DATA ANALYTIC - DISCOVERY SCIENCE

Counting of crowd size with the help of Camera

Mentor: Dr. Mayank Vatsa & Dr. Richa Singh

- Problem Statement: Counting the crowd size with the help of a camera using computer vision techniques.

- Overview

    1. Crowd counting via cameras is important in this current situation of COVID 19 pandemic when surveillance is not an easy task. In this project, based on smartphone cameras or drone cameras, counting how many people are in the given video is an important task. Using computer vision techniques, in this project, for a given video, the requirement is to count the people present.
    2. Using the computer vision and deep learning approach this project can count the number of people in the crowd and can also detect the violation of social distancing.

- Application of this project

    1. When people in large numbers gather in a place it increases the risk of spread on the contagious disease (Coronavirus) therefore it is important to count the number of people to stop Mass Spread of coronavirus. Along with counting the number, it is also important to monitor the social distancing among the crowd.
    2. Social Distancing implies that people should physically distance themselves from one another, reducing close contact, and thereby reducing the spread of contagious diseases, such as coronavirus.
    3. Social Distancing involves a minimum of 6 feet (2m) distance between two persons.

- Object Detection and Object Tracking

    1. Object detection implies to determine wherein an image/frame an object is. An object detector is more computationally expensive and therefore slower than an object tracking algorithm.
    2. Models that are used for object detection include:

        a. Haar Cascades
        b. HOG + Linear SVM
        c. Some deep learning-based object detectors such as:

i. Faster R-CNNs
ii. YOLO (You Only Look/Live Once)
iii. SSD (Single Shot Detectors)

3. Object tracking algorithms include:

    a. Kernelized Correlation Filters

    b. Median Flow

    c. MOSSE

- Technology Used in This Project

1. This project includes the use of OpenCV, Python and deep learning to apply YOLO to the video stream.

2. Python: It's an interpreted, high-level, general purpose programming language.

3. OpenCV (Open Source Computer Vision Library): It is an open source computer vision and machine learning software library. OpenCV provide common infrastructure for computer vision application. It works on various programming languages python, java, C++.

4. YOLO (You Only Look Once): It is a network that uses Deep Learning algorithms for object detection.

5. Counting the number of people is done by counting the detected bounding boxes of people.

6. The social distancing detector, it is done by mapping the pixel distance.

- YOLO (You Look Only Once) and R-CNN (Used for people Detection and Counting number of people)

1. There are three types of DL object detectors:
    a. R-CNN, Fast R-CNN and Faster R-CNN.
    b. Single Shot Detector (SSD)
    c. YOLO

2. R-CNN uses two stage detector strategy whereas SSD and YOLO uses one stage detector strategy

3. YOLO performs detection by classifying certain objects within the image/video determining where they are located.

4. R-CNN is an object detector that removes the Selective Search Requirement and instead rely on Region Proposal Network (RPN).
    a. RPN is fully convolutional and can predict the object bounding boxes and "Objectness" scores i.e. a score quantifying how likely it is a region of an image may contain an image.
    b. The outputs of RPNs are then passed int R-CNN component for final classification and labelling.

5. While R-CNNs tend to very accurate but the R-CNN family of networks are incredibly slow at their speed, obtaining only 5 FPS on a GPU.

6. To increase the speed of Deep Learning based object detectors, both Single Shot Detector (SSD) and YOLO were introduced.

    a. These algorithms treat an object detection as a regression problem, taking a given input image and simultaneously learning bounding box coordinated and corresponding class label probabilities.

7. YOLO object detector is capable of super real time object detection, obtaining 45 FPS on a GPU.

8. In this project YOLOv3 is used and YOLO is trained on COCO dataset.
    a. COCO dataset is consisting of around 80 labels.
    b. The labels include People, Bicycles, Cars and trucks, Airplane, etc.

- Social Distancing Detector

1. Working of social distancing detector:
    a. Apply object detection to detect all people (and only people) in a video stream.
    b. Compute the pairwise distances between all detected people using the pixel distances.
    c. Based on these distances, check to see if any two people are less than N pixels apart.

2. For the most accurate results the camera is calibrated through intrinsic/extrinsic parameters so that the pixels can be mapped to measurable units.

3. For the sake of simplicity, project's OpenCV social distancing detector implementation rely on pixel distances.

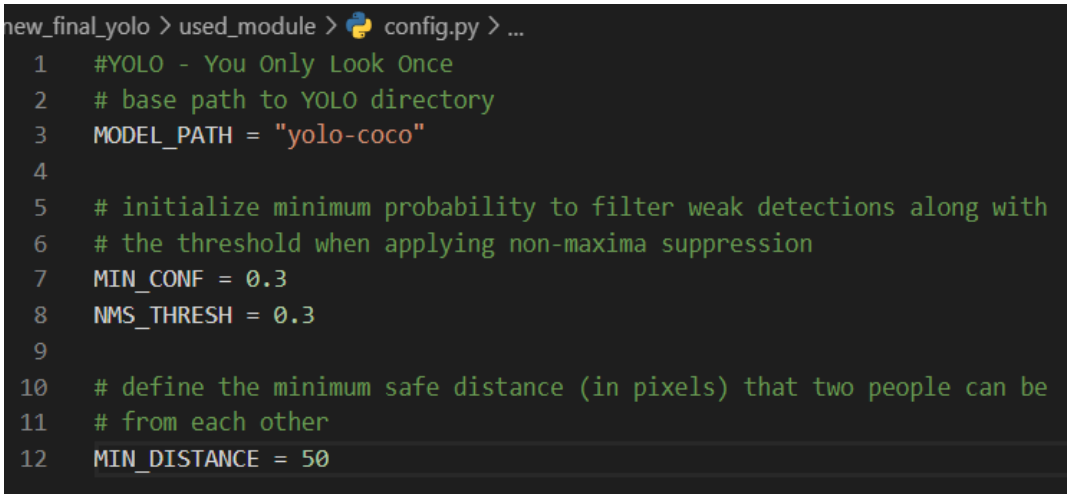
4. The main python driver script, glues all the pieces together into full-fledged OpenCV social distancing detector.

- Project Structure

1. This project is consisting of three python scripts, two working scripts and one driver script to run the program:
    a. Config.py
    b. Detection.py
    c. Driver_main.py

2. YOLO-COCO includes the CNN architecture, pre-defined weights and class names. This YOLO model is compatible with OpenCV's DNN module.

3. Config.py (Configuration file):



a. This configuration file is to store important variables.

MIN_CONF is the minimum probability to filter weak detections

MIN_THRESH is the non-maxima suppression value

MIN_DISTANCE is the minimum distance (in pixels) for social distancing

b. Non-Maxima Suppression (NMS): It is a technique used in many computer vision algorithms. It is a class of algorithms to select one entity (e.g. bounding boxes) out of many overlapping entities.

4. Detection.py script:

a. This detection python file involves the code for YOLO object detection with OpenCV.

b. The object detection logic is put in a function in this script for convenience. Doing so Free up driver script's frame processing loop from becoming cluttered.

c. Line 1-4: All packages and libraries are imported from python library and from config file.

new_final_yolo > used_module > detection.py > detect_people

```python
from .config import NMS_THRESH
from .config import MIN_CONF
import numpy as np
import cv2

def detect_people(frame, net, ln, personIdx=0):
    (H, W) = frame.shape[:2]
    results = []

    # construct a blob from the input frame and then perform a forward
    # pass of the YOLO object detector, giving bounding boxes and associated probabilities
    blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),
        swapRB=True, crop=False)
    net.setInput(blob)
    layerOutputs = net.forward(ln)

    # initializing lists of detected bounding boxes, centroids, and confidences
    boxes = []
    centroids = []
    confidences = []
```

d. detect_people function will filter out only class “people” from the detections.

e. frame: frame from the video input or live streaming
f. net: the pre-trained YOLO object detection model
g. ln: the YOLO CNN output layer names
h. personIdx: this index is specifically for the person class
i. results list is initialized that consist of person prediction probability, bounding box coordinates for the detection and the centroid of the object.

```
    # loop over each of the layer outputs
    for output in layerOutputs:
        # loop over each of the detections
        for detection in output:
            # extract the class ID and confidence (i.e., probability)
            # of the current object detection
            scores = detection[5:]
            classID = np.argmax(scores)
            confidence = scores[classID]

            # filter detections by (1) ensuring that the object
            # detected was a person and (2) that the minimum confidence is met
            if classID == personIdx and confidence > MIN_CONF:

                box = detection[0:4] * np.array([W, H, W, H])
                (centerX, centerY, width, height) = box.astype("int")

                # use the center (x, y)-coordinates to derive the top
                # and and left corner of the bounding box
                x = int(centerX - (width / 2))
                y = int(centerY - (height / 2))

                # update our list of bounding box coordinates,
                # centroids, and confidences
                boxes.append([x, y, int(width), int(height)])
                centroids.append((centerX, centerY))
                confidences.append(float(confidence))

    # apply non-maxima suppression to suppress weak, overlapping
    # bounding boxes
    idxs = cv2.dnn.NMSBoxes(boxes, confidences, MIN_CONF, NMS_THRESH)
```

j. Line 29-34: looping over the detections to extract out the classsID and confidence and filter the detection by:
   - Ensuring object detected was a person
   - Minimum confidence i.e. probability is met

k. Line 38-52: Scaling the bounding box coordinates back relative to the image, YOLO returns the centre (x, y) coordinates of the bounding box followed by its width and height.

```
58      # ensure at least one detection exists
59      if len(idxs) > 0:
60          # loop over the indexes we are keeping
61          for i in idxs.flatten():
62              # extract the bounding box coordinates
63              (x, y) = (boxes[i][0], boxes[i][1])
64              (w, h) = (boxes[i][2], boxes[i][3])
65
66              # update our results list to consist of the person
67              # prediction probability, bounding box coordinates,
68              # and the centroid
69              r = (confidences[i], (x, y, x + w, y + h), centroids[i])
70              results.append(r)
71
72      # return the list of results
73      return results
```

l. Line 58-73: ensure at least one detection exist and update the results list and return the results list.

5. Driver_main.py script:

a. This main program driver file is responsible for looping over frames of a video stream, counting the number of people present in the frame and ensuring that people are maintaining a healthy distance.

```
1  from used_module import config as config
2  from used_module.detection import detect_people
3  from scipy.spatial import distance as dist
4  import numpy as np
5  import argparse
6  import imutils
7  import cv2
8  import os
9  import datetime
```

b. It is compatible with both input video files and webcam live streams.

c. Line 1-9: importing the packages, libraries including the “config”, “detect_people” function and Euclidean distance (shortened to dist) to determine the distance between the centroids.

d.

```
11  # construct the argument parse and parse the arguments
12  ap = argparse.ArgumentParser()
13  ap.add_argument("-i", "--input", type=str, default="",
14      help="path to (optional) input video file")
15  ap.add_argument("-o", "--output", type=str, default="",
16      help="path to (optional) output video file")
17  ap.add_argument("-d", "--display", type=int, default=1,
18      help="whether or not output frame should be displayed")
19  args = vars(ap.parse_args())
20
21  # load the COCO class labels our YOLO model was trained on
22  labelsPath = os.path.sep.join([config.MODEL_PATH, "coco.names"])
23  LABELS = open(labelsPath).read().strip().split("\n")
24
25  # derive the paths to the YOLO weights and model configuration
26  weightsPath = os.path.sep.join([config.MODEL_PATH, "yolov3.weights"])
27  configPath = os.path.sep.join([config.MODEL_PATH, "yolov3.cfg"])
28
29  # load our YOLO object detector trained on COCO dataset (80 classes)
30  print("[INFO] loading YOLO from disk...")
31  net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)
32
33  # determine only the *output* layer names that we need from YOLO
34  ln = net.getLayerNames()
35  ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]
36
37  # initialize the video stream and pointer to output video file
38  print("[INFO] accessing video stream...")
39  vs = cv2.VideoCapture(args["input"] if args["input"] else 0)
40  writer = None
```

e. Line 12-19: Constructed the argument parse to take the commands from the command line prompt/terminal.

f. Line 21-31: loading the YOLO-COCO models and using the YOLO paths now the model can be load into memory.

g. Line 34-40: getting the layer names in order to process results and initialize video stream and pointer for the output video file.

```
44  # loop over the frames from the video stream
45  while True:
46      # read the next frame from the file
47      (grabbed, frame) = vs.read()
48
49      # if the frame was not grabbed, then it is reached the end of stream
50      if not grabbed:
51          break
52
53      # resize the frame and then detect people (and only people) in it
54      frame = imutils.resize(frame, width=700)
55      results = detect_people(frame, net, ln,
56          personIdx=LABELS.index("person"))
57
58      violate = set()
```

h. Line 55-58: Using the function "detect_people" the results are grabbed for the YOLO object detection.

Violate set is initialized that maintains list of people who violate social distancing.

```
62      if len(results) >= 2:
63
64          centroids = np.array([r[2] for r in results])
65          D = dist.cdist(centroids, centroids, metric="euclidean")
66
67          # loop over the upper triangular of the distance matrix
68          for i in range(0, D.shape[0]):
69              for j in range(i + 1, D.shape[1]):
70                  # checking if distance between any two centroid pairs
71                  # is less than the configured number of pixels
72                  if D[i, j] < config.MIN_DISTANCE:
73
74                      violate.add(i)
75                      violate.add(j)
```

i. This plot is to ensure there are at least 2 people detection in order to compute pairwise distance maps.

j. Line 64-65: Extracting all centroids from the results and compute Euclidean distance between all pairs of centroids.

k. Line 67-75: looping over the upper triangular of the distance matrix checks if the distance between any two centroids is less than the configured number of pixels.

```
79      # loop over the results
80      for (i, (prob, bbox, centroid)) in enumerate(results):
81          # extract the bounding box and centroid coordinates, then
82          # initialize the color of the annotation
83          (startX, startY, endX, endY) = bbox
84          (cX, cY) = centroid
85          color = (0, 255, 0)
86
87          if i in violate:
88              color = (0, 0, 255)
89          # draw (1) a bounding box around the person and (2) the
90          # centroid coordinates of the person,
91          cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
92          cv2.circle(frame, (cX, cY), 5, color, 1)
```

l. Line 83-85: Extract the bounding box and centroid coordinates and initialize color of box to green.

m.Line 87: if the index pair is in the violate set then color is updated to red.

n. Red color bounding shows that social distancing not found with [WARNING] notation.

o. No. of violations: no. of people doing social distancing

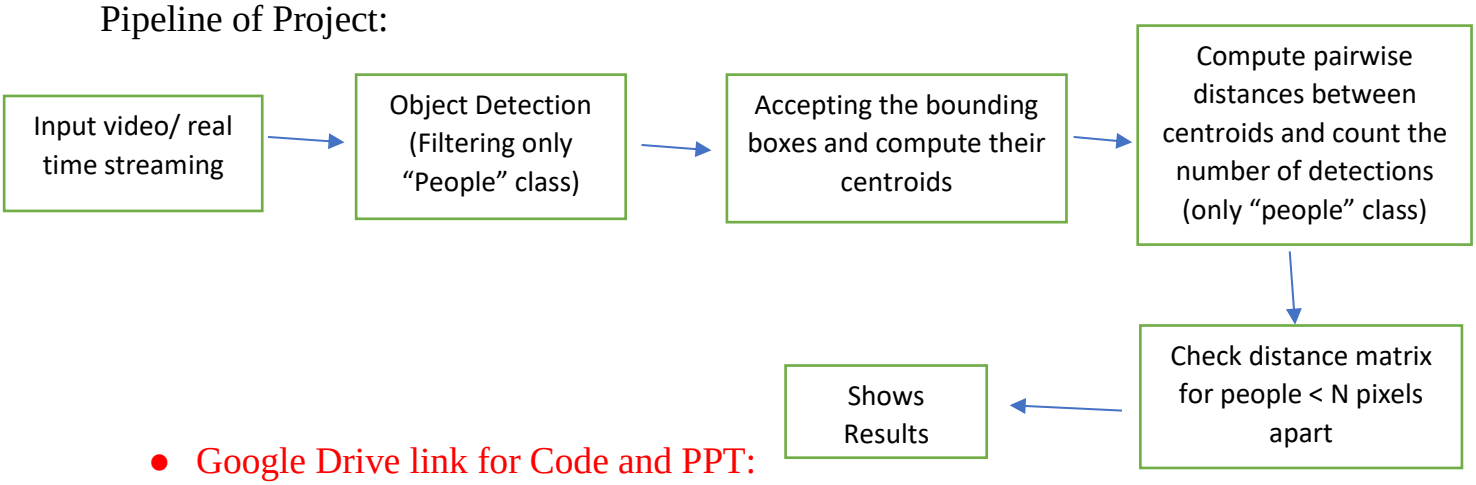
```
125
126     currentDT = datetime.datetime.now()
127
128     print("[INFO] People present in frame: {}".format(len(results)))
129
130     if len(violate)!=0:
131         print("[WARNING] Social Distancing Not Found")
132
133     print(currentDT.strftime("Date : %Y-%m-%d Time: %H:%M:%S"))
```

p. No. of People in frame: no. of people seen at present in the frame

```
C:\Users\hp\Documents\Programming\Project\Su
C:\Users\hp\Documents\Programming\Project\Su
[INFO] loading YOLO from disk...
[INFO] accessing video stream...
[INFO] People present in frame: 13
[WARNING] Social Distancing Not Found
Date : 2020-08-19 Time: 23:40:15
[INFO] People present in frame: 13
[WARNING] Social Distancing Not Found
Date : 2020-08-19 Time: 23:40:16
```

q. Line 126-133: Prints number of people in frame, date and time. Also, prints warning if social distancing not found int the command line terminal.

Pipeline of Project:



- Google Drive link for Code and PPT:

https://drive.google.com/file/d/1MoY2PmMvMzslF7pp4LRkhSEL6haHvzeV/view?usp=sharing

- Reference:
  1. https://www.youtube.com/watch?v=kdLM6AOd2vc&list=PLS1QulWo1RIa7D1O6skqDQ-JZ1GGHKK-K
  2. https://www.pyimagesearch.com/2018/08/13/opencv-people-counter/
  3. https://www.pyimagesearch.com/2020/06/01/opencv-social-distancing-detector/
  4. https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/
- Hardware Used:
  1. Laptop – HP Envy X360
  2. Ram – 8GB
  3. Processor – AMD Ryzen 5
  4. GPU – Radeon Vega Gfx 2.10 GHz
  5. System Type – 64 bit OS

--------Thank You--------

Project by – Vaibhav Meena (B19BB049)

Bioscience and Bioengineering Dept.