COMP 4102 - Final Project

Human Counter

Team members:

Andrew Burry - 100832328

Cody Bennett - 101035873

Jake Jazokas - 101083496

Abstract

Human detection is a well documented computer vision problem with applications becoming more and more prevalent in our everyday lives. The goal of our project was to accurately count the number of people in an image or video. To achieve this goal, we explored several detection utilities readily available within the OpenCV library, namely cascade classifiers, background subtractors and deep neural networks. After comparing the results of each of the human detection methodologies, the deep neural network based detector was the most successful at detecting humans in both images and videos with a high degree of accuracy.

Introduction

We implemented an application that is able to identify and count the number of people in either an image or a video. This is a good application for computer vision as it involves core computer vision methods in order to achieve the goal. Our first implementation, for images, relies on feature detection and non-maxima suppression and our first implementation for tracking in videos utilizes masks, contours, and background subtraction. This problem is also challenging because we must be able to segregate overlapping people as well as track people accurately as they move across a scene in a video, which results in continuously changing contours to track and identify properly. We expanded our application further to introduce a neural network to show more improved results and help with these issues.

Background

Human detection is a well documented computer vision problem with applications becoming more prevalent in our everyday lives. We implemented a web-based application which processes images and videos to detect occurrences of humans and videos within each image or frame. Images and videos were processed through several algorithms, namely, cascade classifiers, background subtractors and deep neural networks.

We first began this project by examining research findings and datasets presented by the CAVIAR research group [1]. These results helped us build a conceptual model of the problem at hand and helped us learn an important part of the history behind object detection.

The first detection method that was explored was human detection through Haar feature-based cascade classifiers developed by Paul Viola and Michael Jones [2]. The algorithm utilities are available within the OpenCV library for public use [3]. A cascade classifier is a 2-class neural network which is trained on images with and without the

object of interest. Cascade classifiers apply kernels of each haar-like feature to detect its presence. A few haar-like features can be observed below [4].

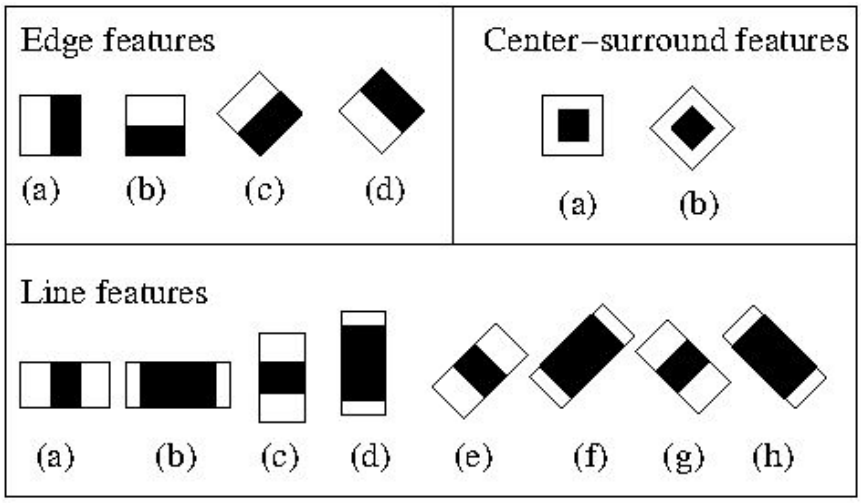


Figure 1: Example haar-like features [4].

After applying each kernel, we can uncover the possible presence of each haar-like feature. Once a list of features have been extracted from an image, they are ordered (cascaded) to be as computationally efficient as possible.

The method we researched to implement as our initial algorithm was a motion detection and background subtraction. This relied on the OpenCV provided functions. OpenCV is a free computer vision and machine learning library that is open source [3]. Our application used the createBackgroundSubtractorMOG2() algorithm within this library [3]. This estimates the background based on the first few frames of the video and then we compare that to subsequent frames [3].

The second object detection method that was explored was object detection through a trained neural network. Our neural network model was trained by loading in the pre-trained weights and configuration files provided by YOLOv3 [5]. This model makes predictions about whether or not objects belong to specific, identifiable

categories, such as a person or dog, by sending the image, as input, to the neural network. Most implementations apply the model to various parts of the image to make the prediction whereas YOLOv3 applies the model to the image as a whole[3]. YOLOv3 is accessible through the opencv library [3]. The trained neural network is then used to perform human recognition in images and videos.

We used the Django python module [6], in order to create a website that acts as an interface between our algorithms and the user. Django allows you to create a webserver using a python frontend and backend. The website allows a user to upload their own videos or images to test against our algorithms

Finally the last built in libraries we used were numpy and imutils. Numpy is a common python package that is used from scientific computations [7]. We used this to help with calculating key computer vision equations to solve our problem. Imutils is a package that has built in functions to help with image processing [8]. We use this to implement a non-maxima suppression algorithm.

For more specific details about the specific implementation of these algorithms please see our approach section.

Approach

For human detection in images the first implementation we did utilizes a cascade classifier. For this we initially blur the image, using a Gaussian Blur; To remove noise and help reduce spurious detections.

$$\sigma = 2 \qquad I = Image \qquad K = \begin{bmatrix} 0.10186806 & 0.11543164 & 0.10186806 \\ 0.11543164 & 0.13080118 & 0.11543164 \\ 0.10186806 & 0.11543164 & 0.10186806 \end{bmatrix}$$

$$\mathrm{I}_{Blurred} = I * k$$

We then attempt to classify faces in the picture, as there will be one face per person. Should we not be able to detect any faces we added an additional classifier to detect bodies. This helps us detect people that are too far away for their face to be detectable or people not facing the camera.

Given feature list = a list of features created from a cascade classifier file
- The list of features make up the object, that is going to be detected
Each feature = a matrix containing black and white values

$$\text{i.e a line feature} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Each filter is convolved with the image in order to determine if it is valid, if a feature is determined valid on a section of the image, only then will subsequent filters be tested in that section.

$\text{Image} = \text{I} \qquad \text{Feature List} = \text{F}_l$

$$\text{f(n)} = \begin{cases} I * F_l[n] & \text{if } n = 0 \\ I * F_l[n] & \text{if } n > 0 \text{ and } I * F_l[n-1] > 0 \\ 0 & \text{otherwise} \end{cases}$$

[2]

After this we loop through our classified objects and create a rectangle around them. We then apply non-maxima suppression with an overlap threshold to ensure people are only counted once per image, and to further help with spurious detections. Now, since we should only have real detections left, we loop through our detected people and draw rectangles around them. We then count how many people make it through this threshold and output the final image.

For video detection the first implementation we did utilizes a background subtraction technique. This algorithm loads the video in and initializes a kernel and background subtractor. We use the built-in one provided by opencv by using the cv2.createBackgroundSubtractorMOG2() method [3]. For each frame in the video we

then create a mask using the kernel and background subtractor. This is done by applying the previously mentioned background subtractor to each frame.

$$\mathrm{F}_{mxn} = \text{An image frame in the current video, with size m by n}$$
$$\mathrm{B}_{mxn} = \text{An estimation of the background of the current frame F}$$
$$\mathrm{M}_F = \text{The output mask of the frame F}$$
$$\mathrm{M}_F = F_{mxn} - B_{mxn}$$

After that we use the previously mentioned kernel to reduce noise in the mask to help with invalid detections. We then threshold, rather than using the built in shadow detection provided by opencv, the mask to filter out their shadows, since the shadows resulted in an object big enough to be a person. Since the shadow was not as intense, the threshold would filter out its lower pixel values and allow true “person” pixels to remain.

$$\mathrm{Output(i,j)} = \mathrm{M}_F(i,j) \text{ if } M_F(i,j) > Threshold$$

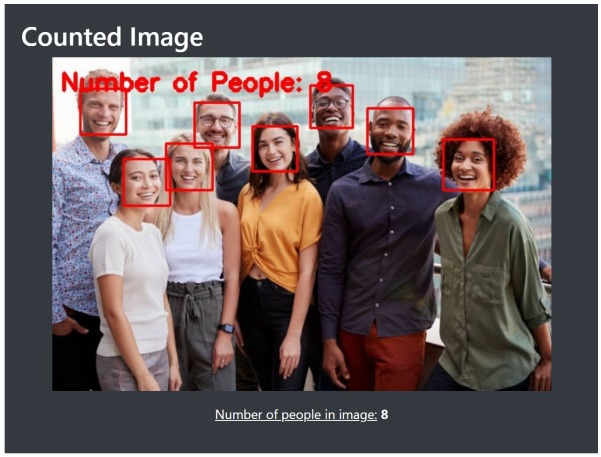
After that we create a contour around the resulting people to mark their outline. We then create rectangles to enclose these contours. We run these rectangles through the non-maxima suppression algorithm using the imutils.non_max_suppression method with an overlap threshold [8]. This helps to ensure people are not counted twice if their contours are not continuous as the two rectangles are still close enough for us to identify them as one person rather than two. Finally, we draw the suppressed rectangles around the contours on the output video and count the number of rectangles, or people, we have detected.

After implementing the first methods for images and videos we were able to accurately track people. However, there were still some specific cases in which people were too close and it only detected one person or if a person walked behind an object that partially occluded them they may not be detected. To further our accuracy of identifying people we expanded our project to introduce a neural network. For this we used YOLOv3(You Only Look Once) [5].

Our implementation of YOLOv3 for human detection in images first builds the model by loading the provided weights and configuration and then feeding them to a deep neural network. Our implementation uses the opencv deep neural network by using the cv2.dnn.readNetFromDarknet() function [3]. Note we don't use the actual YOLOv3 neural network but our own implementation of it using its training data. We then build the input, called a blob or binary large object, for the neural network from the image and send it to the network. We then loop through the output of the network which is another blob that contains the detections. We loop through each detection in the output and send it through a parser to determine if it passes as a person. The detection contains information about the bounding box surrounding the detected object. In addition to that it contains the neural network's probability of the object belonging to each one of its identifiable categories, as it can identify more than just people. We then make sure that a human is the most probable object and, if it is, we threshold the probability to ensure that only detections with high confidence are included. We then draw boxes around the objects that pass the threshold and count the number of people we detect. The process for our implementation of YOLOv3 for videos follows a very similar process for how it works with images. It builds the model using the same training data. We extract the video data to ensure we can output with the same data, such as resolution, video length, and the video's frames per second. We then loop through each frame and send the frame as an input to the neural network. It counts people using the same method as images and outputs how many people it detected in that frame and writes it to the output video. After we loop through all the frames we output the video.

On top of all these implementations our application runs on a Django web server [6]. This allows for custom videos and images to be used for testing purposes as it allows for videos to be uploaded dynamically. We don't have to have our videos in a predefined location and can browse the computer we are on for the video or image we want to run through the program.

Results



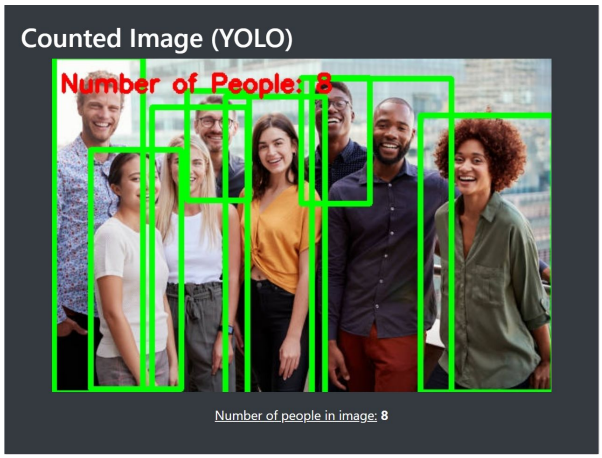


Figure 1.1 Cascade Detector In An Image

Figure 1.2 Yolo Detector In An Image

- As you can see in both Figure 1.1 and 1.2, our algorithm and our implementation of Yolov3 both detect the correct number of people

Figure 2.1 Cascade Detector In An Image

Figure 2.2 Yolo Detector In An Image

- Figure 2.1 and 2.2 both show that each implementation accurately counts the number of people present

Figure 3.1 Cascade Detector In An Image

Figure 3.2 Yolo Detector In An Image

- Figure 3.1 shows a specific example of a person not being detected with our initial algorithm. We attempted to play around with threshold values and other hyperparameters to fix this issue but it would not detect the fifth person. This is why we extended our project to include a deep neural network to help with identification. In Figure 3.2, you can see the fifth person is detected correctly.

Figure 4.1 Cascade Detector In An Image

Figure 4.2 Yolo Detector In An Image

- Figure 4.1 and 4.2 show both our algorithm and the Yolo network accurately classifying the number of people during a video

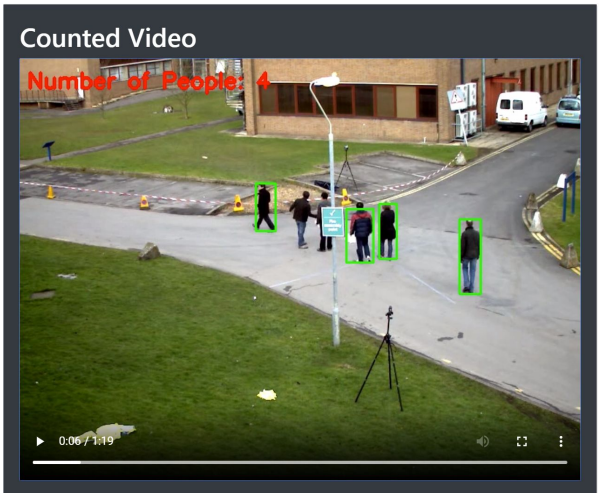
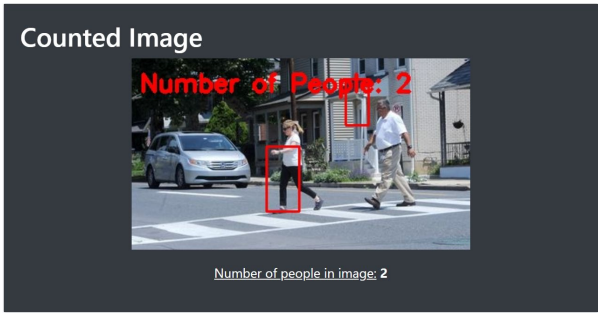


Figure 5.1 Cascade Detector In A Video

Figure 5.2 Yolo Detector In A Video

- Figure 5.1 shows that our algorithm can not detect the two people standing behind the pole and sign in the video. This is due to them being stationary for many frames and being detected as part of the background. Figure 5.2 shows how Yolo improves this detection by being able to detect the person who is less covered. Both still are unable to detect the person whose majority of their body is behind the pole.



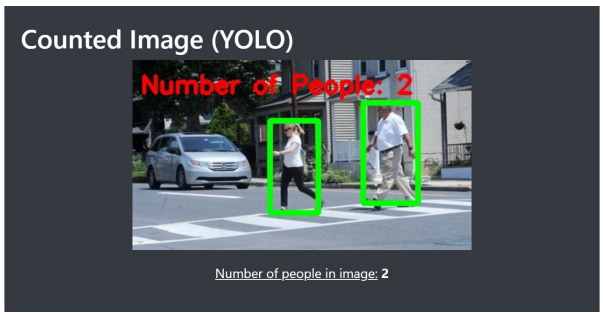
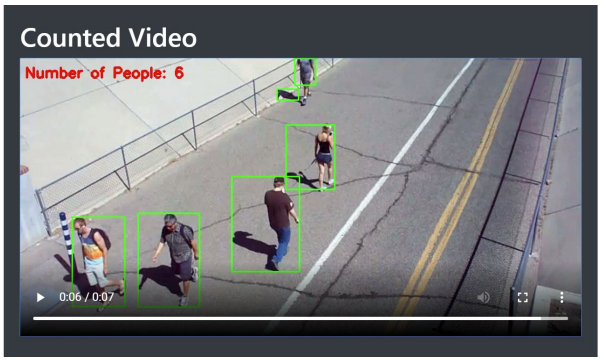


Figure 6.1 Cascade Detector In A Video ***Figure 6.2 Yolo Detector In A Video***

- Figure 6.1 shows an error in our algorithm but still counts properly. Figure 6.2 shows how Yolo solves this issue.



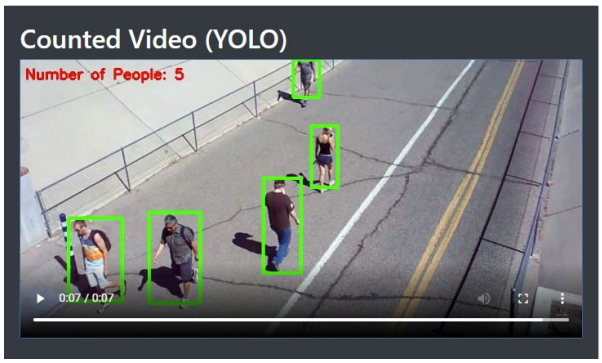


Figure 7.1 Cascade Detector In A Video ***Figure 7.2 Yolo Detector In A Video***

- Figure 7.1 shows that our algorithm sometimes detects shadows as people if they are solid enough to pass our threshold. Even after changing the threshold and other flexible parameters it would still detect shadows as people in some specific situations. Figure 7.2 shows Yolo does not have this issue.

List of Work

Equal work was completed by all members of the group

GitHub Page

https://github.com/JakeJazokas/HumanCounter/tree/master

References

[1] CAVIAR Project/IST 2001 37540, n.d. http://homepages.inf.ed.ac.uk/rbf/CAVIAR/.

[2] P. Jones, Paul Viola, and Michael Jones. "Rapid Object Detection Using a Boosted Cascade of Simple Features." . In *University of Rochester. Charles Rich* (pp. 905–910).2001.

[3] Bradski, G.. "The OpenCV Library".*Dr. Dobb's Journal of Software Tools* (2000).

[4] Paulo Menezes, José Carlos Barreto, and Jorge Dias. "Face tracking based on haar-like features and eigenfaces".*IFAC Proceedings Volumes* 37, no.8 (2004): 304 - 309.

[5] Redmon, Joseph, and Ali, Farhadi. "YOLOv3: An Incremental Improvement".*arXiv* (2018).

[6] “Django.” Django. Django Software Foundation, April 1, 2020. https://djangoproject.com/.

[7] Walt, Stéfan van der, S. Chris, Colbert, and Gaël, Varoquaux. "The NumPy Array: A Structure for Efficient Numerical Computation".*Computing in Science & Engineering* 13, no.2 (2011): 22-30.

[8] jrosebr1. “jrosebr1/Imutils.” GitHub, August 18, 2019. https://github.com/jrosebr1/imutils.