**CS506 Programming for Computing**

**HOS05A– Machine Learning**

2/15/2025 Created by Jin Chang

4/10/2025 Reviewed by Clark Ngo

School of Technology & Computing (STC)

City University of Seattle (CityU)

**Before You Start**

* **Screenshots may be different from your environment.**
* The directory path shown in screenshots may be different from yours.
* There might be subtle discrepancies along with the steps. Please use your best judgment while going through this cookbook-style tutorial to complete each step.
* Some steps may not be explained in detail. If you are not sure what to do:

1. Consult the resources from the course.
2. If you cannot solve the problem after a few tries (usually 15 -30 minutes), ask a TA for help.

**Learning Outcomes**

Students will be able to:

* Explain how the Haar Cascade Classifier (HCC) works
* Implement HCC step-by-step in Python for face detection, eye detection, and eyeglasses detection
* Utilize the implementation on the actual face images

**Section 1: Introduction to Haar Cascade Classifier**

**Section 2: Implementation of Haar Cascade Classifier using OpenCV**

**Section 3: Haar Cascade Classifier Parameter Exploration**

**Resources**

* <https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html>
* <https://github.com/opencv/opencv/tree/4.x/data/haarcascades>

The original publication is from COMPUTER VISION AND PATTERN RECOGNITION 2001.



**Section 1: Introduction to Haar Cascade Classifier**

A Haar Cascade Classifier is a machine learning algorithm for image object detection. It is particularly famous for face detection, where it quickly scans an image for potential object locations by analyzing "Haar-like features" in a cascading manner. This means it uses a series of progressively more complex checks to efficiently identify the object of interest while discarding irrelevant areas of the image very early on.

**Haar-like features:** These are simple rectangular regions in an image where the pixel intensities are compared between different areas (like the area around the eyes vs. the area below the nose) to identify potential object characteristics.

**Cascade structure:** The classifier is organized in stages, each with weak classifiers based on Haar-like features. If a window in the image fails to pass a particular stage, it is immediately discarded as not containing the target object, saving computation time.

**Training process:**

* **Positive samples:** Images containing the object you want to detect (e.g., faces).
* **Negative samples:** Images without the object.
* **Feature selection:** The algorithm analyzes Haar-like features and selects the most discriminative ones for each stage, prioritizing features that best separate positive and negative examples.
* **Boosting:** The selected features are combined using a boosting algorithm (like AdaBoost) to create strong classifiers that can accurately detect the target object.

**Example explanation:**

"Imagine you're trying to find a face in a crowd. A Haar Cascade classifier (HCC) would first quickly scan the image for areas with strong contrast between dark and light regions (like the eye sockets), likely to be faces. If a region looks promising, it will then apply more detailed checks, like checking for the shape of the nose and mouth, only if the initial checks pass. This way, it can quickly eliminate most images not containing a face, focusing only on potential face areas."

**Strengths of Haar Cascade:**

* **Fast real-time detection:**

Due to the cascading structure, it can quickly analyze images and identify objects efficiently.

* **Simple to implement:**

Compared to more complex object detection methods, Haar Cascade is relatively easy to understand and use in practice.

**Weaknesses of Haar Cascade:**

* **Limited accuracy:**

HCC can be susceptible to false positives, especially in challenging lighting conditions or when objects are partially occluded.

* **Not very robust to pose variations:**

Detecting objects significantly rotated or positioned differently from the training data may be difficult. Also, if the orientation is severely off, HCC will struggle to detect the target objects. As a result, accuracy will be compromised based on the rotation of the objects.

**Section 2: Implementation of Haar Cascade Classifier using OpenCV.**

The OpenCV module provides tutorials on implementing it and various trained models. We will utilize this resource in this HOS to understand how ML generally works.

* 1. Prepare your Jupyter Notebook environment in codespaces: <https://cityuseattle.github.io/docs/environment/codespaces_jupyter/>
  2. Create a file named classifier.ipynb
  3. Install the following dependencies to import cv2 in GitHub Codespaces:

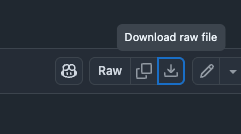
|  |
| --- |
| **Mac:** sudo apt-get update  sudo apt-get install -y libgl1  pip install opencv-python matplotlib |
| **Windows:** pip install opencv-python matplotlib |

* 1. Download the following XML files of already trained models from the GitHub below.

<https://github.com/opencv/opencv/tree/4.x/data/haarcascades>

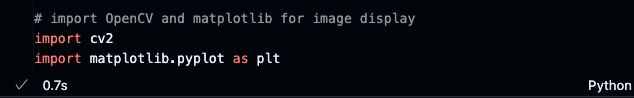
[haarcascade\_frontalface\_default.xml](https://github.com/opencv/opencv/blob/4.x/data/haarcascades/haarcascade_frontalface_default.xml) (Note this is a trained model for face detection)

[haarcascade\_eye.xml](https://github.com/opencv/opencv/blob/4.x/data/haarcascades/haarcascade_eye.xml) (Note this is a trained model for eye detection)

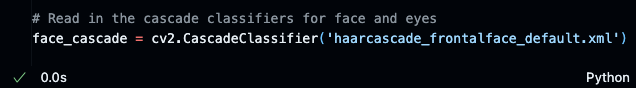


* 1. After downloading the models (XML files), save them.

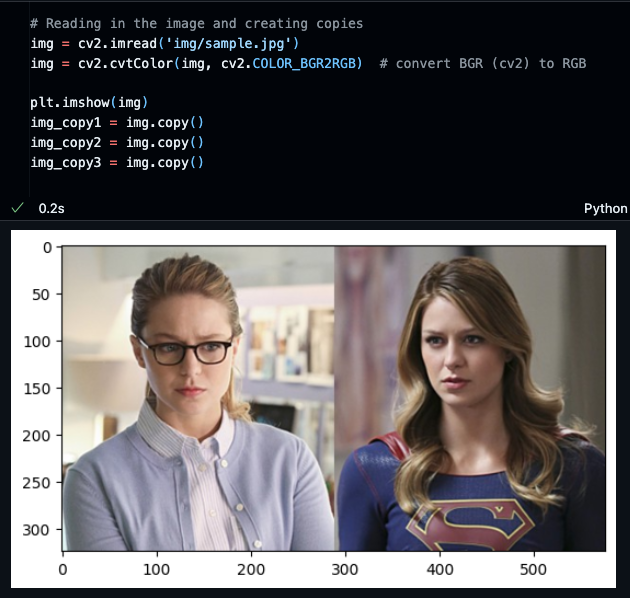
**Step 1**. Import the modules



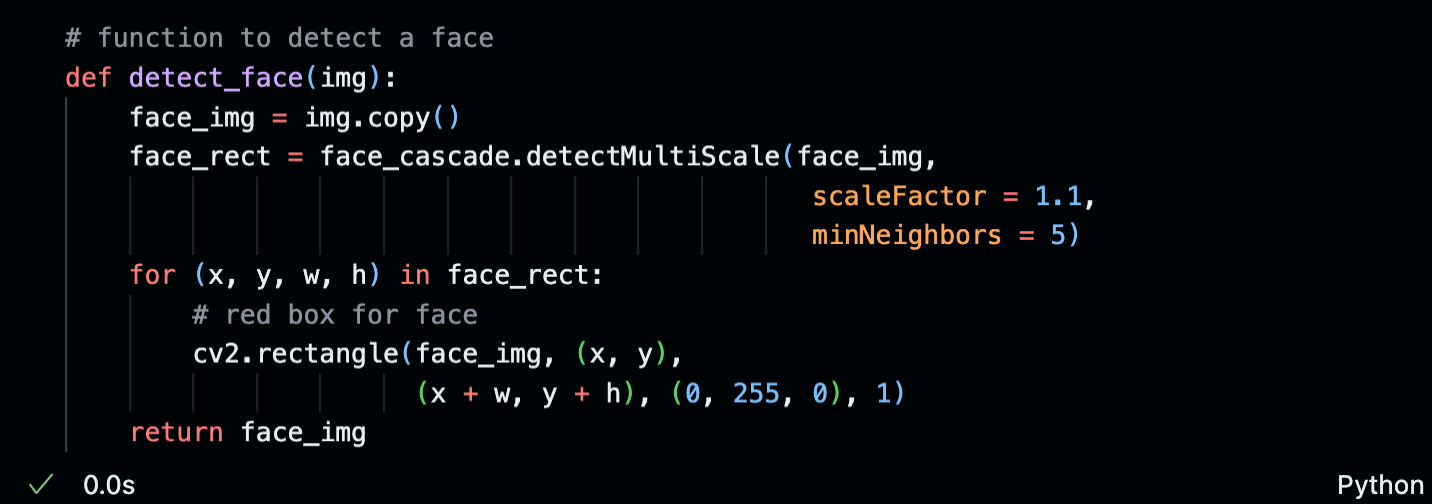
**Step 2**. Load the face detection train model. Your path to the downloaded models will differ from the following code snippet. Make sure you remember where you saved those XML files.



**Step 3**. Use the cv2 module to read the sample.jpg for testing. Since cv2 uses reversed RGB, convert BGR to RGB to correct the color space for the plt module to display.



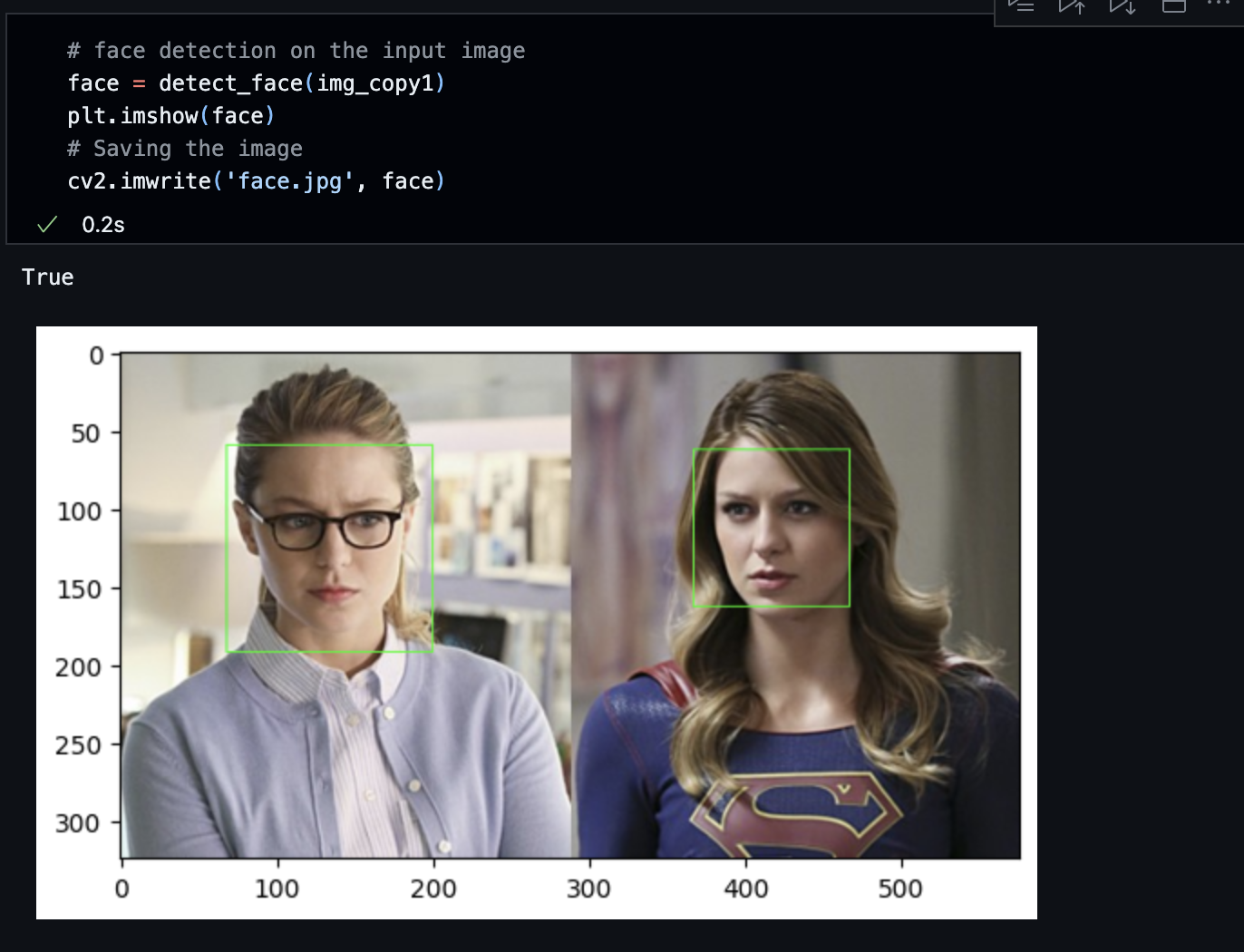
**Step 4**. Define a function called detect\_face(img) to find the faces in the input image, img.



Two critical parameters to pay attention to are scaleFactor and minNeighbors. These will impact the overall accuracy of the face detection rate. Refer to the following link for more information about detectMultiScale(). We’ll examine these parameters in later sections.

<https://docs.opencv.org/3.4/d1/de5/classcv_1_1CascadeClassifier.html>

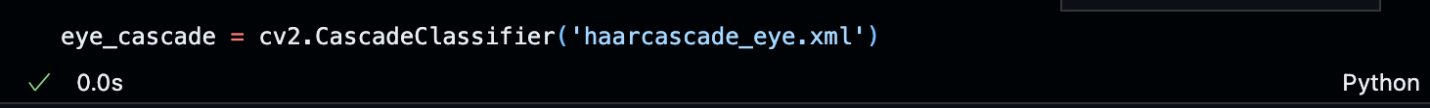
**Step 5**. Invoke the detect\_face() and display the input image with the bounding boxes locating the faces in the image.



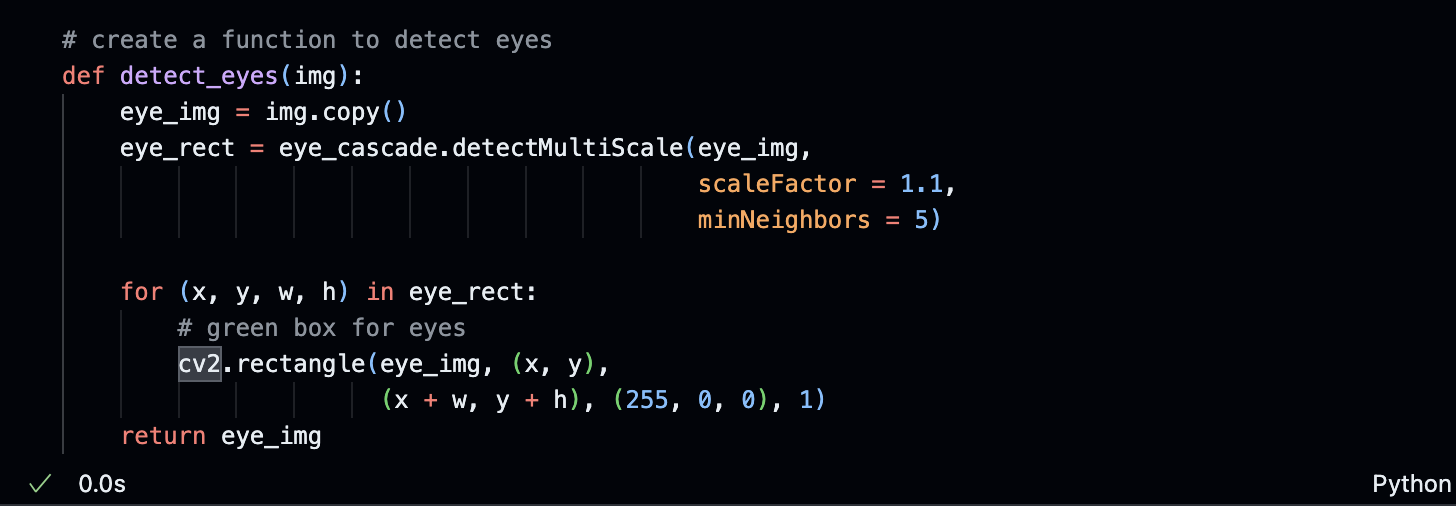
Feel free to test on various face images by varying the scaleFactor and minNeighbors and observing how it impacts the positive rate (meaning detecting the presence of a face).

Now, let’s try eye detection.

**Step 6**. Load the eye detection train model. Your path to the downloaded models will differ from the following code snippet. Make sure you remember where you saved those XML files.



**Step 7**. Define a detect\_eyes() as shown below.

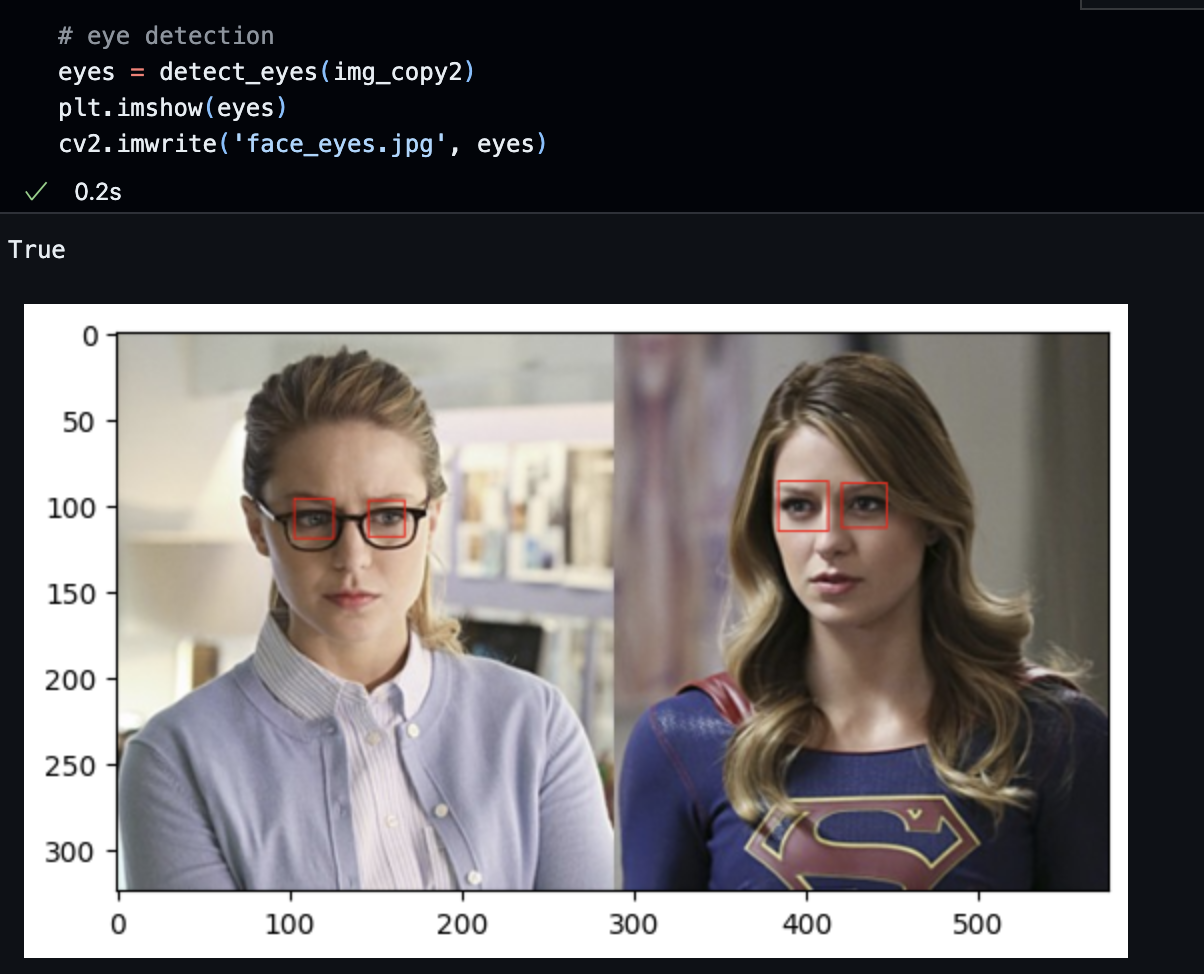


Similarly, the values for scaleFactor and minNeighbors for eye detection vary, impacting the overall detection rate. Since we’ll use a similar resolution for this experiment, you don’t need to adjust minSize or maxSize.

**Step 7**. Invoke the detect\_eye() and display the image with the red bounding boxes locating the eyes in the image.

A black background with white text

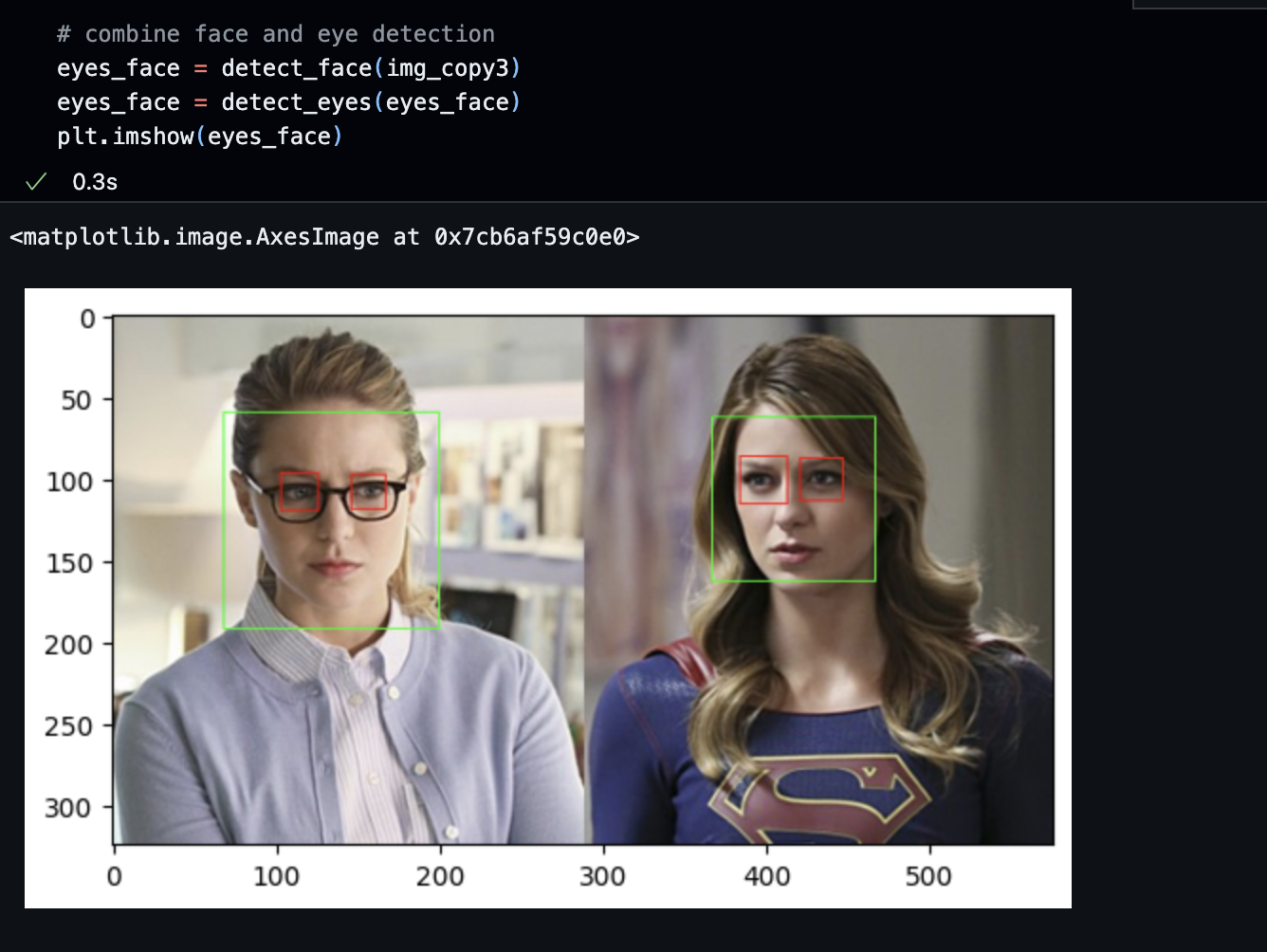
AI-generated content may be incorrect.



**Step 8**. Let’s combine face and eye detection and display what the algorithm found in the image.

**A screen shot of a computer code

AI-generated content may be incorrect.**



**Section 3: Haar Cascade Classifier Parameter Exploration**

Let’s try another image, train5.jpg (img/train5.jpg). If you run the code by replacing the filename in the cv2.imread(), this is what the algorithm produces. Face detection works fine with the same parameters we used in the sample.jpg.

A person with a beard

Description automatically generated

However, in eye detection, it performed poorly. It found 5 areas where they think they are the eyes or the candidates of the eyes, resulting in 3 false positive findings. You want to have those two large boxes enclosed by the blue arrows. So, we need to adjust or optimize those scaleFactors and minNeighbors, minSize, or maxSize.

The scaleFactor specifies how much the image size can be scaled down to improve performance over time, with some compromise on detection quality. For example, if you set the scaleFactor at 1.1, you reduce the image size by 10%, hoping it is sufficient to detect faces. However, using a larger value like 1.8 significantly reduces the image size, hence lower accuracy.

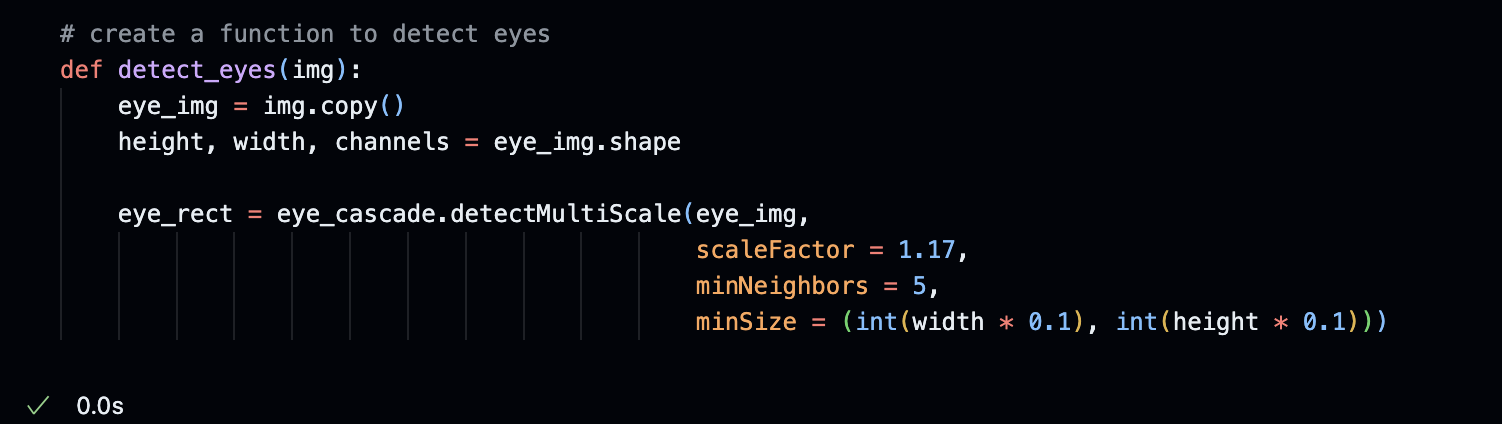
The second parameter we need to consider is minNeighbors. If you’re using lower values, you’re thinking of more candidates for eye detection. Higher values mean that you feel fewer candidates for detection. Other parameters that you will find helpful are minSize and maxSize. minSize specifies the minimum size of the detected region to consider. In other words, any regions smaller than minSize will be discarded. maxSize specifies the maximum size of detected areas to consider. Any regions that are larger than maxSize will be ignored.

A person with beard and mustache

Description automatically generatedA person with beard and mustache

Description automatically generated

|  |  |
| --- | --- |
| A person with beard and mustache  Description automatically generated | **scaleFactor = 1.1**  **minNeighbors = 5** |
|  | **scaleFactor = 1.17**  **minNeighbors = 5** |
|  | **scaleFactor = 1.17**  **minNeighbors = 5**  **minSize = (int(width\*0.1), int(height\*0.1)) ignoring the smaller box.** |



Finally, it is always a good idea to normalize the size of the image to the same size to avoid any variability caused by the differences in image size and resolution. We’ll update the read () routine as follows.



We'll need to adjust the following parameters once we normalize the image size to 240 x 240. You may end up with different parameter settings.



And these are the outputs of a few more samples.

A person with a beard and a beard

Description automatically generated(train5.jpg)

A person with a red rectangle and green rectangle

Description automatically generated(train1.jpg)

A person with a green rectangle and red rectangle

Description automatically generated(train2.jpg)

**Challenge 1: Since this is a face detection algorithm, test the algorithm on the images without faces and see if it works as expected.**

**Challenge 2: Since this is an eye detection algorithm, test the algorithm on the images without eyes or faces and see if it works as expected.**

**Push your work to GitHub.**

Follow instructions here: <https://cityuseattle.github.io/docs/git/codespaces_submission/>