Computer Vision Assignment 7

Kai-Po Lin (kl72); Department of Computer Science

1. Install OpenCV

The website I followed for installation: https://www.geeksforgeeks.org/how-to-install-opencv-4-on-macos/

(1) **The process of installing OpenCV 4.5.5.64**

⇨ Since I already installed miniforge before for my mac (by following the instructions here: https://www.mrdbourke.com/setup-apple-m1-pro-and-m1-max-for-machine-learning-and-data-science/), downloading OpenCV isn't a hard thing for me. All I do is follow method 1 in the geeksforgeeks.org webpage above and install it with my "pip".

⇨ Steps I installed OpenCV:

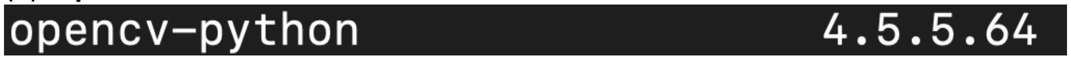
(a) **Activate my virtual environment:** conda activate kb
(b) **Upgrade pip to the latest version:** pip3 install --upgrade pip
(c) **Install OpenCV:** pip3 install OpenCV-python

(2) **miniforge v.s miniconda (Cited from:** https://stackoverflow.com/questions/60532678/what-is-the-difference-between-miniconda-and-miniforge**)**

⇨ miniforge is for macOS aarch64 (arm64) chips (M1, M1 Pro, M1 Max) because miniconda doesn't support aarch64 (arm64). Moreover, there are also miniforge versions for all Linux architectures.

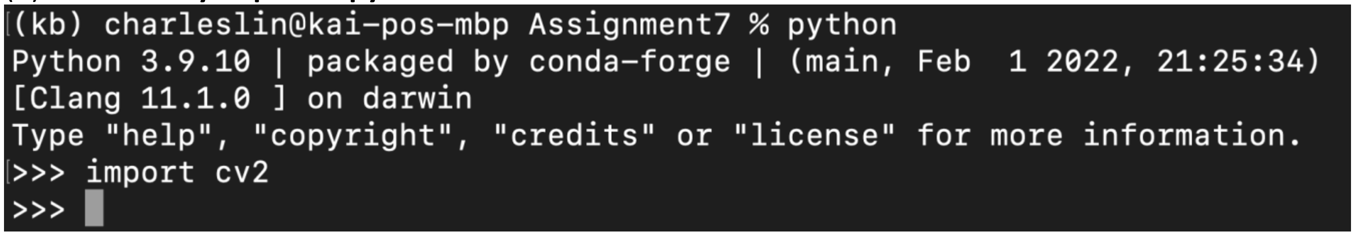
(3) **I finished installation on my virtual environment**

(a) **OpenCV version**



```
opencv-python                    4.5.5.64
```

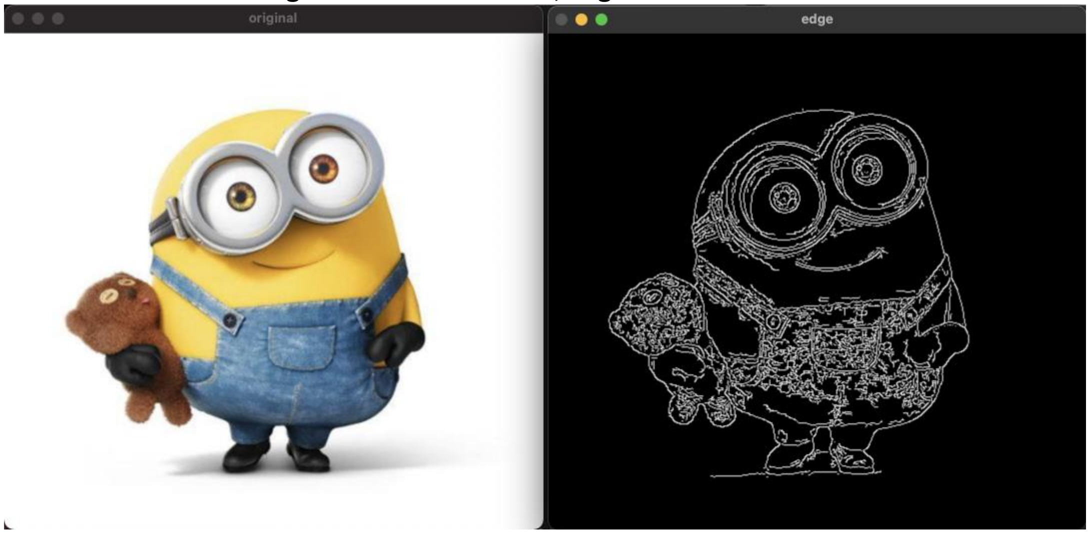
(b) **Successfully import in python IDE**



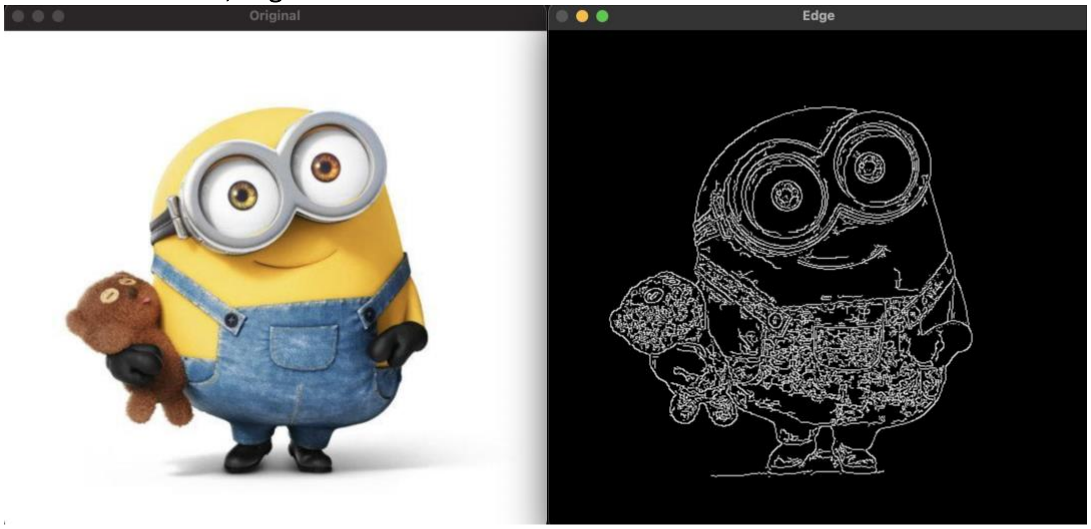
```
(kb) charleslin@kai-pos-mbp Assignment7 % python
Python 3.9.10 | packaged by conda-forge | (main, Feb  1 2022, 21:25:34)
[Clang 11.1.0 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> 
```

(4) **Perform canny edge detection on an image** (Image cited from: https://s.yimg.com/ny/api/res/1.2/I5hkvlJIiCoYQhlxDJMw_A--/YXBwaWQ9aGlnaGxhbmRlcjt3PTY0MDtoPTU4OQ--/https://s.yimg.com/uu/api/res/1.2/KIPdHTWS5T1Gdk9MDAkyjg--~B/aD00OTc7dz01NDA7YXBwaWQ9eXRhY2h5b24-/https://media.zenfs.com/en/tagsis_com_142/4f12347bad71ed062efe30a460b3adb4)

Base Threshold Setting: Low Threshold: 50, High Threshold: 150

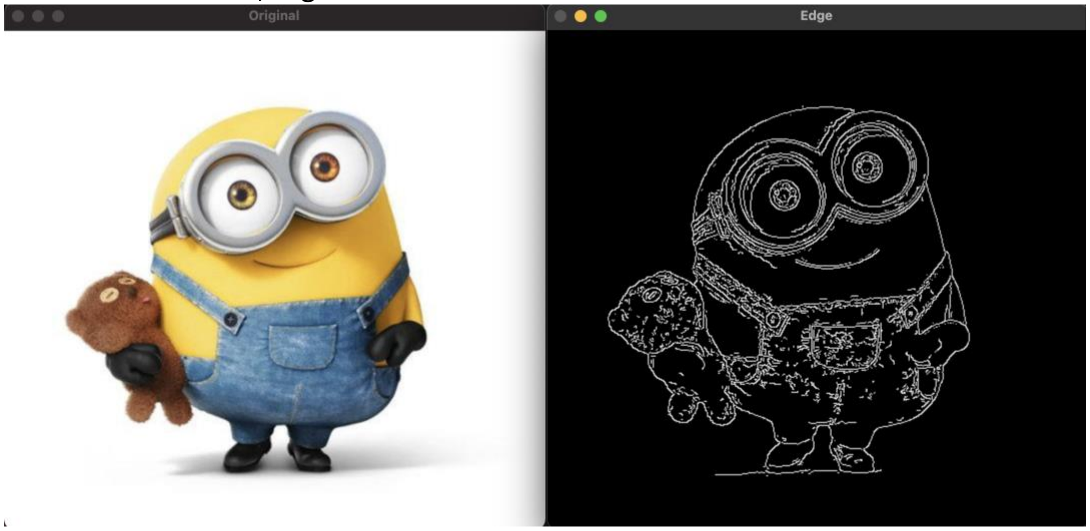


Low Threshold: 0, High Threshold: 150



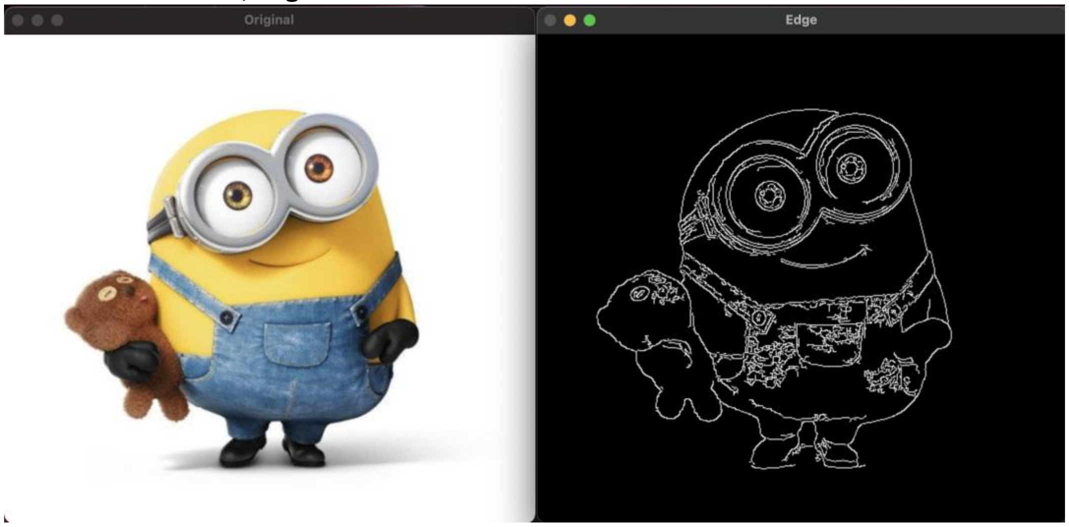
⇨ Lower the low threshold adds more edges to the edge image.

Low Threshold: 100, High Threshold: 150



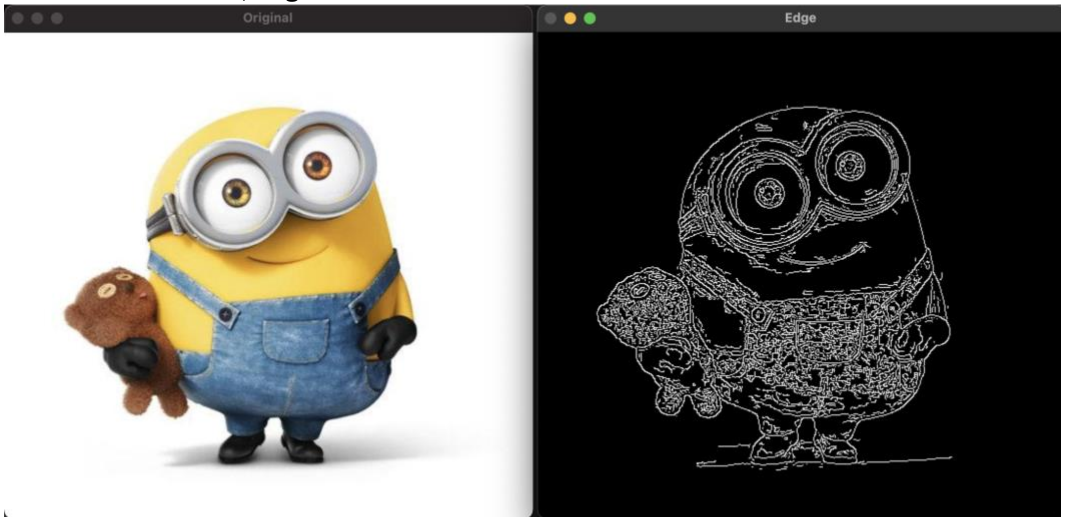
⇨ Raising the low threshold reduces the edges on the edge image.

Low Threshold: 50, High Threshold: 300



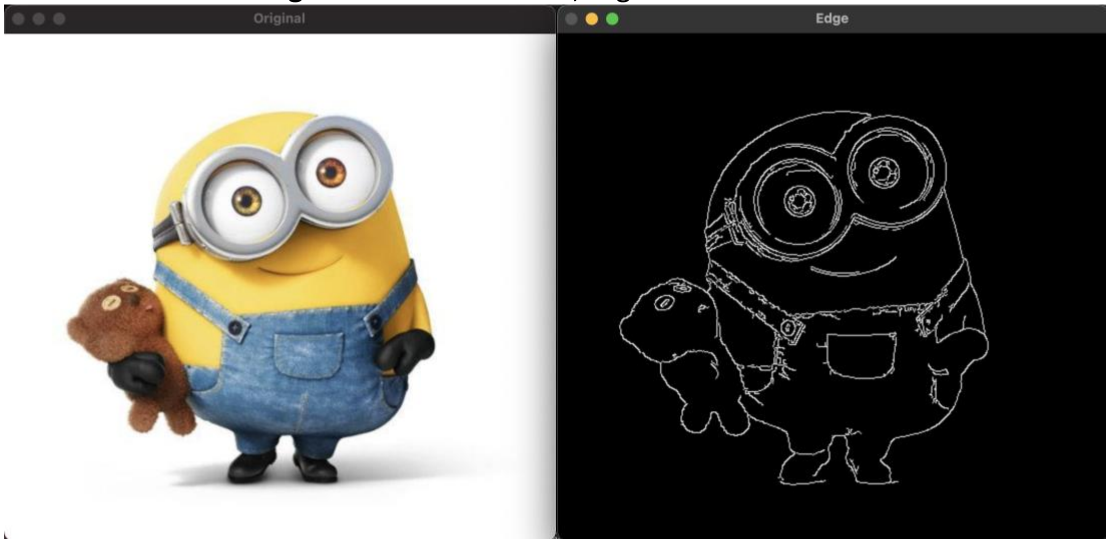
⇨ Raising the high threshold reduces the edges on the edge image.

Low Threshold: 50, High Threshold: 100



⇨ Lower the high threshold adds more edges on the edge image.

Best Threshold Setting: Low Threshold: 120, High Threshold: 400



Q: How to close the displaying window of OpenCV?
Ans: Press q on the images.

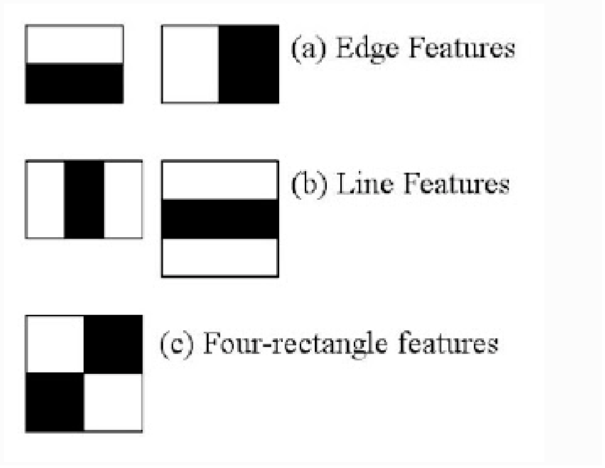
2. Face Detection

(1) **How does detection work?**
(Cited from: https://www.datacamp.com/community/tutorials/face-detection-python-opencv)

How Haar-Cascade work?

1. "Haar Features" Extraction

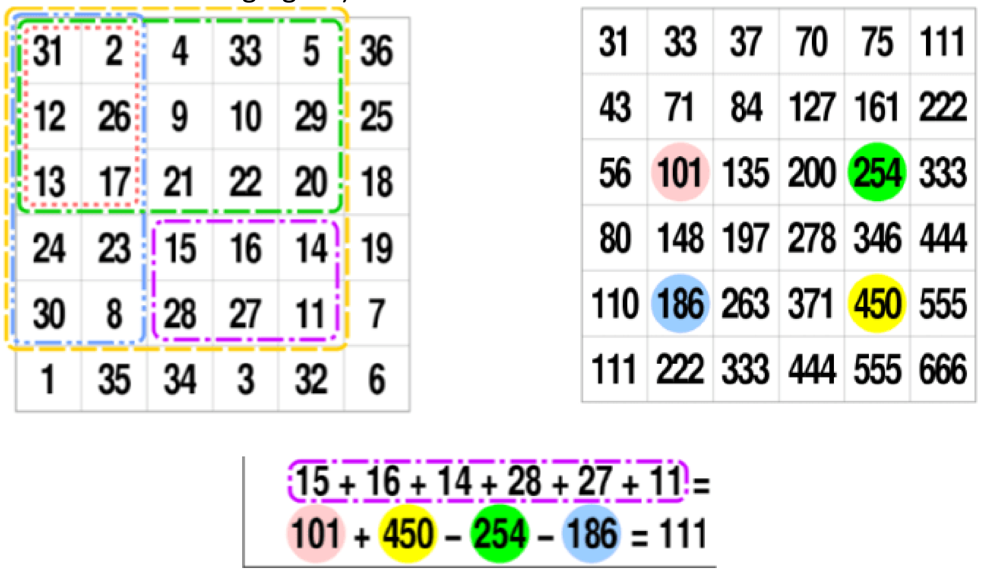
⇨ The classifier begins by extracting Haar features from each image. **Haar features are kinds of convolution kernels** that primarily **detect whether a suitable feature is present on an image or not**. The examples of the Haar features are mentioned below:



The feature is essentially a single value obtained by subtracting the sum of the pixels under the white region and that under the black.

2. Q: How to calculate the sum of the region? Ans: By using the concept of 'Integral Images'.

⇨ To calculate the sum of pixels in any feature window, we do not need to sum them up individually. All we need is to calculate the integral image using the 4 corner values (please refer to the following figure).

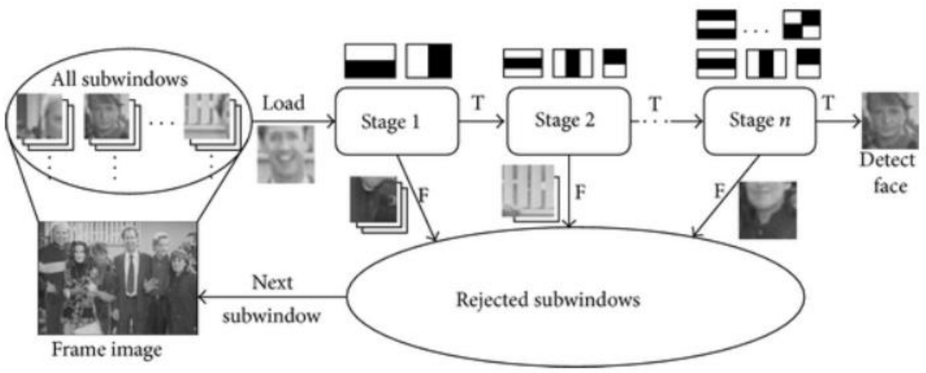


31	2	4	33	5	36
12	26	9	10	29	25
13	17	21	22	20	18
24	23	15	16	14	19
30	8	28	27	11	7
1	35	34	3	32	6

31	33	37	70	75	111
43	71	84	127	161	222
56	101	135	200	254	333
80	148	197	278	346	444
110	186	263	371	450	555
111	222	333	444	555	666

$$15 + 16 + 14 + 28 + 27 + 11 = 101 + 450 - 254 - 186 = 111$$

3. Eliminate calculating all feature maps
   (1) Adaboost
   ⇨ They use Adaboost to select the best features out of the entire chunk. By going through this process, the amount of features reduces from around 180,000 to around 6000.
   (2) Using 'Cascade of Classifiers'
   ⇨ The cascade classifier essentially consists of stages where **each stage consists of a strong classifier**. This is beneficial since it eliminates the need to apply all features at once on a window. Rather, **it groups the features into separate sub-windows and the classifier at each stage determines whether or not the sub-window is a face**. **In case it is not, the sub-window is discarded along with the features in that window**. If the sub-window moves past the classifier, it continues to the next stage where the second stage of features is applied.



The Parameters of cv2.CascadeClassifier.detectMultiScale():
1. scaleFactor => Parameter specifying how much the image size is reduced at each image scale
2. minNeighbors => Parameter specifying how many neighbors each candidate rectangle should have to retain (usually 3~5). The bigger, the detection is stricter.
3. minSize => Minimum possible object size. Objects smaller than that are ignored
4. maxSize => Maximum possible object size. Objects bigger than this are ignored

Ex:
```
faces = face_cascade.detectMultiScale(
    gray,
    scaleFactor=1.2,
    minNeighbors=5,
    minSize=(260, 260)
  )
```

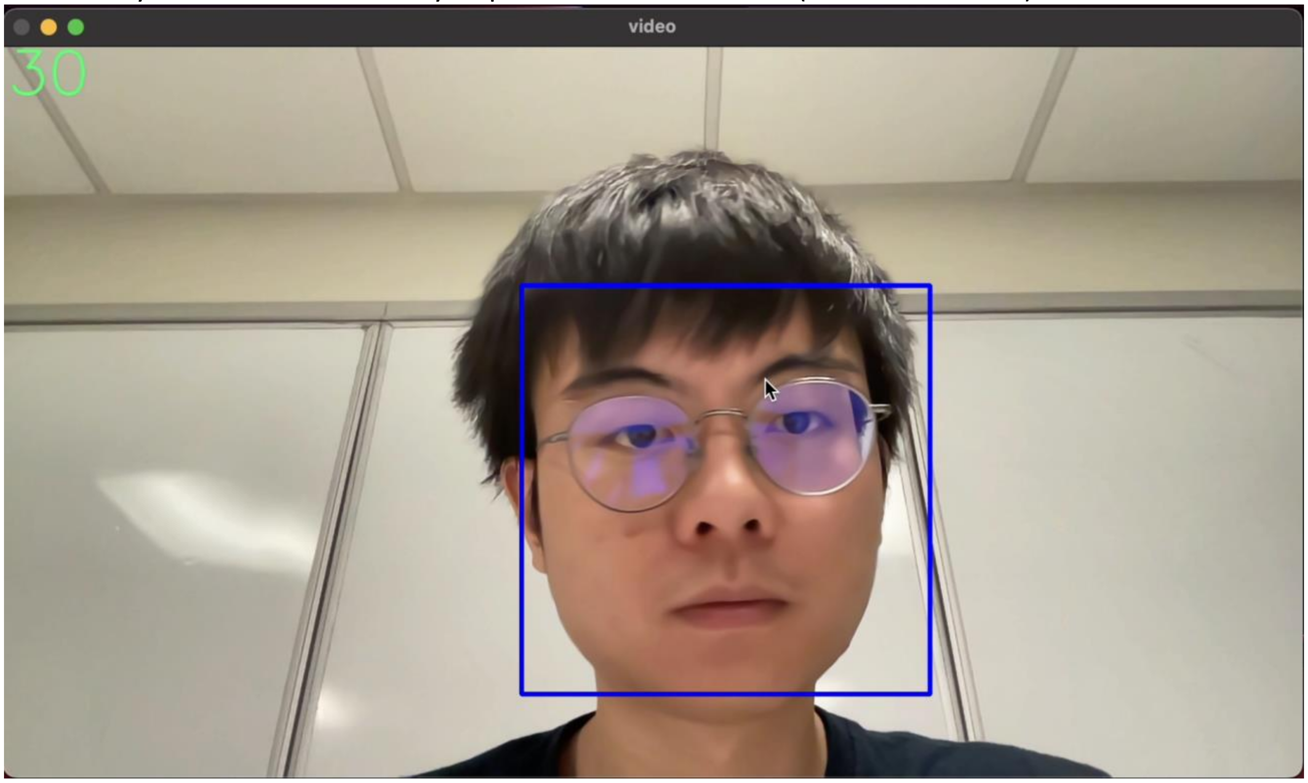
(2) What kind of challenges have you faced?
⇨ I think the overall implementation is pretty straightforward since the tutorial has already been provided and I've built a face verification system for the National Intercollegiate Athletes Games in Taiwan before. The only thing I tried to figure out in this section is the parameters in cv2.CascadeClassifier.detectMultiScale(). After that, I found the best values combination as the example above.

(3) Cases where it fails

⇨ Face detection shows it's vulnerable in the environment such as uneven light, some facial parts were covered, or the user's face wasn't direct to the camera. These circumstances lead the facial detection to failure since they undermine the features for the Haar-Cascade algorithm to make predictions.

(4) Describe any methods you used to accelerate the program

⇨ I think the face detection itself performs really great on my M1 Max Macbook Pro computer, so I did not use additional methods to accelerate the program. I considered that using the pre-trained model from haar-cascade and using the OpenCV package for real-time display was already the most efficient way to perform face detection (FPS about 28~31).



3. Real-Time CV Applications – Virtual Background (like the one in Zoom)

(Cited from: https://learnopencv.com/pytorch-for-beginners-semantic-segmentation-using-torchvision; https://learnopencv.com/applications-of-foreground-background-separation-with-semantic-segmentation/)

(1) Summary: In one page, summarize the key aspects of the implementation you used to solve the CV application of your choice. You will be penalized if you miss commenting on aspects that were crucial for the algorithm to function.

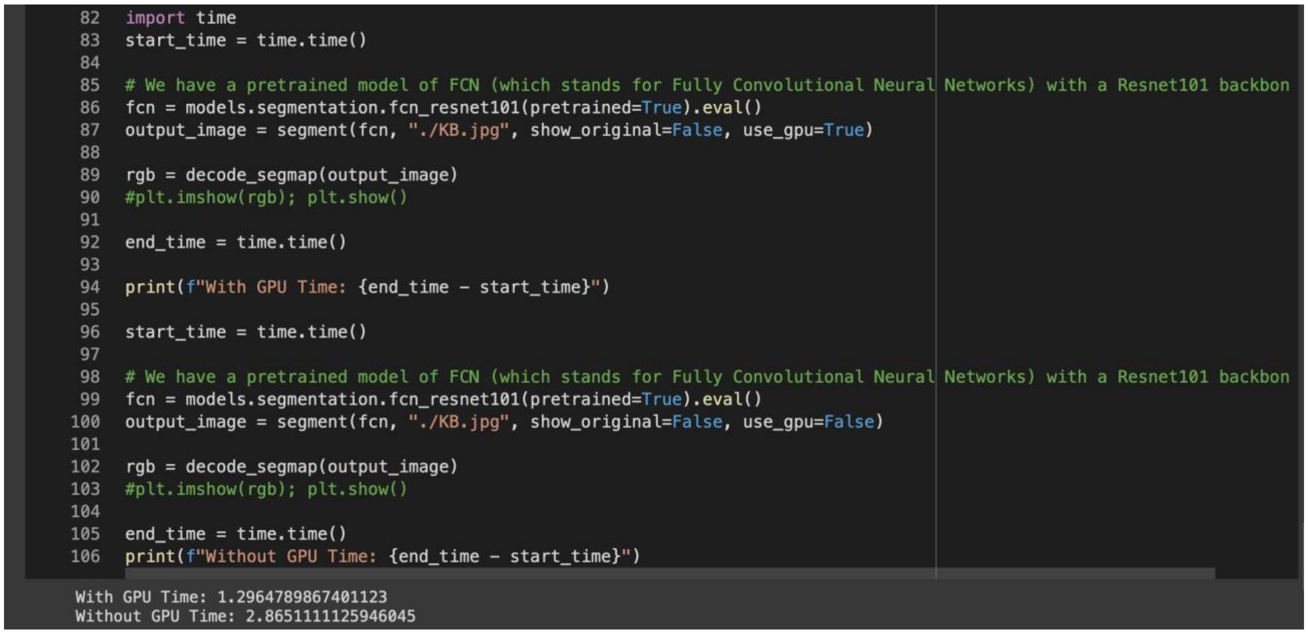
Implementation:

(1) Overall concept

In the virtual background section, I combined the concepts from the previous section. Instead of using some predefined images like the tutorials above, the input foreground image now will be the images captured by my webcam in real-time. Moreover, I added some functions to the background setting, which we could add more background images (support images end with '.jpeg', '.jpg', '.png', '.bmp') under the folder ‘Background’, and the users could switch between those backgrounds by clicking ‘a’ and ‘d’ from the keyboard. We could press ‘q’ if we want to quit this application.

(2) The implementation of the program could be divided into three parts:

- Object detection
  - ⇨ In object detection, we use FCN (Fully Convolutional Networks) as our object detector to analyze where the objects are within the image (real-time image from the webcam). The network requires some preprocessing to the input images, such as resizing and central cropping the image to 224*224, normalizing the image with the ImageNet specific values where mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225], and turned the input NumPy array to a pillow image. Moreover, in this section, I enable to use of GPU for the network prediction if the computer supports it. This would fasten two times the overall process as I tested it with Google Colab (please refer to the following image). With GPU, the prediction takes about 1.29 seconds, while without GPU, the prediction takes about 2.86 seconds.



```
import time
start_time = time.time()

# We have a pretrained model of FCN (which stands for Fully Convolutional Neural Networks) with a Resnet101 backbon
fcn = models.segmentation.fcn_resnet101(pretrained=True).eval()
output_image = segment(fcn, "./KB.jpg", show_original=False, use_gpu=True)

rgb = decode_segmap(output_image)
#plt.imshow(rgb); plt.show()

end_time = time.time()

print(f"With GPU Time: {end_time - start_time}")

start_time = time.time()

# We have a pretrained model of FCN (which stands for Fully Convolutional Neural Networks) with a Resnet101 backbon
fcn = models.segmentation.fcn_resnet101(pretrained=True).eval()
output_image = segment(fcn, "./KB.jpg", show_original=False, use_gpu=False)

rgb = decode_segmap(output_image)
#plt.imshow(rgb); plt.show()

end_time = time.time()
print(f"Without GPU Time: {end_time - start_time}")
```

```
With GPU Time: 1.2964789867401123
Without GPU Time: 2.8651111125946045
```

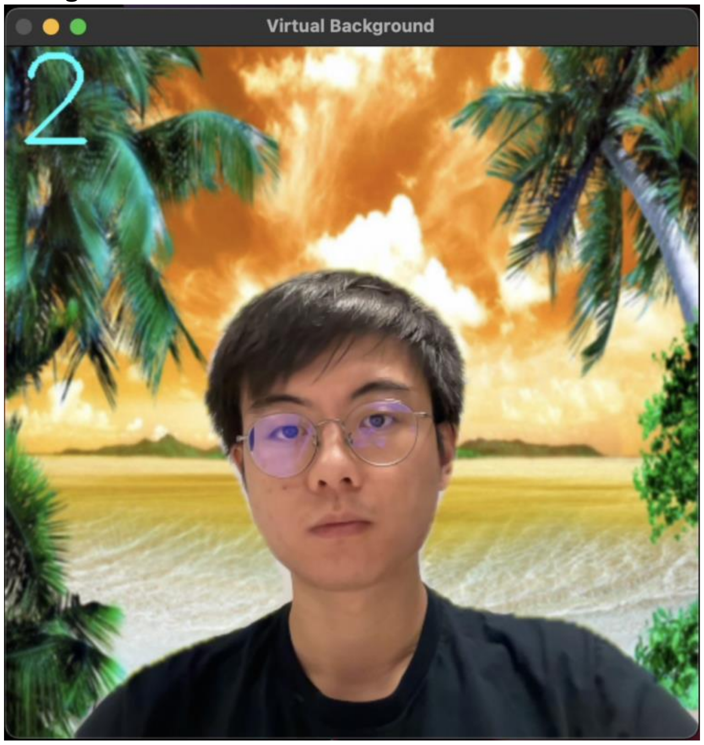
- Colored the object we want in the picture (object human in this assignment)
  - ⇨ Humans in the fcn network will be labeled as 15. That is, in this function, I am coloring the labeled image (after object detection) into specific RGB color to differentiate the foreground (human) and the background (non-human part).

- Image matting
  - ⇨ This is the main function where I form the foreground image and the background image together. After reading the foreground image (real-time image from the webcam) and the background image (given background image within the ‘Background’ folder), I resized them to be the same shape as the output image of the object detection function, which is the output image after the prediction of

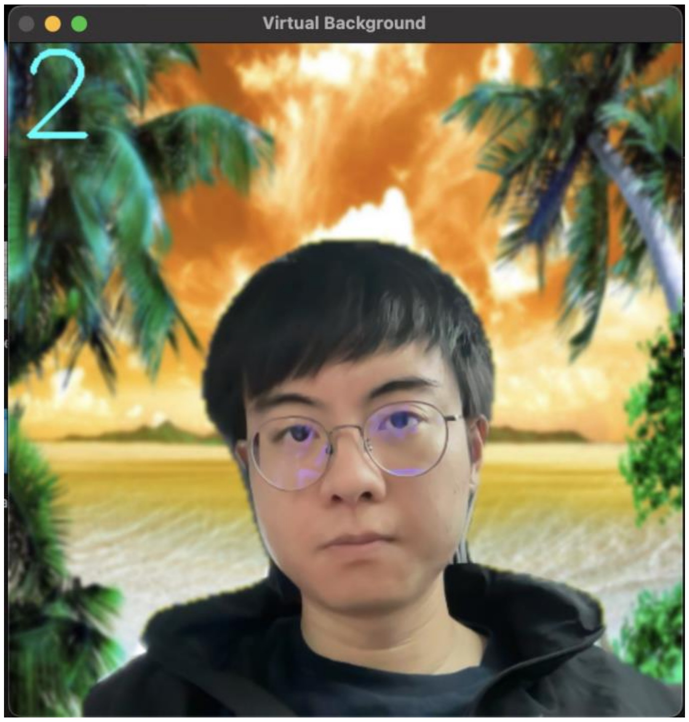
the fcn network (I’ve resized the output of the fcn network to 512*512 for better displaying to the output image onto the computer screen).
Then, I create a gaussian filter for the background image such that the subject is in focus, and the background is out of focus so that the final image will be more harmonic. I also applied a gaussian filter onto the threshold mask to make the edges between the foreground image and the background image smoother. The threshold mask here is a mask that I defined the value below 128 to be non-human (set to 0), and the value above that to be human (set to 1).
Eventually, I multiply the threshold mask with the foreground to show the human in the original image and to hide the background in the original image. Moreover, multiply 1 – threshold mask to show the background with human of the given background image and hide the human area in the background image. The output will be the summation of the two images after processing with the threshold mask.

(2) Parameters: What are all the important parameters that are up to the user of your algorithm to decide? For example, the threshold value in binarization. Report images or video recordings from your live feed that demonstrate the effect of tweaking these parameters. Justify these effects.

⇨ 1. The filter size of the background gaussian blur. If the user inputs a higher value, the background will be blurrier.

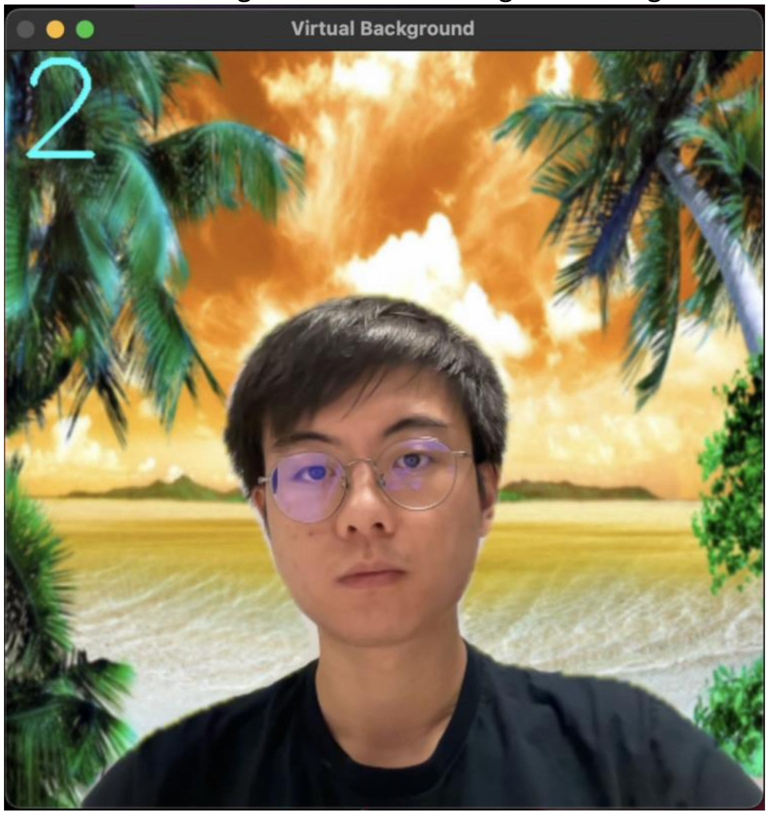


Background image with 3*3 window for the Gaussian Blur

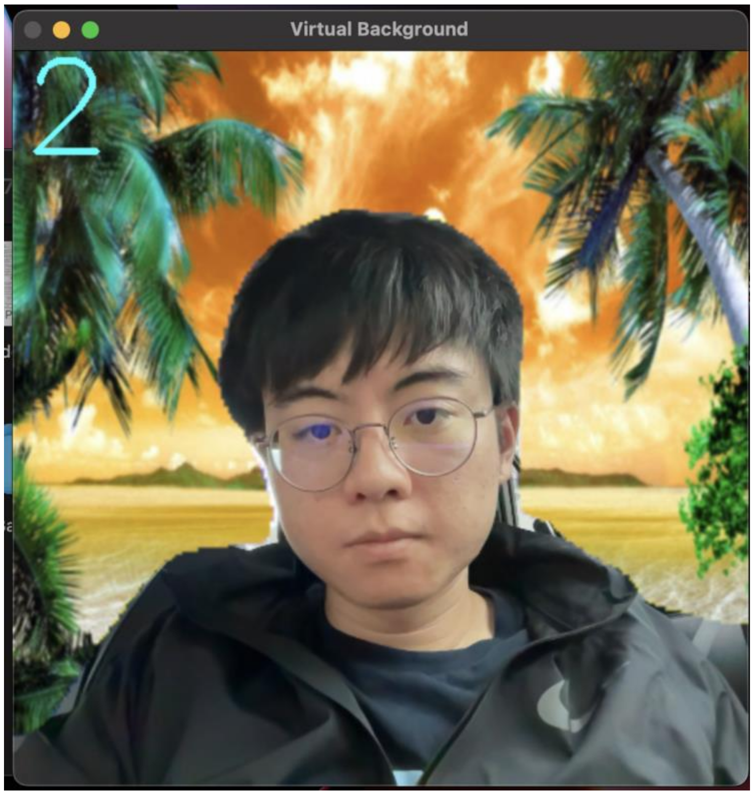


Background image with 7*7 window for the Gaussian Blur

2. The filter size of the threshold mask gaussian blur. If the user inputs a higher value, the edges between the background and the foreground image will be smoother.

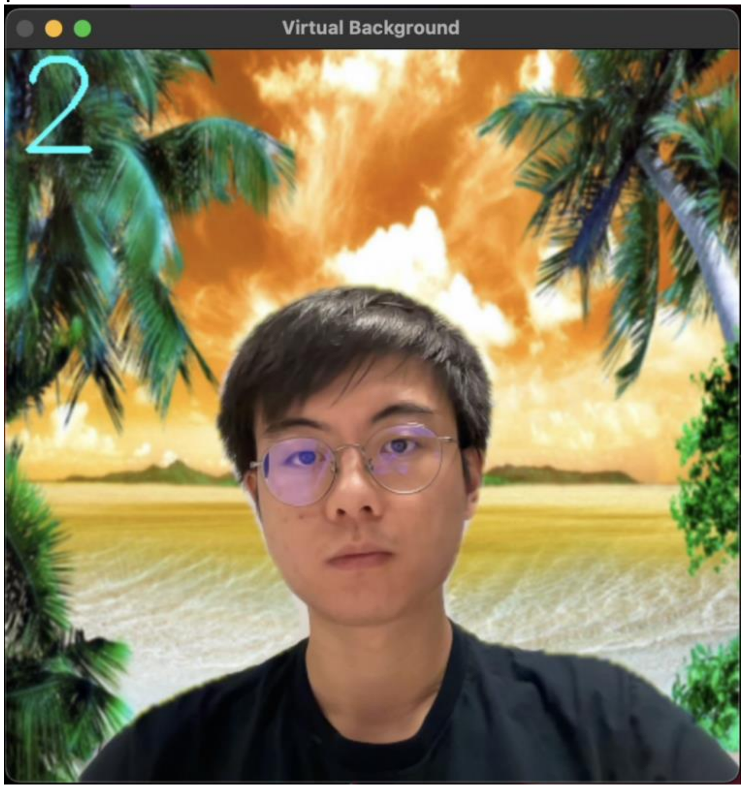


Threshold mask with 7*7 window for the Gaussian Blur (as we can see, the edges between the two images are smoother)

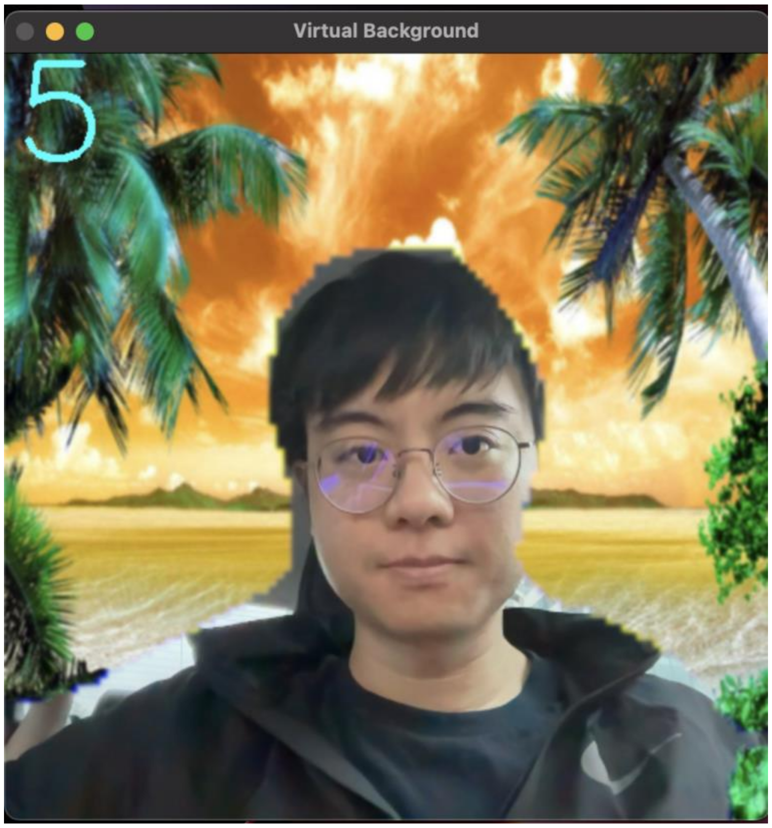


Threshold mask with 3*3 window for the Gaussian Blur (as we can see, the edges between the two images are steeper)

3. The value of resizing and center cropping (the two values should be the same). If we set the value higher, the FPS will be lower, which means the video will be lagger; however, the prediction will be better.



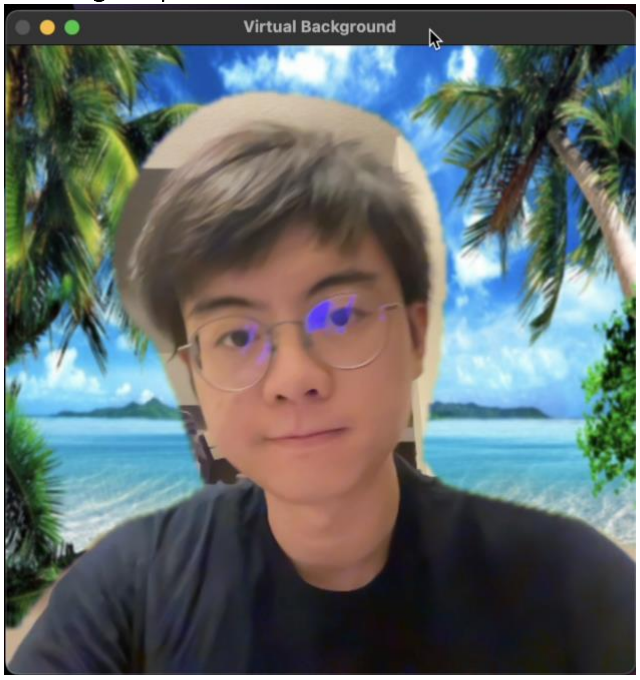
Resizing and central crop to 224*224



Resizing and central crop to 100*100

(3) Failure cases: Report images or video recordings from your live feed where the algorithm fails to perform the required application. Justify why that's the case.

⇨ I have a video called "issue_fcn_virtual_background.mov", which demonstrates **if we follow the tutorial by setting 256 sizes in transforms.Compose.Resize() and 224 in transforms.Compose.CenterCrop()**, we will get the result like the video. The following picture is an image captured from the video.



In conclusion, we have to set the resize and center crop parameters to the same value. I considered the reason is that if we resize and center crop the image to different values, **we might miss some of the information within the image**. For example, if the person occupied almost all of the image, then if we resized the image to 256*256 and center cropped the image to 224*224, we might lose some features like the hair or even some part of the face, which was cropped by the central crop process.

4. Conclusion

(1) Issue

The biggest issue I encountered was the one I stated in the previous section: **the resize and center crop parameters should be set as the same value**. The tutorial mentioned that we have to resize the image to 256*256 and central crop the image with 224*224; however, it is incorrect. My prediction of that setting is really bad since there will always be some error detection near the face. I eventually find out this error and perform decent results on background switching.

(2) Surprise

I think this assignment is really interesting and I learned plenty of skills from it although I've already designed facial verification systems before. For example, in question 3, I haven't thought to combine the object detection method with computer vision could perform virtual background switching like zoom before. I considered that this assignment is really useful, and I totally enjoy this assignment.