2020

Social distance monitoring

INFO634-002- DATAMINING FINAL PROJECT

- LIKHIL KUMAR RACHURI(LKR46)
- MADHU BANDRU(MB4236)
- VUTHEJ KRISHNA REDDY(VV334)

Abstract

Social distancing has become a part of our lives more than ever during the recent times. It merely is to maintain a physical distance to help in preventing the spread of a contagious disease.

From our project, we would like to analyze a video which displays people walking around and provide the output in the same frame whether or not the people shown in the video are maintaining a social distance between them. The code will then be generalized in order to take any sample video as input and to provide the output. This will potentially help authorities to monitor public areas for practicing social distance, which will eventually help to curb the spread of a contagious disease.

Introduction

Currently, the world is facing its biggest pandemic COVID-19. COVID-19 (Corona Virus disease 2019) is defined as an illness caused by a novel coronavirus, now called severe acute respiratory syndrome coronavirus 2. Coronavirus was first reported on December 31, 2019. WHO declared this outbreak as a global emergency on January 30, 2020, and then global pandemic on March 11, 2020.

Many countries declared lockdown to protect their citizens and stop spreading the virus. As the nation's economy is falling, many countries are forced to lift lockdown and are re-building their economy to survive. Now, governments across the globe have new challenges to monitoring the people for maintaining social distancing to stop spreading of the virus, which is a tedious one.

Why social distance important

Social distancing is also called physical distancing, which means keeping space between ourselves and others in public areas. COVID-19 mainly spreads when one comes in close contact with an infected person for a prolonged period. Generally, it spreads when an infected person coughs, sneeze, or droplets from their mouth or nose fall on others. The essential recommendation for maintaining social distance is 6 feet between two people.

Now that governments lifted lockdown in a few nations and people are allowed to work as usual with some instructions like maintaining social distancing, Hand sanitizing. But governments are still concerned about the spread of this disease. It's hard to track whether or not social distancing is being maintained at all public places. Hence, developing an algorithm that detects the violations in social distancing will greatly help authorities in preserving the spread of the virus.

Tools and Packages

Below are tools and packages used for our project. Complete code is developed in python language.

- Language – Python
- Integrated Development Environment – juypter Notebook/Lab
- Packages –
    - OpenCV
    - Numpy
    - TensorFlow
    - Imutils
    - Object_detection

Methodologies

We have implemented this project using two different algorithms.

1. HOG – Histogram of Oriented Gradients
2. DNN – Deep Convolution Neural Network

HOG:

Histogram of Oriented Gradients is a feature descriptor used in computer vision and image processing to detect objects.

Algorithm implementation includes following steps:

a. Gradient computation
b. Orientation binning
c. Descriptor blocks
d. Block normalization
e. Object recognition

This technique counts occurrences of gradient orientation in a localized proportion of the image. HOG descriptor identifies whether or not each pixel is an edge. It extracts gradient and orientation by calculating both direction and magnitude by breaking down the image into smaller regions. Histogram with 9 bins will then be created by categorizing the magnitude or the gradient into 9 different directions from 0 to 180.

Another critical point to note is that the descriptor scales down the image to 1:2 ratio (width to height). For the ease of calculations, the size of 64x128 is preferred in almost all of the cases since the image will be broken down into blocks or cells of 8x8 to extract the features.

These small changes in the x and y directions are then calculated to form the gradients. This process will be iterated across all pixels of the image. This can be simply thought of the change in the intensity across x and y directions for each individual pixel.

By taking the square root of the summation of the square of gradient values across x and y axes, the magnitude will be obtained. Direction for this magnitude is obtained by an inverse tangent.

As mentioned earlier, each 8x8 cell is then represented as a histogram, which is divided into 9 bins, which covers the angle from 0 to 180 degrees. These bins are equally distanced by a difference of 10 degrees. The magnitude of the 8x8 cells for all pixels in the image conveys the length that needs to be considered for each bin in the histogram.

In order to partially remove the noise or the unwanted dense areas with high intensities, we will smooth or normalize the data. This brings us to the topic of block normalization. This normalizing of variation in the light is obtained by considering the 16x16 cells.

By combining all the features of 16x16 cells that we have obtained so far, features of the final image will be obtained. Thus the HOG feature descriptor uses the intensity or rather changes in the intensity as the main criteria in finding out the features of the image.

DNN:

For our project, we have considered the model as ssd_mobilenet_v1_coco_2017_11_17. This is a pre-trained model on the novel data set from Common Objects in Context (COCO). This single convolution network predicts bounding box locations by classifying these locations in just one pass. This architecture makes single-shot detection (SSD) quick and efficient. By using the feature maps, SSD identifies the coordinates of bounding boxes.

The exact shape of the object which is to be detected is not predicted by the SSD model, but rather the outline of where the box is located will be displayed.

Mobilenet architecture initially filters the input frame of the video in the depth-wise convolution layer, and then it combines all these filtered outputs from the depth-wise convolution layer to generate new features for object detection.

Input to this model should be a tensor created from the image. The output of this ssd_mobilenet_v1_coco_2017_11_17 model always gives us a dictionary object with four items namely

- Detection_scores with Float Datatype and with a size of 100
- Detection_classes with Float Datatype and with a size of 100
- Num_detections talks about the number of objects that have been identified.
- Detection_boxes if shape (, 100, 4) which specifies the location of the bounding box.

Detection_scores is the score given by the model for an object that it has identified. It can range upto 100 as percentage which implies the accuracy or confidence.

Detection_classes is the label provided by the model for the object that it has identified. These labels will be used in training of the model.

Num_detections will provide the exact number of objects identified by our model in one frame.

Detection_boxes provides us the location of the bounding box for every object in the frame. The results from detection_boxes are in normalized form ranging from 0 to 1.

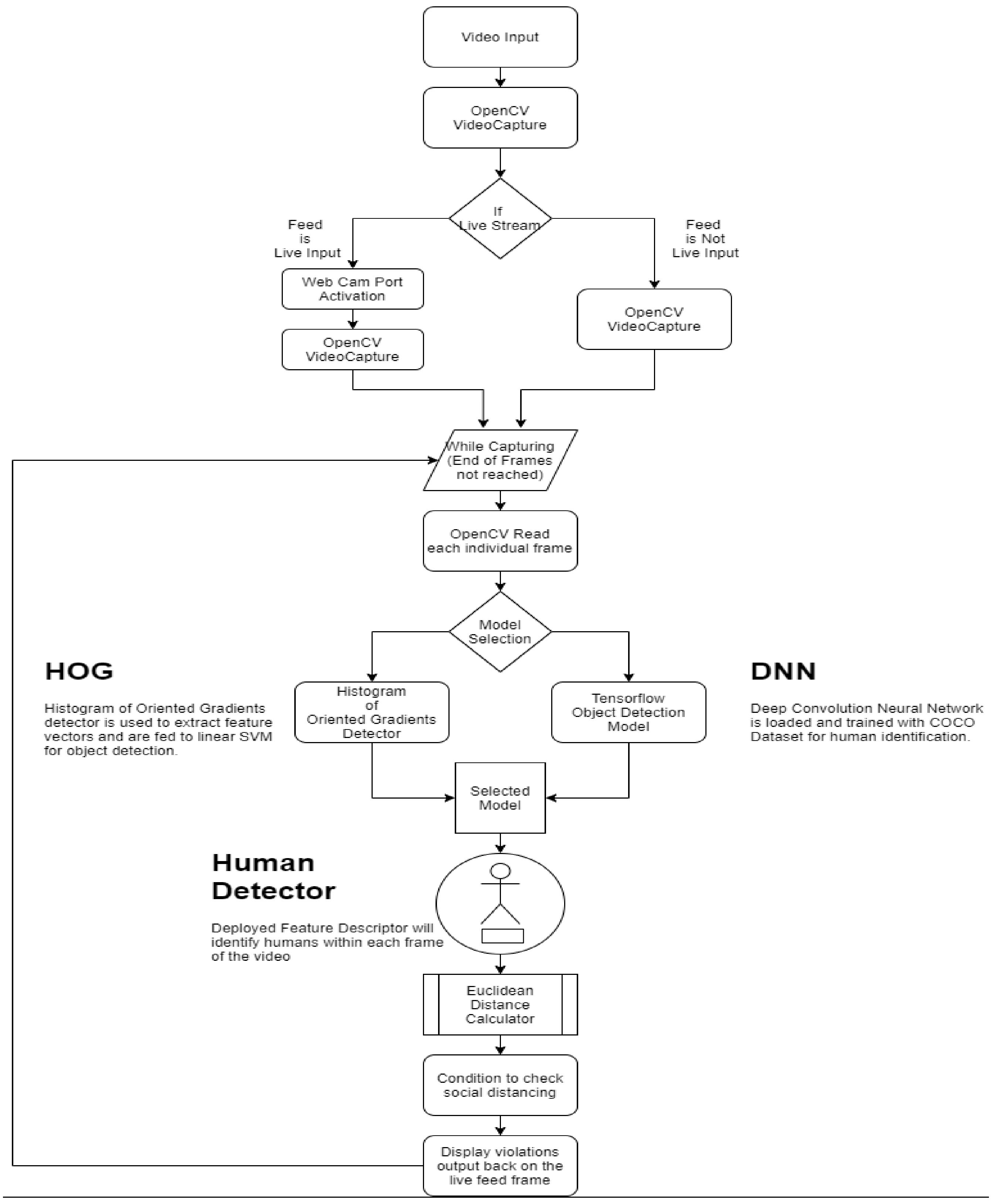
Working

The input video is processed by iterating over the frames. Each frame is fed into the SSD Object detection model. Firstly, the frame is converted into a NumPy array and then converted to a tensor. A new dimension needs to be added to this frame as the model expects more than one image. Hence a new empty dimension is added.

When this frame is passed to the model, we will receive a dictionary object with four items, as mentioned above. We will then be filtering out the objects only with the class of value ‘1’, which implies a “Person.” A threshold needs to be set based on the detection_scores to further filter out the boxes. In our case, this is set to a default of 0.35.

These filtered out boxes are in normalized form and needs to be scaled to the frame size by multiplying with frame height and width. This information of the location of bounding boxes is then fed to OpenCV to plot the rectangles to further process the flow.

Design/Flow


Video Input
OpenCV VideoCapture
If Live Stream
Feed is Live Input
Feed is Not Live Input
Web Cam Port Activation
OpenCV VideoCapture
OpenCV VideoCapture
While Capturing (End of Frames not reached)
OpenCV Read each individual frame
Model Selection
Histogram of Oriented Gradients Detector
Tensorflow Object Detection Model
Selected Model
Euclidean Distance Calculator
Condition to check social distancing
Display violations output back on the live feed frame

HOG

Histogram of Oriented Gradients detector is used to extract feature vectors and are fed to linear SVM for object detection.

DNN

Deep Convolution Neural Network is loaded and trained with COCO Dataset for human identification.

Human Detector

Deployed Feature Descriptor will identify humans within each frame of the video

Input video provided to our algorithm is captured by using the VideoCapture module in OpenCV. The system will check whether the provided input video is a live feed of data or rather a previously recorded data. If the feed that we need to process is a live video, then we need to initialize the webcam port before we start to capture the video. Otherwise, if the feed is provided as input from a specific path, we don't need to initialize the webcam port.

The captured video will then be divided into frames for further analysis. Each frame will be read by using the OpenCV.read() method before providing it to the feature descriptors.

Based on the user input, analysis of Individual frames will be carried out by either Histogram of Oriented Descriptor or the TensorFlow's Deep Convolution Neural Network descriptor to identify the humans in the frame.

Each individual human will then be outlined by a rectangular box before proceeding to calculate the Euclidean distance amongst all the individuals in the provided frame.

The centroid of each rectangular box outlining each individual is considered for the distance calculation. Pairwise Euclidean distance is then computed and checked for a threshold of approximately 6 feet, which is the height of the rectangular box that we have.

If a person is found violating this distance with any one individual, he will then be outlined by a red rectangular box in the output video. However, if a person is maintaining a reasonable distance from others throughout the video, a green rectangular box will be displayed outlining that person in the output video.

This process will be followed and repeated over all the frames within the video and will simultaneously be displayed with green or red rectangular boxes conveying the message of social distance practice.

At any given instance, we are also calculating the number of violators by analyzing each frame, and the same is being displayed in the output. Along with this, we are also providing the percentage of people who are abiding by this social distancing practice in the output video.

Once the processing is done over entire frames of the video, we will be saving the output video, which can be played anytime in the future to evaluate how well social distancing is being practiced in that particular location.

Testing

Testing is one of the most important section of the project. We have performed below test cases using already recorded videos and also using live feed captured through system camera. We have illustrated results below.

1. HOG – Input a recorded video
2. DNN – Input a recorded video
3. HOG – Input live feed
4. DNN – Input live feed

Test case with HOG

Test results for recorded video or live feed using the HOG algorithm will be of similar type. In HOG, pixels with high intensity are highlighted and detected as objects. In certain instances, even empty places with high light intensity were treated as humans, and a rectangular box was positioned, which is incorrect.

Using the centroid point of the objects, we are calculating Euclidean distance. If the Euclidean distance is less than the threshold value, the distance between those two objects or humans is less than 6 feet, and social distancing is being violated. For those objects which are violating social distance are highlighted with red rectangular box and centroid is also turned to red. If the distance is greater than the threshold, then those objects are practicing social distancing and are highlighted with a green rectangular box and with green centroids.

If an object with another object is detected as violating social distance, then both objects are highlighted with a red rectangular box and red centroids. Then both objects are not verified again for violation in that frame. If an object is not violating with another object, then it is marked as green and then further verified for violation with other persons.

The same process is repeated for every frame, and results are displayed simultaneously.

We have also calculated the number of persons & percentage of violating social distance for each frame and displayed the same. As said above in HOG, objects are detected based on the intensity of pixels.
So those kinds of objects are also considered when evaluating and calculating violation.

Test case with DNN:

Test results for recorded video or live feed using the DNN algorithm will be of similar type. Highlighting a violated and non-violated object using DNN will be the same as the HOG algorithm. The major difference comes while detecting the object.

In the Deep Convolution Neural Network algorithm, all objects are detected rather than just detecting humans. Unlike in the HOG descriptor, this algorithm accurately detects humans. But if in the frame, if two or more objects are together or very near, then all the objects are together considered as a single object, and a rectangular box is highlighted over the whole group rather than one individual. But this is rarely observed as most of the time, and the algorithm proved to be accurate.

Here we have also calculated the number of persons & percentage of violating social distance for each frame and displayed the same.

We were unable to identify metrics to evaluate the performance of both these object detectors without actually avoiding human intervention. Had there been already a dataset with all possible humans in all the frames of the video, models could have been evaluated efficiently by using the labels. As this wasn't the case, human intervention was unavoidable when it comes to evaluating the models for accuracy on our code.

Results

Below are outputs displayed when we execute the both models.

1. Highlight objects with Red color if violate social distance.
2. Highlight objects with Green color if no social distance violation.
3. Highlight objects center point with Red color if violate social distance.
4. Highlight objects center point with Green color if no social distance violation.
5. Display number of violations means number of objects violate social distance in frame.
6. Display percentage of violation in frame.
7. Save processed video of HOG and DNN algorithm for given video input.
8. Save processed video of live feed.

Challenges Faced

The identification of empty spaces in the frames of the video was sometimes understood as humans due to the difference in intensity over the edges by the HOG feature descriptor.

Another challenge we've faced is with the DNN model. If multiple objects were to be in close proximity, the model is identifying the complete group of closely displayed objects (in our case, humans) as one single object. This leads to inconsistencies.

To eliminate these issues, models need to be further trained with data sets that contain these anomalies.

Future Work

As a future work, we can make below improvements –

1. Improve HOG algorithm to detect only objects.
2. Improve HOG algorithm to avoid detecting non-object pixels based on intensity.
3. Improve DNN algorithm to differentiate between objects which are very close.
4. Add new feature to detect the violation from bird eye view.
5. Add alerting system when percentage of violation is reached maximum threshold.

References

[1] https://docs.opencv.org/2.4/
[2] https://www.tensorflow.org
[3] https://www.pyimagesearch.com/2015/11/16/hog-detectmultiscale-parameters-explained/
*[4]*https://www.programcreek.com/python/example/84776/cv2.HOGDescriptor

[5] https://www.analyticsvidhya.com/blog/2020/04/build-your-own-object-detection-model-using-tensorflow-api/