CSC420 Group Project Report - Social Distancing Detector

Khizr Khan and Shuhan (Freddy) Zheng

1 Abstract

Due to the COVID-19 pandemic, many people need to maintain a safe distance between each other in order to prevent transmission of the virus. We want to implement, benchmark, and report on computer vision-based social distancing detection. By doing so, we believe that we can contribute to the creation of more effective detectors for determining whether people in images and videos are staying a safe distance from each other. To accomplish this, we have built a framework which takes as input a video and performs social distancing detection. This framework detects humans within the video, applies homography techniques to adjust for depth and distance, detects whether humans are at least a safe distance of each other (2 meters), and outputs a labelled video in which social distancing violations are visually marked. This framework implements and allows two leading object detection approaches to be used interchangeably (Faster R-CNNs and YOLOv3). Information is logged as it runs, thus allowing for comparisons between the two approaches. We use this framework to detect social distancing in 4 different video datasets with both Faster R-CNNs and YOLOv3, and perform analysis on the data. We derive several insights on social distance detection from this data and conclude with a comparison on the performance of Faster R-CNNs versus YOLOv3 for social distancing detection, and we finally present that a social distancing detector using Faster R-CNNs correctly detects social distancing violations 2.52 times more frequently than YOLOv3.

2 Introduction and Literature Review

There have been several attempts at implementing a social distancing detector , as well as existing papers comparing the precision and efficiency of object detection models at the task of detecting and classifying objects [6]. However, as of the time of proposing this project, we did not find existing research on how the choice of object detector influences the final outcomes for social distancing detection - whether a violation exists, the number of violations, etc. Detecting humans is a crucial step in social distancing detection, but it is also the step in which experimentation is valuable due to the black-box nature of neural networks. Because cameras applying social distancing detection will tend to remain in place, homography is a problem which can be manually solved with user calibration such that it can predictably work (this will be further discussed in section 3). We thus focus on object detection and its impact on final results, where we believe experimentation and analysis will provide the most value to the development of social distancing detectors.

One approach linked by the Professor in the project title description is using faster R-CNNs [1]. The link includes an implementation of a basic approach to detecting social distance using this approach by Aravind Pai. It performs object detection using pre-trained model weights on objects in an image to create boxes around detected people, and it computes the euclidean distance between them. But (as the author acknowledges), this is only a starter and it's lacking important features: for example, the euclidean distance does not take into account perspective and only computes distance in terms of pixels, so distances of people further away from the image's camera perspective do not correspond to distances of people closer to the camera.

Landing.ai, a company from Deep Learning researcher Andrew Ng, has also created a social distancing detector [2]. This detector is stated as the inspiration for the previously discussed detector in article [1]. It's more advanced and computes the location of people within an image/video from a bird's eye view, but the implementation details for this approach is not as readily available. Another implementation (by Aqeel Anwar) also detects social distancing using a deep neural networks [3]. However, unlike approach 1, this approach performs additional processing and detects distances between objects from a bird's eye view.

A paper by Singh et. al [6] proposes a framework for social distance detection through deep learning. They detail their framework on page 7 of the linked paper, and they use a YOLOv3 model trained on surveillance footage and compute the depth of an object via the formula d = ((2 3.14 180)/(w + h 360) 1000 + 3) where w and h are the width and height of the bounding box, with reference to [7]. This

allows for depth-adjusted distance measurement without a bird's eye view. Afterward, they compute the pairwise L2 norm of the bounding boxes and use the dense matrix of the L2 norm to assign the neighbors for each detected person that satisfies the closeness threshold. Any detected person which meets the closeness property is assigned a neighbor or neighbors forming a group, and the formation of groups indicates the violation of social distancing. This is quantified by the measure np/ng, where ng is the number of groups/clusters identified and np is the total number of people found in close proximity. While this paper considers the performance different object detection approaches in the context of classifying objects, it does not relate these numbers to the final performance measure for social distancing detection.

However, this paper chooses to use YOLOv3 as the object detection algorithm based on its performance on class labelling rather than in the context of the complete social distancing detection framework, and it only shows the output of the detector rather than measuring the output against a set of ground truth labels. Our project will attempt to build on this existing paper by improving on these two points.

In order to gain some understanding of Faster R-CNNs and YOLOv3 which are two of the leading approaches for object detection, we also reviewed some existing literature and publications discussing them and their comparative performance in the context of detecting objects [6][14][15][16].

3 Methodology, Results, Experiments

3.1 Social Distancing Detector Framework (Methodology)

3.1.1 Inputs and Outputs

All relevant code has been submitted alongside this report. See the README.md file for instructions on how to install dependencies and run the code. The social distancing detector as input:

- A filepath to the video to operate on,
- A method to object detection (supports Faster R-CNNs and YOLOv3)
- The maximum number of frames to process, with an option to compute all frames.
- A filepath to write the logs.
- A choice of whether to run on the CPU instead of GPU if using Faster R-CNNs.

The detector outputs:

- A directory named "frames_{video_name}" will be generated containing all frames prior to labelling with the social distancing detector. If this directory already exists and there is an appropriate number of frames contained inside, then this step will be skipped.
- A directory named "labelled_frames_{video_name}" be generated containing all frames after to labelling with the social distancing detector. Like the previous directory, this step will be skipped if such the directory already exists.
- If an appropriately-named output video file does not already exist in the directory: a video named "labelled_video.avi" of equal length to the input video, but with detection people and social distancing violations labelled.
- A JSON file containing information such as social distancing violations, the number of detected people, and information on homography.

3.1.2 Pipeline

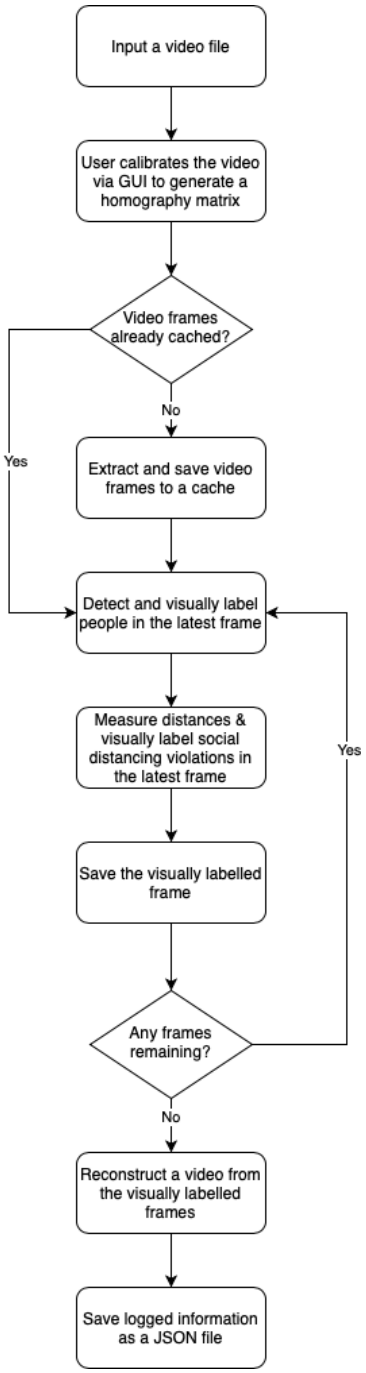

Fig. 3.1.1.1: An illustration of the steps in the social distancing detector's pipeline.

The high-level pipeline of the detector are as follows.

1. Preparatory steps: The inputs are processed in order to create an appropriate logger, and the chosen object detection neural network will have its saved weights loaded. At every step below, the relevant information will be saved to the logger.

2. The user selects four points in an image corresponding to a rectangle in the real world. The user is then asked to enter the dimensions of this rectangle in real-life. A homography matrix is produced based on this calibration process.

3. If a cached directory containing all frames of the input video is not detected, then a cache directory will be created to store all frames of the input video on the file system. This is to avoid unnecessary memory usage.

4. Each frame will be read from the file system cache, processed, and saved to the file system in an appropriate output directory. During the processing of each frame:

   (a) All humans will be detected using the chosen object detection method (Faster R-CNN or YOLOv3) and logged.

   (b) Social distancing violations are detected based on eucledian distance adjusted for the homography matrix and the user-inputted dimensions.

   (c) All detected humans and social distancing violations are labelled on the frame and the frame is saved.

5. The processed frames with labelled people and social distancing violations are then used to reconstruct an output video with equal length to the input video. This output video has detected humans and social distancing violations labelled.

6. All relevant logged information will be saved in a JSON file.

3.1.3 Software Design / Architecture

There are 5 files which are used for the social distancing detector:

- object_detection.py: Contains implementations for the object detection methods used in the project, using existing packages such as detectron2 and darknetpy. An interface is defined for any object detector used, and each object detector adheres to this interface so that they can be used interchangeably in the detector.

- metrics.py: Contain implementations for metrics which are logged and/or modify a frame as it is being processed. Metrics are also designed to be interchangeable and adhere to a defined common interface. The social distancing detection via homography-adjusted eucledian distance, as well as the image labelling, is a metric defined here. Note on design: while we originally planned to perform our analysis real-time largely using metrics, due to project delays we instead decided to create a separate script (analysis.py) below which performs experiment analysis based on the logs rather than in real-time as the detector runs.

- calibration.py: Contains the implementation of the calibration process, where users select a rectangular patch on the ground and and provide dimensions via a graphical user interface for the purposes of homography.

- homography.py: Contains code which generates a homography matrix to be used by the detector, based on inputs from calibration.py.

- detector.py: The driver of the program. It contains the code which processes each frame using implementations in the modules listed above. It also contains the definition of the Logger and contains the functions which convert a video to frames and then back from frames to a video, as well as the caching of these frames.

Another file, log_analysis.py is used for analysis of the logs and produces the results found in experiment analysis section of the report.

3.1.4 Object Detection

For object detection, we use both Faster R-CNNs and YOLOv3 in order to compare the social distancing detection outcomes using each approach.

Because both Faster R-CNNs and YOLOv3 use Convolutional Neural Networks (CNNs), we will briefly discuss the implementation of CNNs.

1. An filterbank consisting of many convolution filters is first applied to an input image, in order to create a 3D matrix that which is an output map with many channels (one for each filter)[20].

2. Afterwards, this matrix is repeatedly transformed through several additional layers that each apply filters and potentially techniques (such as max pooling, which results in invariance to small shifts in position)[20].

3. After passing through these layers, the matrix is passed through one or two fully/densely connected layers in which the output is the dot product between the filter and the input[20].

4. Finally, the output of these densely connected layer(s) is fed to a classification layer, which is a vector in which each dimension informs us of the chance of the original input image belonging to a certain class[20].

However, a standard CNN does not suffice for the task of object detection. This is because there can be multiple objects of interest, so the length of the final output layer is not fixed [16]. It is thus necessary to choose different regions from an input image and classify whether there is an object of interest in each region. However, attempting every possible region would be very computationally expensive, and Faster R-CNNs and YOLOv3 both provide a more efficient solution to this problem.

Faster R-CNNs build upon R-CNNs, so R-CNNs will first be explained. A R-CNN uses an algorithm called selective search to find 2000 candidate regions of interest to feed into a CNN, rather than of feeding a very large number of regions into a CNN. Selective search, at a high level, segments the image into many candidate sub-regions and uses a recursive algorithm to combine smaller candidate regions into larger ones [16]. These regions are fed into a CNN which extracts features from the regions, and these features are fed into a Support Vector Machine in order to classify whether an object of interest exists in the candidate region. The algorithm also predicts the coordinates of the bounding box for the object in addition to its presence.

Skipping the second-generation Fast R-CNN, Faster R-CNNs are the third-generation R-CNN algorithm and differs from the original R-CNN in that selective search is not used at all and a Faster R-CNN provides the image to the CNN rather than providing its regions to the CNN, in order to produce a convolutional feature map [16]. A neural network is then used to predict the candidate regions. A Region of Interest (ROI) pooling layer is then applied to the convolutional feature map and the candidate regions. ROI pooling allows a single feature map to be efficiently used to classify and provide bounding boxes for all the proposals generated by RPN in a single pass [21].

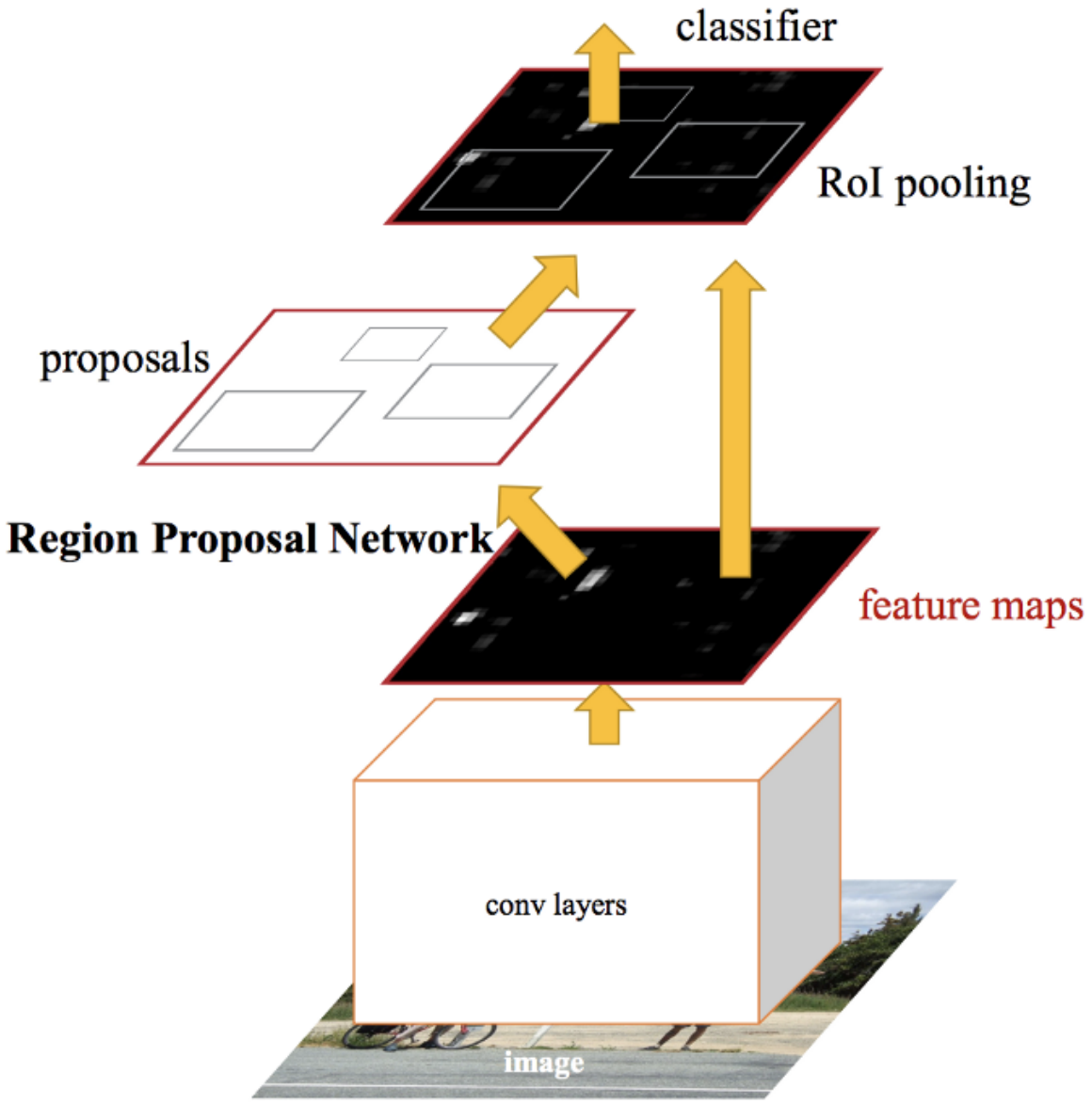

Fig. 3.1.4.1: A high-level diagram illustrating the steps used in a Faster R-CNN, sourced from [16]

YOLO, on the other hand, is a simpler and more efficient algorithm. YOLO only looks at portions of the image which have a high chance of containing objects of interest, and uses a single CNN to predict to predict the bounding boxes and class probabilities of the boxes [16]. YOLO takes an image and splits it into a grid, and within each tile of the grid, a certain number of bounding boxes are considered. A class probability is computed for each of the bounding boxes, and objects with probability beyond a certain threshold are classified and labelled [16].

As widely-used implementations of these models are available online, we choose to use available packages rather than re-implement these models. For Faster R-CNNs, we use Facebook's Detectron2 [17]. For YOLOv3, we use Darknet and its python binding Darknetpy [18][19].

It is worth noting that there are several generation of YOLO object detection implementations. We use YOLOv3 (published on March 2018) instead of YOLOv4 (published on April 2020) or YOLOv5 (published on July 2020) because of the longevity and stability of the older YOLOv3 as well as the greater support for existing packages. For example, darknet is the commonly used package for YOLO implementations, and darknetpy is the popular python binding for darknet, but darknetpy has issues supporting YOLOv4 and YOLOv5 due to how recent the newer approaches are.

3.1.5 Homography

To identify distances between humans in the images, we must first find the locations of each human detected. Even though the object detection library returns coordinates of the humans in the image frame we cannot use this information to find distances between the humans. We cannot use the euclidean distance between the human's image coordinates since the distance per pixel can vary across the image. This is because the image projection plane is not the same as the plane of the ground that the humans are standing on. To counter this issue we must transform the image to a birds eye view where the distance per pixel is uniform across the image. The first step in this process is to calibrate the camera. Then we must calculate the transformation (homography) matrix. Finally we can apply the transformation to the pixel coordinates to obtain real-world coordinate.

When the program begins, a calibration process is required by the user as seen in Figure 3.1.5.1. The first frame of the video is displayed to the user. The user is then prompted to select four points on the image that create a rectangle in the real world. Then the user is asked to enter the dimensions of this rectangle in the real world. The calibration process is then complete and the inputted values will be used later to create a homography matrix.

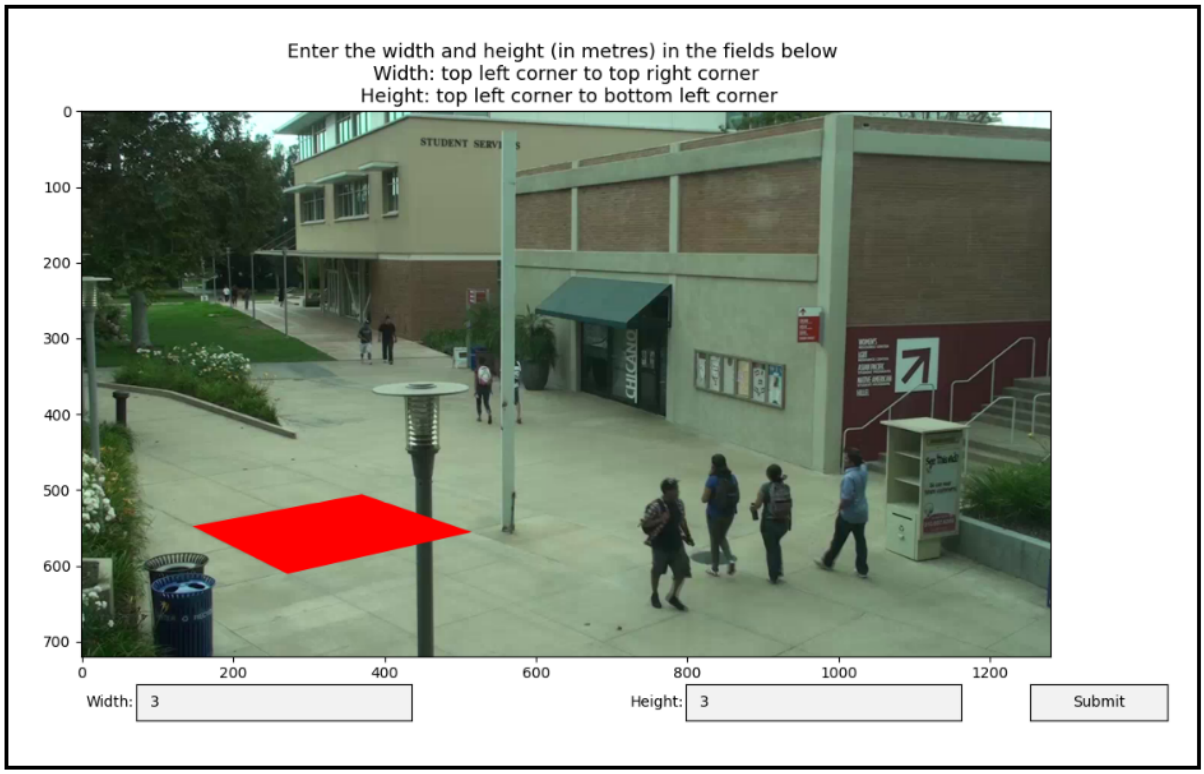

Fig. 3.1.5.1: Calibration process, VIRAT Dataset

To perform the birds eye transformation, a homography matrix H is required. $H = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}$

The product of a pixel coordinate and the homography matrix is the transformed pixel coordinates from the birds eye view perspective. This relationship can be observed in the equation below where H is the homography matrix, (x,y) are the image coordinates, and (x',y') are the transformed image coordinates.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

To create a homography matrix, we need 4 sets of matching coordinates of (x,y) to (x',y') which results in 8 unknown values. Fortunately, we obtained these values from the inputs taken from the calibration

process. The 4 points of the rectangle represent the original (x,y) coordinates in our unknown values. We can create the remaining transformed coordinates (x',y') using the dimensions of the rectangle as seen in Figure below.

(x, y)	(x', y')
(top left x, top left y)	(0, 0)
(top right x, top right y)	(width, 0)
(bottom left x, bottom left y)	(0, height)
(bottom right x, bottom right y)	(width, height)

Fig. 3.1.5.2: Calibration coordinates used to compute homography matrix

We can then find our homography matrix unknown values by reshaping the matrix into a vector instead and solving the following equation below. Refer to [8] to follow the reasoning behind this calculation. This problem is now a variation of the linear least squares problem and various strategies can be used to solve it: Our implementation uses a numpy library call to do so.

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 & -y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x'_2x_2 & -x'_2y_2 & -x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -y'_2x_2 & -y'_2y_2 & -y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x'_3x_3 & -x'_3y_3 & -x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -y'_3x_3 & -y'_3y_3 & -y'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x'_4x_4 & -x'_4y_4 & -x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -y'_4x_4 & -y'_4y_4 & -y'_4 \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Once the homography vector is found we can convert back into a matrix. Now the homography matrix can be used to calculate the true coordinates of the humans in the image.

Alternate homography approaches considered:

To tackle the homography issue, our initial goal was to implement a method that does not require any manual input required from the user so our detector would be convenient and accessible to users. Unfortunately this restriction creates issues of its own. In the end we chose the strategy that best suits a social distancing detector as its results in the most accurate results using the same homography matrix throughout the video. This choice was also made after consulting with our project advisor (Elsa) who described our method as an "elegant requirement" to "obtain the benefit of increased accuracy for distance estimation".

One alternate strategy we considered was discussed in a paper by Singh et. al [6]. They compute the depth of an object from the camera via the formula d = ((2 3.14 180)/(w +h 360) 1000 + 3) where w and h are the width and height of the bounding box . Then the depth value is added to the image coordinate of the humans as a 3rd coordinate to create a form of 3d coordinate for each human. The paper uses these coordinates to find distances between people and checks if the distance crosses a certain threshold. While the process used in this paper does require manual input, we chose not to proceed with this method since it is quite simple, but more importantly it is not very accurate. This is because using the depth from the camera as a third coordinate is not a valid substitute to true real-world coordinates. As a result this strategy was not deemed as suitable since the distance estimation would not be accurate enough for a social distancing detector.

3.1.6 Metrics / Logging

The following information is logged throughout the program's execution and saved in a JSON file each time the social distancing detector:

- The pixel coordinates of the humans detected in each frame.

- The number of detected humans per frame.
- The total frame count.
- The selected corners and inputted height/width by the user during the calibration step of the models.
- The homography matrix generated.
- Relevant information perrtaining to the metrics defined in metrics.py, such as whether eucledian distance was violated.

These saved logs are used for the purposes of experiment analysis and debugging.

3.2 Experiments and Data Collection (Methodology/Experiments)

All logged data files are uploaded on Google Drive and can be found in [13] in the bibliography or by clicking this text

3.2.1 Data Collection

When collecting datasets for this procedure, it is important to gather data that is relevant to the problem at hand. For our social distancing detector we performed tests on 4 sets of video datasets. We found that each of these datasets tested a key characteristic of the social distancing detection problem. Frames from these datasets can be viewed in Figure 3.2.1.1. The first dataset is the Oxford town centre Dataset which consists of "pedestrians in a busy downtown area" [9]. Pedestrians walking on sidewalks is a great example since it represents the most common outdoor scenario where people are in close proximity to others. The second dataset is a video of a European handball training game at a sports hall [10]. This dataset is especially useful to the social distancing detector problem since it consists of an indoors environment. As business and workplaces open back up around the world, social distancing rules will still be in place. Social distancing detectors may be used in higher frequency in indoor buildings in the near future. Offices may use detectors to maintain safe distance between employees and fast-food chains may use them to create socially distant waiting lines. There are plenty of applications for indoor uses of a social distancing detector which makes this dataset especially important.

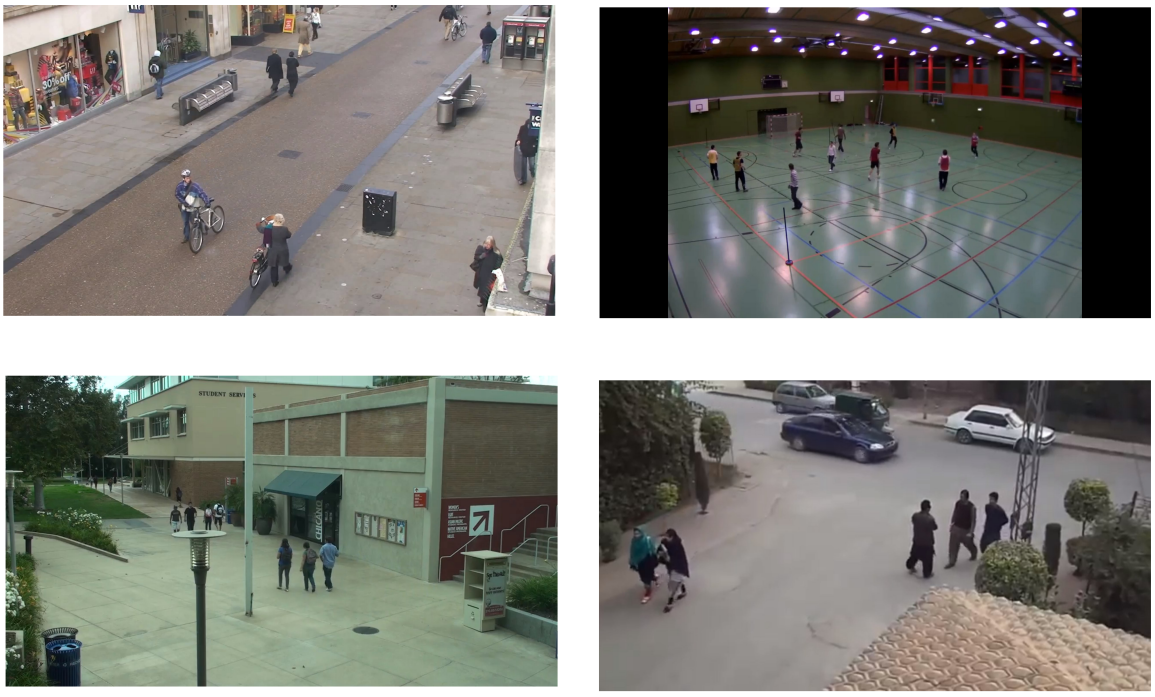

Fig. 3.2.1.1: Oxford Town Centre Dataset (top left), Sports Hall Dataset (top right), VIRAT Dataset (bottom left), Peshawar Dataset(bottom right)

The third dataset is the VIRAT dataset which is a video of students walking across a campus walkway [11]. This dataset is useful since it consists of subjects walking closer to the camera as well as far away from the camera as well. This change in distance to camera pushes the object detection techniques to their limits and identifies how close the subjects need to be to the camera for the detector to operate successfully. The fourth dataset is by Faisal Imtiaz and consists of pedestrians walking by in the city of Peshawar, Pakistan [11]. Covid-19 is affecting all countries of the world and countries may have to use social distancing detectors since a working vaccine still seems in a distant future. This dataset aims to test whether the object detection techniques can still perform well when the human subjects are of non-western societies. Some of these differences could arise for varying clothing, facial features, environments and many other factors.

3.2.2 Creating Data Results

Running the detector on a video can take large amounts of time similar to most image processing operations. Even though our first data set is roughly 30 seconds long, it takes the detector roughly 8 hours to analyze the entire video. To run tests on the datasets it is simpler to first log the characteristics of the video that we want to compare and later test on the logs themselves. This process can be made even quicker if the logs are created using a machine with a dedicated GPU. As a result we ran our detector on a Microsoft Azure server equipped with a dedicated GPU used for data processing. To successfully compare the two object detection techniques, we ensured that the only variable between the logs of the two techniques were the techniques used themselves. This meant that the calibration inputs had to be the same across the two techniques as well. To meet this requirement, we chose to manually enter the calibration inputs in the code rather than rely on mouse input. The outputted log files are uploaded online and are linked in the bibliography[13].

3.3 Experiment Analysis (Experiments/Results)

The implementations in the script analysis.py are used to produce the results in section 3.3.1 and 3.3.6, using the 8 JSON log files in the directory experiment_logs. These log files are generated from applying the social distancing detector to the 4 discussed video datasets, using Faster R-CNN and YOLOv3 for object detection with each dataset.

3.3.1 Overview of Results

Fig. 3.3.1.1: Number of people detected per frame by Faster R-CNNs vs. YOLOv3

We see here that Faster R-CNNs generally detect many more people per frame than YOLOv3. While

there is occasionally a false positives as well (see the following examples), based on inspection these cases are exceptionally rare. Thus supports the general consensus that Faster R-CNNs are more precise than YOLOv3 (at the cost of efficiency).

Detected Violating Frames out of Total Frame Count

Violating Frames	Oxford Town Centre Dataset	Sports Hall Dataset	VIRAT Dataset	Peshawar Dataset
Faster R-CNN	911/927	252/450	584/450	694/926
YOLOv3	800/927	216/450	216/450	540/926

Accordingly, we also see many more frames are detected by Faster R-CNNs as violating social distancing measures, compared to the frames detected by YOLOv3 per dataset. More analysis will be performed, but first several example frames from the detection process will be displayed.

3.3.2 Comparing Faster R-CNNs and YOLOv3 - Precision and False Positives

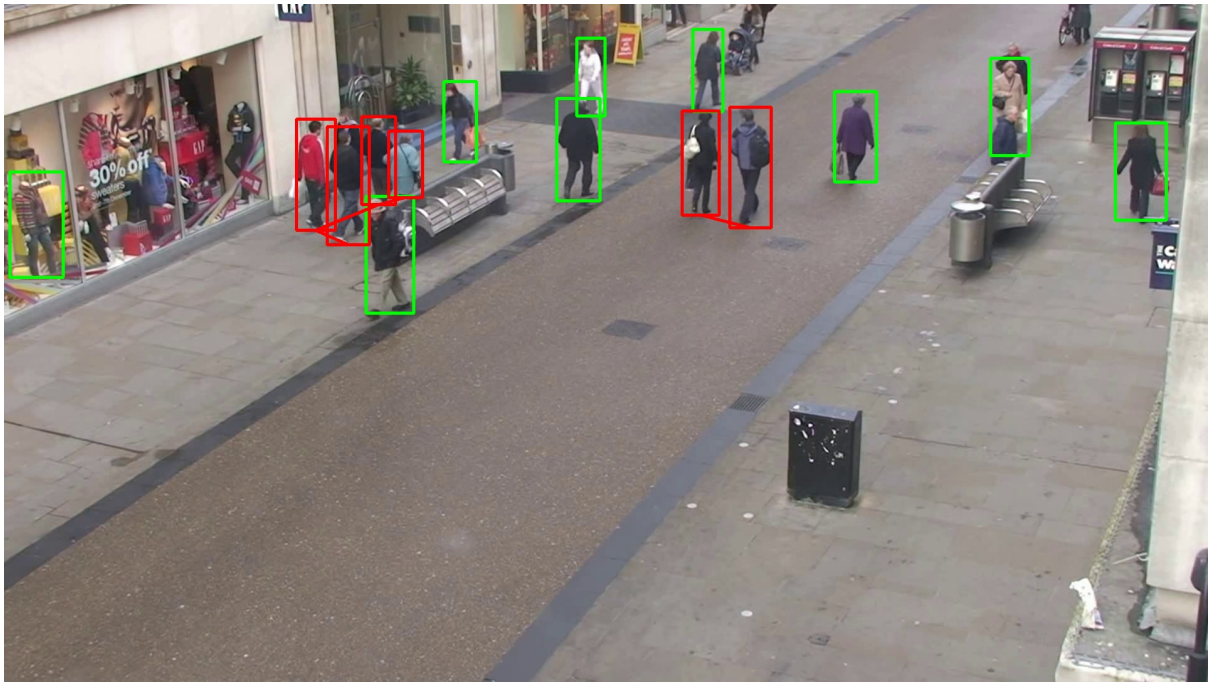

Fig. 3.3.2.1: A frame assessed using Faster R-CNN, Oxford Town Square Dataset

Fig. 3.3.2.2: A frame assessed using YOLOv3, Oxford Town Square Dataset

Based on inspecting the above frames, we find many cases in which YOLOv3 fails to detect a person in the image whereas Faster R-CNN succeeds. In the above example, Faster RCNNs detect a cluster of 4 people all violating social distancing rules in the upper left area of the image, whereas YOLOv3 fails to detect 3 of the 4 people and thus believes that there is no violation in that area.

In this same frame however, we see that Faster R-CNN incorrectly detects one of the mannequins to the left as a human, whereas YOLOv3 does not. Based on inspection of the results, such false positives are uncommon but do exist.

3.3.3 Comparing Faster RCNNs and YOLOv3 - Overlapping People Example

Fig. 3.3.3.1: A frame assessed using Faster R-CNN, Sports Hall Dataset

Fig. 3.3.3.2: A frame assessed using YOLOv3, Sports Hall Dataset

We see here that Faster R-CNNs detects more precisely and detects two out of three overlapping people at the center of the image, whereas YOLOv3 fails to detect any one of the three.

3.3.4 Comparing Faster R-CNNs and YOLOv3 - Long-Distance Example

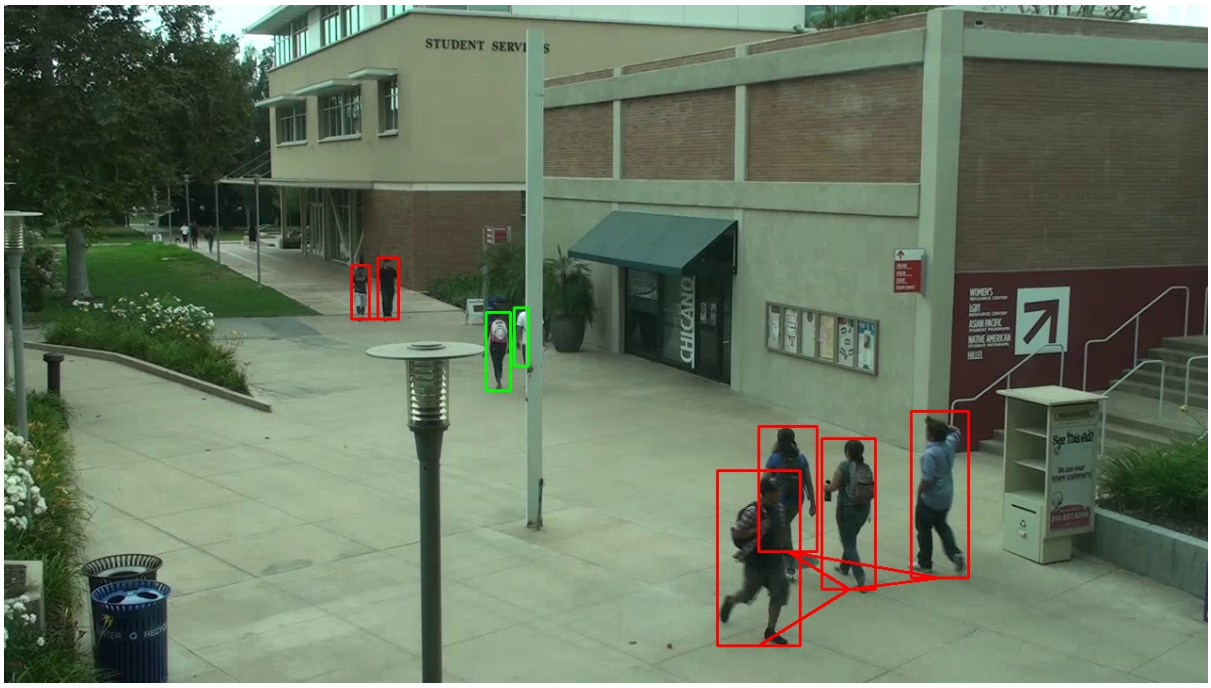

Fig. 3.3.4.1: A frame assessed using Faster R-CNN, VIRAT Dataset

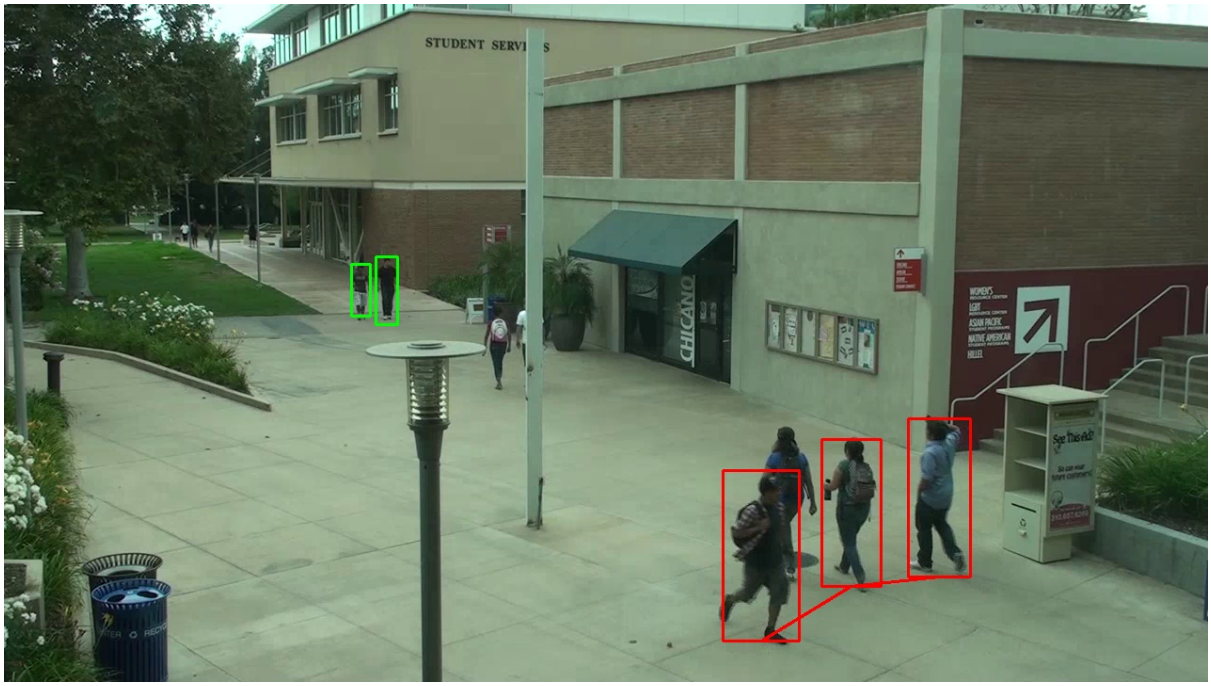

Fig. 3.3.4.1: A frame assessed using YOLOv3, VIRAT Dataset

In this example, we once again see that Faster R-CNN is more effective at detecting nearly individuals than YOLOv3. However, both fail at detecting individuals far in the distance.

3.3.5 Comparing Faster R-CNNs and YOLOv3 - Covered People Example

Fig. 3.3.5.1: A frame assessed using Faster R-CNN, Peshawar Dataset

Fig. 3.3.5.1: A frame assessed using YOLOv3, Peshawar Dataset

In this example, both Faster R-CNN and YOLOv3 fail to detect 3 people: the person on the top-left partially covered by a small tree, the person on the bottom right partially covered by a roof, and a driver in the centre whose body is visible but is sitting within a mini-car. This is an example showing that both Faster R-CNNs and YOLOv3 are inconsistent when detecting indivduals who are partially covered by something else.

3.3.6 Impact on Pairwise Social Distancing Detection Outcomes

Consider again the previous chart discussing the number of people detected per frame:

Fig. 3.3.1.1: Number of people detected per frame by Faster R-CNNs vs. YOLOv3

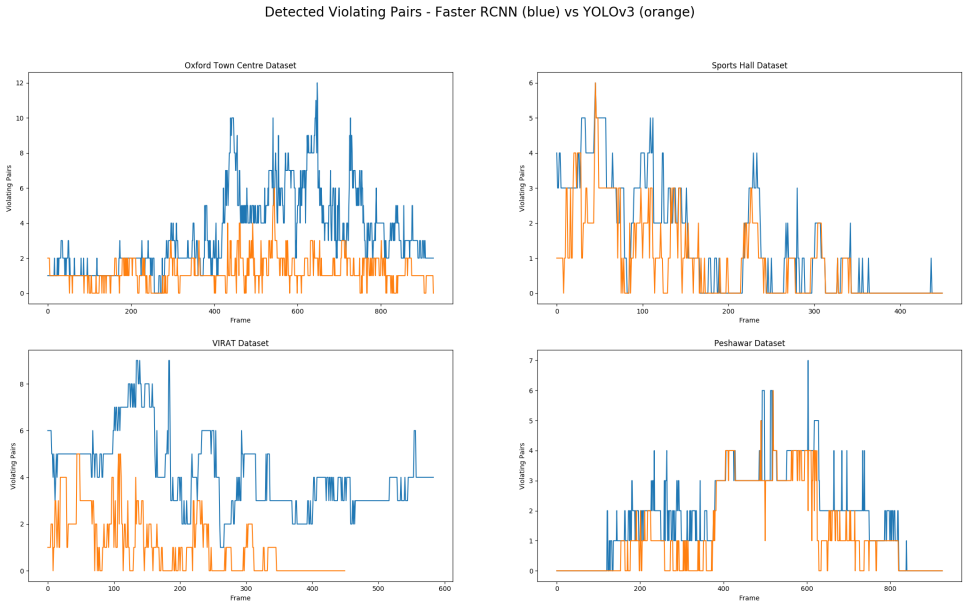

Fig. 3.3.1.1: Number of pairs detected as violating social distance norms (2 meters of distance) per frame by Faster RCNNs vs. YOLOv3

Comparing the two approaches, we see that Faster R-CNNs detect social distancing violations significantly more precisely than YOLOv3.

Ratio of Violating Social Distancing Pairs found by Faster R-CNN vs YOLOv3

Table	Oxford Town Centre Dataset	Sports Hall Dataset	VIRAT Dataset	Peshawar Dataset	Overall
Ratio	2.80	1.55	5.3	1.46	2.52

Over all video datasets that we experiment on, Faster R-CNNs detect significantly more social distancing violations. Based on inspection of the labelled video datasets, we found practically no false positives, so it can be assumed that practically all of the detected pairs are true positives and are thus correct detections. This claim can be verified by inspecting the labelled videos for these datasets that are stored in the linked Google Drives. In total over the 4 datasets, we find that Faster R-CNNs detect 2.52 times more social distancing violations than YOLOv3.

4 Conclusion

4.1 Concluding Statement

Based on the above analysis, we conclude that social distancing detectors using faster R-CNNs are substantially more precise than detectors using YOLOv3, correctly detecting roughly 2.52 times more social distancing violations over our experimental video datasets.

4.2 Notes and Caveats

4.2.1 Efficiency

There are many existing papers which have concluded that YOLOv3 is significantly more efficient than Faster RCNNs, such as Singh et al.'s paper [6]. We do not compare the efficiency of these two approaches in our report because there is already an abundance of such comparisons available. We do acknowledge that while Faster R-CNNs may be significantly more precise than YOLOv3, YOLOv3 can still be used to create a reasonably effective social distancing detector that runs more efficienctly than Faster R-CNNs. This application is quite valuable, particularly in cases where real-time detection is paramount.

4.2.2 Precision

Based on inspection, we see few to none false positives in the labelled videos, so we believe that the ratio of true and false positives detected in this video is roughly representative of the number of true positives.

We previously considered manually labelling several datasets just to completely eliminate the possibility of false positives, but after the assigned Teaching Assistant for this course noted that it would be too laborious, we later dropped the idea.

4.2.3 Usage of Existing Packages

We utilize implementations of Faster R-CNN and YOLOv3 from existing packages. There is reason to believe that both implementations are working correctly - Detectron2 is supported by Facebook AI Research and Darknet is one of the most widely sourced implementations for all generations of YOLO models. However, as we use pre-trained weights provided by these implementors, these models also learned from differing datasets which could produce some degree of variance in the ultimate results.

6 References

References

[1] Pai, Aravind, and Aravind. "Build Your Social Distancing Detection Tool Using Deep Learning." Analytics Vidhya, www.analyticsvidhya.com/blog/2020/05/social-distancing-detection-tool-deep-learning/.

[2] "Landing AI Creates an AI Tool to Help Customers Monitor Social Distancing in the Workplace." Landing AI, landing.ai/landing-ai-creates-an-ai-tool-to-help-customers-monitor-social-distancing-in-the-workplace/

[3] Anwar, Aqeel. "Using Python to Monitor Social Distancing in a Public Area." Medium, Towards Data Science https://towardsdatascience.com/monitoring-social-distancing-using-ai-c5b81da44c9f

[4] Edinburgh Informatics Forum Pedestrian Database homepages.inf.ed.ac.uk/rbf/FORUMTRACKING/

[5] "Discovering Groups of People in Images." Group Discovery, cvgl.stanford.edu/projects/groupdiscovery

[6] Singh, Narinder, et al. "Monitoring COVID-19 Social Distancing with Person Detection and Tracking via Fine-Tuned YOLO v3 and Deepsort Techniques." https://arxiv.org/abs/2005.01385

[7] Paul-Pias. "Paul-Pias/Object-Detection-and-Distance-Measurement." GitHub, 5 May 2020, https://github.com/paul-pias/Object-Detection-and-Distance-Measurement

[8] Rezanejad Murteza, "Matching Planar Objects in New Viewpoints ... And Much More - via Homography", CSC420 Intro to Image Understanding Lecture Slides

[9] Oxford Town Centre video and data, [online]. [Accessed July 2020], http://www.robots.ox.ac.uk/ActiveVision/Research/Projects/2009bbenfold$_h$*eadpose/project.html*

[10] Horst Possegger, Matthias Rüther, Sabine Sternig, Thomas Mauthner, Manfred Klopschitz, Peter M. Roth and Horst Bischof. "Unsupervised Calibration of Camera Networks and Virtual PTZ Cameras" Proc. Computer Vision Winter Workshop (CVWW), July 2020,

[11] A Large-scale Benchmark Dataset for Event Recognition in Surveillance Video" by Sangmin Oh, Anthony Hoogs, Amitha Perera, Naresh Cuntoor, Chia-Chih Chen, Jong Taek Lee, Saurajit Mukherjee, J.K. Aggarwal, Hyungtae Lee, Larry Davis, Eran Swears, Xiaoyang Wang, Qiang Ji, Kishore Reddy, Mubarak Shah, Carl Vondrick, Hamed Pirsiavash, Deva Ramanan, Jenny Yuen, Antonio Torralba, Bi Song, Anesco Fong, Amit Roy-Chowdhury, and Mita Desai, in Proceedings of IEEE Comptuer Vision and Pattern Recognition (CVPR), 2011.

[12] Faisal Imtiaz, (2016, Nov 2), "Pedestrian Walking ,Human Activity Recognition Video ,DataSet By UET Peshawar"
`https://www.youtube.com/watch?v=fWZkkDYTLSI`

[13] Khizr Khan, Shuhan Zheng, "Outputted log files of video characteristics" https://drive.google.com/drive/folders/1yoqSVX5UM9hBRiZsc2b3-KGdK-zWB0q?$usp = sharing$

[14] Redmon, Joseph, and Ali Farhadi. "YOLOv3: An Incremental Improvement", Arxiv,
`arxiv.org/pdf/1804.02767.pdf`

[15] Ren, Shaoqing, et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, Arxiv,
`arxiv.org/pdf/1506.01497.pdf`

[16] Gandhi, Rohith. "R-CNN, Fast R-CNN, Faster R-CNN, YOLO - Object Detection Algorithms." Medium, Towards Data Science, 9 July 2018,
`towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365`

[17] Facebookresearch. "Facebookresearch/detectron2." GitHub,
`github.com/facebookresearch/detectron2`

[18] Pjreddie. "Pjreddie/Darknet." GitHub,
`github.com/pjreddie/darknet`

[19] Danielgatis. "Danielgatis/Darknetpy." GitHub,
`github.com/danielgatis/darknetpy`

[20] Rezanejad Murteza, "Intro to Deep Learning", CSC420 Intro to Image Understanding Lecture Slides

[21] K, Sambasivarao. "Region of Interest Pooling." Medium, Towards Data Science, 30 Sept. 2019,
`towardsdatascience.com/region-of-interest-pooling-f7c637f409af`