**OpenCV: Face Detection**

**Introduction:**

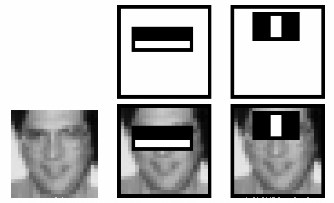
Artificial intelligence is the ability for machines to perform tasks that require a human level of intelligence. Tasks such as recognizing people in a photo or understanding human speech. AI can be reflected in computer programs or machines/robots. Computers are great at processing information; AI seeks to use that information to make smart decisions. Ideally, AI should be able to rationalize and take actions that give it the best chance of achieving a specific goal. I wanted to work on a program that can detect faces in a photo.

A common method of achieving this task is building a model. A model that is used by a convolutional neural network that classifies objects in an image. A CNN is made up of layers of nodes. The first set of nodes make up the input layer which reads in the data. This is a feature map that represents an image. The image is passed to hidden layers of nodes; these layers update the weights accordingly to extract features from the image with the help of the learning rate and other tuning parameters [2]. Additionally, the feature map goes through convolution and pooling operations to absorb the features and compute new ones by sliding a filter over the feature map [3].

Finally, the output layer of nodes known as classifier predicts the type of object in the image. This could be a specific person or animal, etc. An example of a CNN in action can be a network that reads in an image of cats and dogs and attempts to classify which is which. The weights play a vital role in representing the network's knowledge about the image, so it can properly classify things. Based on the type of problem, an appropriate classifier and loss function can be chosen. To make the final prediction and evaluate the prediction, respectively. This is the general approach.

**Method:**

A more specific approach is to use Haar featured based cascade classifiers. It is a machine learning method that trains a cascade function on several positive and negative images. Positive images have the faces and the negative images do not have faces. So, the classifier can determine what a face looks like and what it does not look like. The algorithm starts off with training the classifier with a dataset of positive and negative images. Features are extracted from the images. Each feature is depicted as a single value that can be acquired by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle [4].

 **Figure 1:** The white and black rectangles are used to calculate features [4].

When computing features, it can help to apply filter matrices of varying sizes to the image. This however would take a lot of computation considering all the filter matrix sizes and their possible locations across the image. For each feature calculation, the sum of the pixels under white and black rectangles is required. To alleviate this computation, integral images where introduced. No matter how large your image is, it decreases the amount of computation for a specific pixel to an operation requiring only four pixels [4]. This makes things a lot quicker.

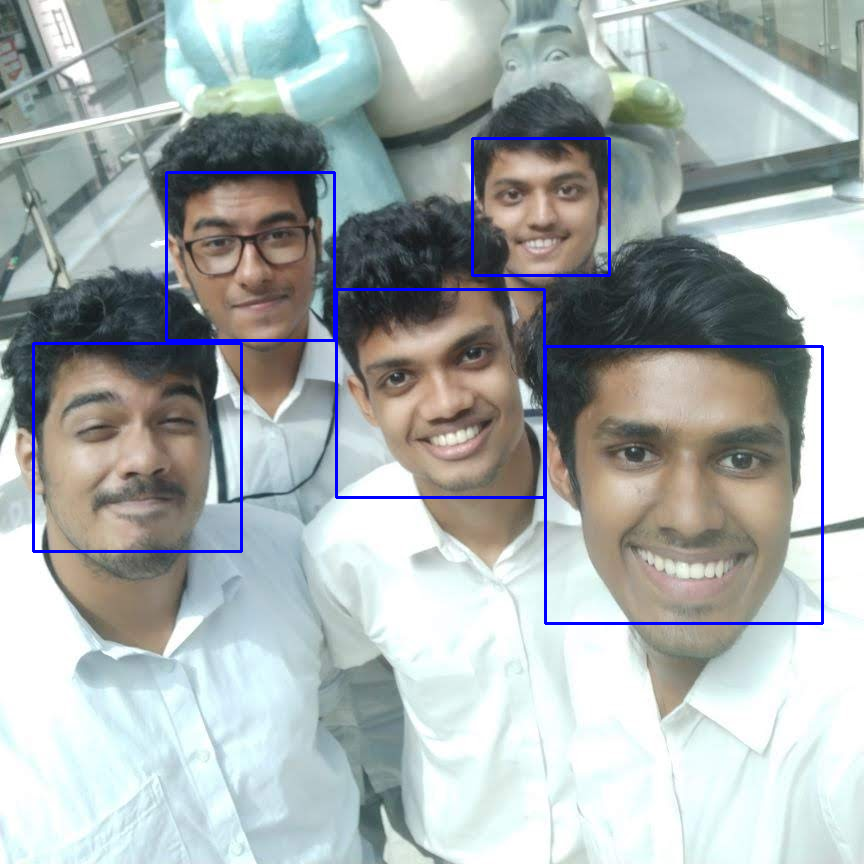
Not all features are relevant in identifying a face. Some features can correspond to the background of the image and others may be trivial in detecting a face. We must select the best features out of all the features obtained. This can be attained with AdaBoost. Each feature is applied to every training image. For each feature a threshold is found that classifies whether this feature is most likely to belong to face or nonface. There are bound to be some misclassifications, that is why the features with the least amount of error are chosen. These features can be used to correctly classify faces and nonfaces in images. At the start of the process, each feature is given the same weight [4]. After each classification attempt, the weights of incorrectly classified images are increased. This is general paradigm of AdaBoost. We seek to dismiss the images we have performed well on known as the easy examples and prioritize images that we have performed poorly on known as the hard examples [1]. Updated error rates and weights are calculated after each iteration; the process goes until we are content with accuracy of the algorithm.

The final classifier is a weighted sum of the weak classifiers. They are weak since they do not possess the capability to classify an image on their own, but together they can. So far, the method works, but it’s quite extensive in terms of time and features used. One key observation about images is that most of the area in an image represents a nonface region. Considering this, it would make sense to first check if the window is a face region or not. If it’s a face region we want to process this part once and not look at it again, so we can move on to other regions that might have a face [4]. Doing things this way ensures we spend more time looking at relevant regions where faces might be.

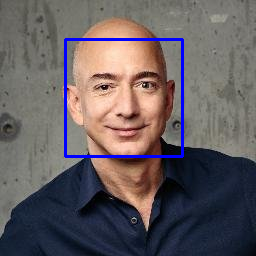
This is where the notion of cascade of classifiers come into play. Features are assembled into groups that represent different stages of classifiers. In the case in which a window fails, we throw it out. Not examining the remaining features associated with it. If it passes, we move on the next stage of features. If the window manages to successfully pass all the stages, it is confirmed as a face region [4]. The stages are meant to weed out the nonfaces, so we are only left with faces.

**Discussion:**

Fortunately, OpenCV has a pretrained model for the cascade classifier that we can use. The cascade classifier is created, and the model is loaded into the program to detect faces in an image with the configured XML file. Finally, the detection is done using the detectMultiScale() which creates a bounding box for the detected face.



**Figure 2:** Bounding boxes detect five faces in a single image [5].



**Figure 3:** Bounding box detects Jeff Bezos’s face [6].

One obstacle I encountered was finding the appropriate XML file to use with the program. Sometimes the file could not be read because it was in the wrong directory or the program was missing the correct file path. This was implemented in Google Colab; the build environment had a conflict with OpenCV. OpenCV has a cv.imshow() that displays an image, but that function was incompatible with Google Colab. Fortunately, Google Colab has their own version of the function that provides the same behavior called cv2\_imshow().

**Conclusion:**

In addition to detecting faces in photos, I want to detect faces in video as well. I was not able to do so since Google Colab is a virtual machine and does not have access to my computer camera. Although, there is a way to use JavaScript to access the computer camera, I am currently not capable of merging JavaScript’s functionality of recording video and capturing pictures with Python’s functionality of detecting the faces in the picture. That is the plan for a future project. Also, I would like to classify the faces as specific people as well. Given the time to feed the program a labeled dataset containing images and names for the faces in the image, the program can move its classification up to the next level.

Regarding the strengths of the program, the program was able to read in an image and detect multiple faces using a pretrained model. One more future goal for the program is to detect animal faces as well. The current classifier being used is only trained for human faces, so it’s only good for that. Perhaps with another classifier or an entirely newly made model from scratch animal faces can be detected too.



**Figure 4:** The bounding boxes fail to detect the German Shepperd’s face [7].

**References:**

1. Niels Lobo. University of Central Florida. <http://www.cs.ucf.edu/courses/cap4453/4453NeuralNetworks4-1.pdf>. Accessed 17 Apr. 2020.
2. Pawel Wocjan. <https://github.com/schneider128k/machine_learning_course/blob/master/slides/anatomy_of_neural_network.md>. Accessed 17 Apr. 2020.
3. Pawel Wocjan. <https://github.com/schneider128k/machine_learning_course/blob/master/slides/CNN_slides.pdf>. Accessed 17 Apr. 2020.
4. OpenCV. <https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html>. Accessed 17 Apr. 2020.
5. <https://www.youtube.com/watch?v=7IFhsbfby9s> Accessed 17 Apr. 2020.
6. <https://www.youtube.com/watch?v=D5xqcGk6LEc> Accessed 17 Apr. 2020.
7. WikiMedia Commons. <https://commons.wikimedia.org/wiki/File:German_Shepherd_Dog_with_disc.jpg>. Accessed 17 Apr. 2020.