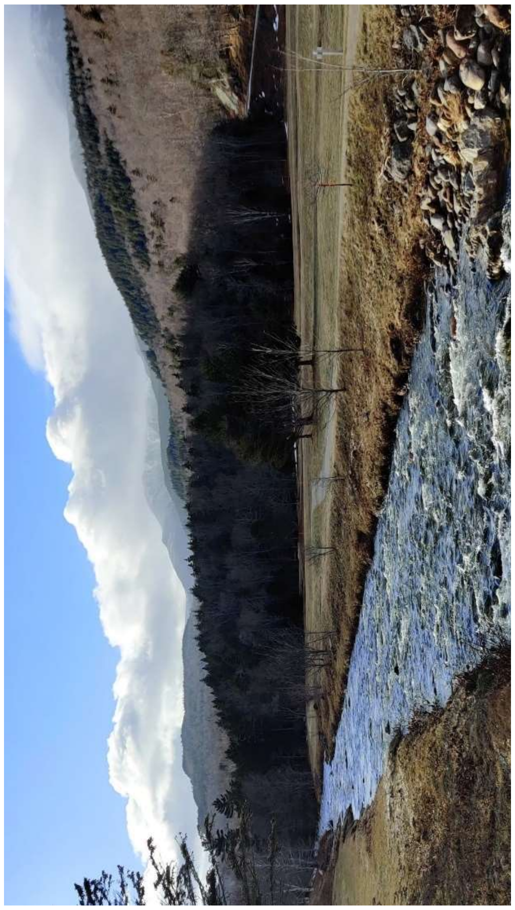


5. Sepia Effect using the following equations:
   R’ = 0.393 * R + 0.769 * G + 0.189 * B
   G’ = 0.349 * R + 0.686 * G + 0.168 * B
   B’ = 0.272 * R + 0.534 * G + 0.131 * B

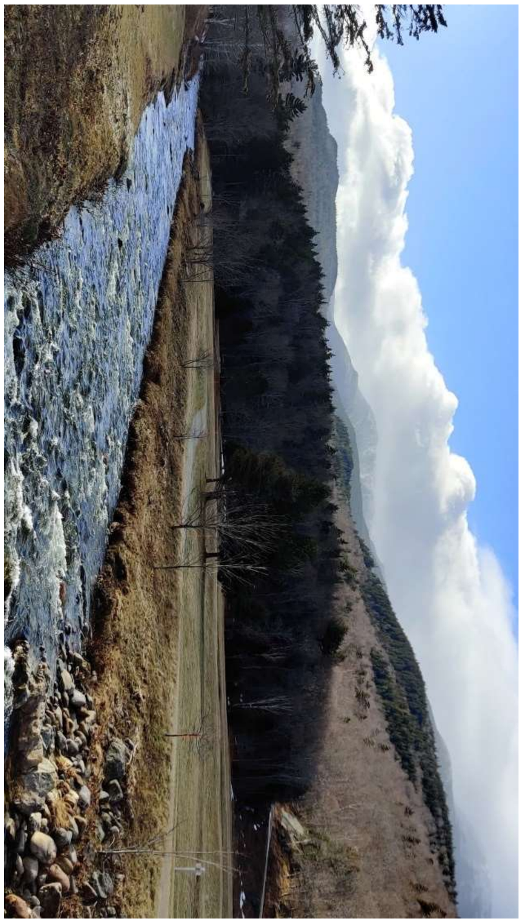


6. Mean Blur (with varying filter size) Example images with filter size 5 and 7 respectivey. (These images don’t look too blurred here despite the large filter size because these are originally pretty large images so to be able to actually see the effect of blurring look at the full images (meanBlur2.jpg and meanBlur3.jpg) in the SavedImages folder.)

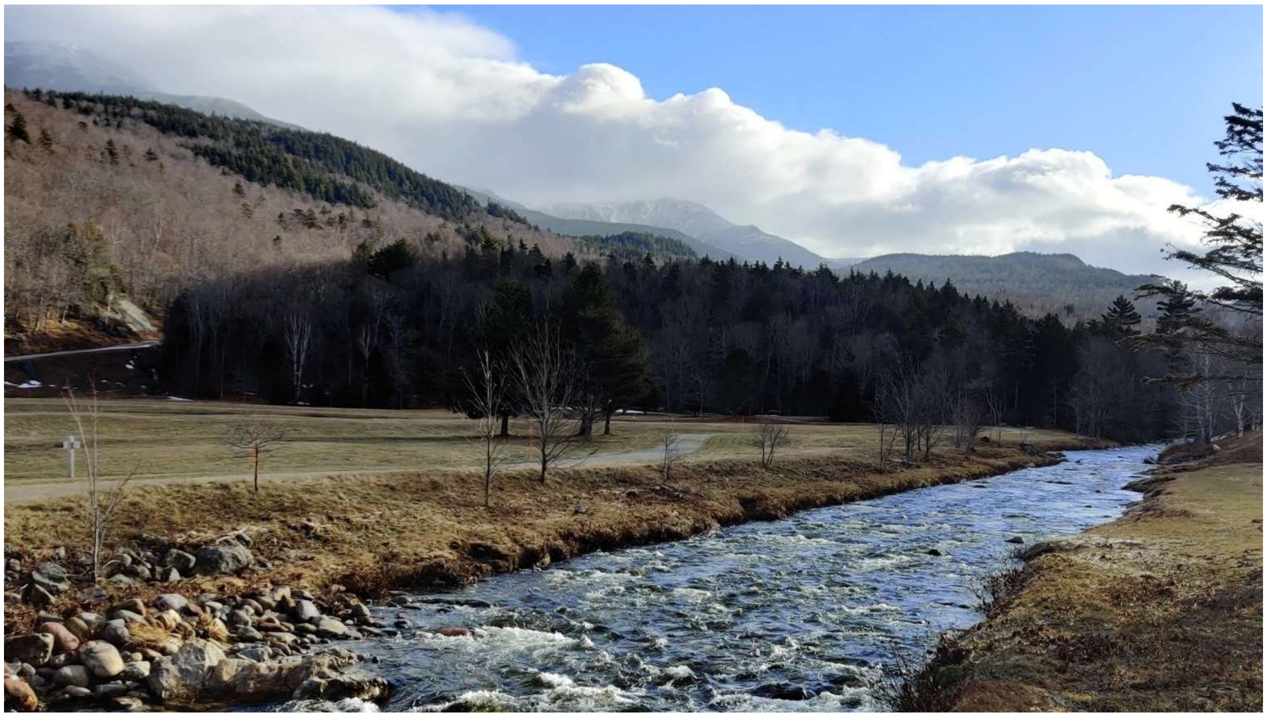
7. Rotate the Image Anti Clockwise (by rotating the whole pixel matrix one pixel at a time.)



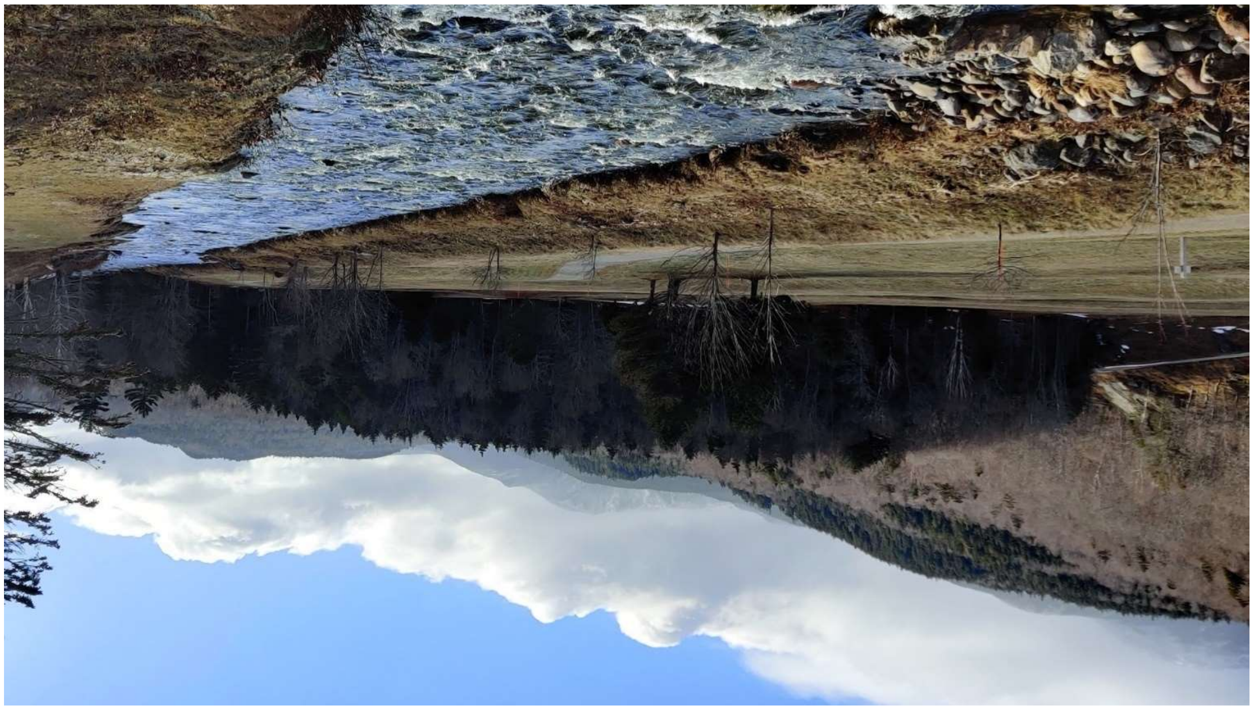
8. Rotate the Image Clockwise (by rotating the whole pixel matrix one pixel at a time.)



9. Mirror Image (by flipping the whole pixel matrix one pixel at a time.)



10. Upside-down Image (by flipping the whole pixel matrix upside down one pixel at a time.)



Reflection

1. Being completely new to C++, I started to learn some details of the language along with its syntax by working through this project.
2. Implementing the filters from scratch, I learned a great deal about them.
3. I learned the importance of abstracting out common code. When I first implemented the blur filter I did not implement convolution as a function independent of the details of the blurring function. Then while implementing the Sobel filters I needed to implement and debug the whole thing again. So, instead I factored out convolution of two matrices as a separate function independent of the filter. This also helped me later on when trying to implement other filters which made use of convolution, for example I could implement the Laplace filter pretty quickly thanks to that.
4. I also learned to work with OpenCV Mat objects, accessing and manipulating the individual pixel values and also how to mess around with the data types.

Resources

1. Class lecture
2. Stack Overflow
3. OpenCV documentation
4. Google searches
5. Pic Credits: Me – White Mountain National Forest, NH.