**A**

**SYNOPSIS REPORT**

**ON**

**“AI Eye Prognosticator”**

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Engineering in “**Information Technology Semester - VIII**”

By:

**Names ID No’s.**

Karthik Pillai VU4F1718038

Satyam Yadav VU4F1718040

Mohit Saini VU4F1718045

Under the Guide of: -

Prof. Supriya Chaudhary



**Department of Information Technology**

**Padmabhushan Vasantdada Patil Pratisthan’s College of Engineering**

VasantdadaPatil Educational Complex, Eastern Express Highway, Near Everard Nagar, Sion, Chunabhatti , Mumbai – 400022

**UNIVERSITY OF MUMBAI**

**June, 2021**

**CERTIFICATE**

This is to certify that the project entitled **“AI Eye Prognosticator”** is a bonafide work of **“**Karthik Pillai (VU4F1718038), Satyam Yadav (VU4F1718040), Mohit Saini (VU4F1718045)**”** submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of **“Bachelor of Engineering”** in **“Information Technology Semester - VIII”**.

|  |
| --- |
| Prof. Supriya Chaudhary |
| Guide Co-Supervisor/Guide |

|  |  |
| --- | --- |
|  |  |
|  |  |
| Dr Seema Lade | Dr. Alam N. Shaikh |

**Project Report Approval for B. E.**

This project report entitled “**AI Eye Prognosticator**” by Karthik Pillai, Satyam Yadav, Mohit Saini is approved for the degree of **“Bachelor of Engineering”** in**“Information Technology Semester - VIII”**.

Examiners

1.---------------------------------------------

2.---------------------------------------------

Date:

Place: Mumbai

**ABSTRACT**

Aim of the project is to implement a system which is capable of extracting information from the CCTV footage by using human detection algorithms and human recognition algorithms to derive valuable insights from the footage. The system uses YOLOv3 human detection algorithm (CNN Based Algorithm) to detect humans. This data is further used by Machine Learning algorithms to make predictions regarding the human population density at a particular location, human count and human recognition.

Submitted by:

**Names ID No’s.**

Karthik Pillai VU4F1718038

Satyam Yadav VU4F1718040

Mohit Saini VU4F1718045

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
|  |  | **Page No** |
| **1** | **Introduction** | 5 |
|  | 1.1 Aim of the Project | 5 |
| **2** | **Literature Survey** |  |
|  | 2.1 Summary | 6 |
|  | 2.2 Comparative study | 9 |
| **3** | **Proposed System** |  |
|  | 3.1 Proposed System | 10 |
|  | 3.1.1 Features of Proposed System | 10 |
|  | 3.1.2 Applications of Proposed System | 10 |
|  | 3.2 Requirement Analysis | 11 |
|  | 3.2.1 Hardware Specification | 11 |
|  | 3.2.2 Software Specification | 11 |
| **4** | **Design** |  |
|  | 4.1 Algorithm | 12 |
|  | 4.2 Flowchart | 15 |
|  | 4.3 Yolo Working Diagram | 16 |
|  | 4.4 CNN | 16 |
| **5** | **Implementation** |  |
|  | 5.1 Datasets | 18 |
|  | 5.2 Coding | 18 |
| **6** | **Conclusion** | 33 |
| **7** | **References** | 34 |

1. **Introduction**

In AI Eye Prognosticator we develop a system which uses computer vision & machine learning algorithms to detect humans & different objects. This data is then used to get useful insights from the humans and objects data. For different objects vehicle detection is also implemented. This data is further used to make predictions like space utilization, people density at a point of time, number of people incoming and outgoing and number of vehicles incoming and outgoing.

The data stored is also used to analyse how the space is utilized in a supermarket, mall, shop, highway and road.By using different machine learning algorithms these data can be used to predict customer or humans behaviour in crowded places, supermarket, mall and shop.

**1.1 Aim of the project**

Aim of the project is to implement a system which is capable of extracting information from the CCTV footage by using human detection algorithms and human recognition algorithms to derive valuable insights from the footage. The system uses YOLOv3 human detection algorithm (CNN Based Algorithm) to detect humans. This data is further used by Machine Learning algorithms to make predictions regarding the human population density at a particular location, human count and human recognition.

**2. Literature Survey**

**2.1 Summary:**

**Paper 1. Summary:**

Paper 1: - People counting and human detection in a challenging situation

Reliable people counting and human detection is an important problem in visual surveillance. In recent years, the field has seen many advances, but the solutions have restrictions: people must be moving, the background must be simple, and the image resolution must be high. This paper aims to develop an effective method for estimating the number of people and locate each individual in a low resolution image with complicated scenes. The contribution of this paper is threefold. First, post processing steps are performed on background subtraction results to estimate the number of people in a complicated scene, which includes people who are moving only slightly. Second, an Expectation Maximization (EM)-based method has been developed to locate individuals in a low resolution scene. In this method, a new cluster model is used to represent each person in the scene. The method does not require a very accurate foreground contour. Third, the number of people is used as a priori for locating individuals based on feature points. Hence, the methods for estimating the number of people and for locating individuals are connected. The developed methods have been validated based on a 4-hour video, with the number of people in the scene ranging from 36 to 222. The best result for estimating the number of people has an average error of 10% over 51 test cases.

**Paper 2. Summary:**

Paper 2: - Human detection in surveillance videos and its Applications

Detecting human beings accurately in a visual surveillance system is crucial for diverse application areas including abnormal event detection, human gait characterization, congestion analysis, person identification, gender classification and fall detection for elderly people. The first step of the detection process is to detect an object which is in motion. Object detection could be performed using background subtraction, optical flow and spatio-temporal filtering techniques. Once detected, a moving object could be classified as a human being using shape-based, texture-based or motion-based features. A comprehensive review with comparisons on available techniques for detecting human beings in surveillance videos is presented in this paper. The characteristics of few benchmark datasets as well as the future research directions on human detection have also been discussed.

**Paper 3. Summary:**

Paper 3: Enhancing human detection using crowd density measures and and adaptive correction filter

In this paper we improve a human detector by means of crowd density information. Human detection is especially challenging in crowded scenes which makes it important to introduce additional knowledge into the detection process. We compute crowd density maps in order to estimate the spatial distribution of people in the scene and show how it is possible to enhance the detection results of a state-of-the-art human detector using this information. The proposed method applies a self-adaptive, dynamic parametrization and as an additional contribution uses scene-adaptive learning of the human aspect ratio inorder to reduce false positive detections in crowded areas. We evaluate our method on videos from different datasets and demonstrate how our system achieves better results than the baseline algorithm.

**Paper 4. Summary:**

Paper 4: - Real-time embedded person detection and tracking for shopping behaviour analysis

Shopping behaviour analysis through counting and tracking of people in shop-like environments offers valuable information for store operators and provides key insights in the stores layout (e.g. frequently visited spots). Instead of using extra staff for this, automated on-premise solutions are preferred. These automated systems should be cost-effective, preferably on lightweight embedded hardware, work in very challenging situations (e.g. handling occlusions) and preferably work real-time. We solve this challenge by implementing a real-time TensorRT optimized YOLOv3-based pedestrian detector, on a Jetson TX2 hardware platform. By combining the detector with a sparse optical flow tracker we assign a unique ID to each customer and tackle the problem of losing partially occluded customers. Our detector-tracker based solution achieves an average precision of 81.59% at a processing speed of 10 FPS. Besides valuable statistics, heat maps of frequently visited spots are extracted and used as an overlay on the video stream.

**Paper 5. Summary:**

Paper 5: - Research paper on vehicle detection and recognition

According to Wikipedia, a vehicle is any machine that transports people or cargo. Vehicles include cars, bikes, buses, airplanes, space shuttles, cycles and many more. Vehicle detection and vehicle type recognition is a practical application of machine learning concepts and is directly applicable for various operations in a traffic surveillance system contributing to an intelligent traffic surveillance system. This paper will introduce the processing of automatic vehicle detection and recognition using static image datasets. Further using the same technique, we shall improvise vehicle detection by using live CCTV surveillance. The surveillance system includes detection of moving vehicles and recognizing them, counting the number of vehicles and verification of their permit with the organization. Since algorithms play a very important role in any machine learning program, it is important that we choose the best model for our project. Once the vehicle has been detected, LPR shall be implemented, which is, License Plate Recognition. The recognized number plate shall then be processed to capture the license number. This license number will then be compared to an existing database and checked if it is valid, registered with the organization, permit’s validity, if the vehicle is parked at the allotted parking location and many other parameters. The most benefits of this project would be reduced manual efforts in manual checking of each vehicle and also in maintaining manual records of the same.

**2.2 Comparative study of above 5 papers in table:**

|  |  |  |
| --- | --- | --- |
| Name of Paper | Year | Abstract |
| People Counting and Human Detection in a Challenging Situation | 2011 | First, post processing steps are performed on background subtraction results to estimate the number of people in a complicated scene. Then, an Expectation Maximization (EM)-based method has been developed to locate individuals in a low resolution scene. |
| Human detection in surveillance videos and its applications | 2013 | Object detection is performed using background subtraction, optical flow and spatio-temporal filtering techniques. Then a moving object could be classified as a human being using shape-based, texture-based or motion-based features. |
| Enhancing human detection using crowd density measures and an adaptive correction filter | 2016 | The proposed method applies a self-adaptive, dynamic parametrization and as an additional contribution uses scene adaptive learning of the human aspect ratio in order to reduce false positive detections in crowded areas. |
| Real-time embedded person detection and tracking for shopping behaviour analysis | 2020 | Shopping behaviour analysis through counting and tracking of people in shop-like environments offers valuable information for store operators and provides key insights in the stores layout. |
| Research paper on vehicle detection and recognition | 2020 | This paper will introduce the processing of automatic vehicle detection and recognition using static image datasets. |

**3. Proposed System**

**3.1 Proposed System**

The raw data collected by the CCTV cameras is not used to the fullest in the current scenario. AI Eye Prognosticator is a system which is capable of extracting information from the CCTV footage by using human detection algorithms and human recognition algorithms to derive valuable insights from the footage. The proposed system detects humans and counts the number of people. The data of the CCTV cameras is stored in the system and a huge database is created. The system uses YOLOv3 human detection algorithm (CNN Based Algorithm) to detect humans. Reading the patterns and understanding the nature of the objects, this data is further used by Machine Learning algorithms to make predictions regarding the human population density at a particular location, human count and human recognition. The details such as behaviour of the consumer, their likes and dislikes, the pattern of their shopping can be detected. Suggestion could be given to the owners of the malls to engage more customers and enhance their business. Space utilisation in public places can be optimised.

**3.1.1 Features of Proposed System**

* AI Eye Prognosticator detects humans and counts the number of people.
* It also stores the statistics of incoming and outgoing people
* Predict future incoming & outgoing people count
* Track the path of a person
* Predict most populated place
* Optimise space utilisation in public places
* Analyse the density of people count
* In a mall predict which shop has most customers, Which type of shop has more customers, Which part of the mall has most customers
* Provide business insights to the business owners

**3.1.2 Applications of Proposed System**

1. Shopping Malls - To get insights like, population density at a shop.
2. Colleges/Offices – To get a number of people/students entered into college.
3. Retail shops – To get data like which product is least visible to consumers.
4. Streets/Road – To make optimum use of space.

**3.2 Requirement Analysis**

**3.2.1 Hardware Specification**

* Operating System – Windows 7,8, 10 or Professional editions
* Processor - dual core 2.4 GHz+ (i5 or i7 series Intel processor or equivalent AMD)
* RAM – 4 GB
* Hard Drive - 256 GB or larger solid-state hard drive
* Smartphone Android/iOS

**3.2.2 Software Specification**

* Python 2.7.15 +
* PIP 2.7
* Pycharm
* Jupyter Notebook

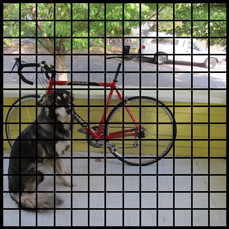
**4. Design**

**4.1 Algorithms**

**YOLO Algorithm**

## YOLO, also referred to as You Only Look Once, is one among the foremost powerful real-time object detection algorithms. It's called that way because unlike previous object detector algorithms, like R-CNN or its upgrade Faster R-CNN, Single Shot MultiBox Detector (SSD), Retina-Net, it only needs the image (or video) to pass just one occasion through its network.

YOLO divides the given image into a grid of n\*n cells (13\*13):



Each of those cells is liable for predicting 5 bounding boxes. A bounding box is nothing but the rectangular box that encloses an object.

YOLO also outputs a confidence score that tells us how certain it's that the anticipated bounding box actually encloses some object. This score doesn’t say anything about what quiet object is within the box, just if the form of the box is any good.

The predicted bounding boxes may look something just like the following (the higher the arrogance score, the fatter the box is drawn):

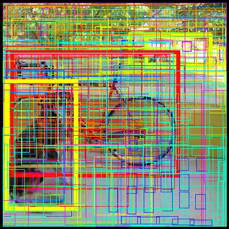


For each bounding box, the cell also predicts a category , it gives a probability distribution over all the possible classes. The version of YOLO we’re using is trained on the COCO dataset. COCO stands for Common Objects in Context which may detect 80 different classes such as: bicycle, boat, car, cat, dog, person then on

COCO is a very large-scale image dataset. It includes annotations for image segmentation, object detection, image labelling and key points. The COCO team itself prepares these segments, labels, key points and lots of more that's why COCO is reliable to use and enables us to make robust models. COCO has several features:

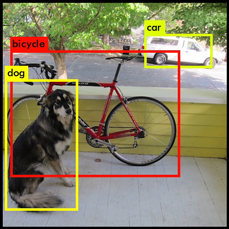
1. Object segmentation
2. Recognition in context
3. Superpixel stuff segmentation
4. 330K images (>200K labeled)
5. 1.5 million object instances
6. 80 object categories
7. 91 stuff categories
8. 5 captions per image
9. 250,000 people with keypoints

The confidence score for the bounding box and therefore the class prediction are combined into one final score that tells us the probability that this bounding box contains a selected sort of object. for instance , the large fat yellow box on the left is 85% sure it contains the thing “dog”:



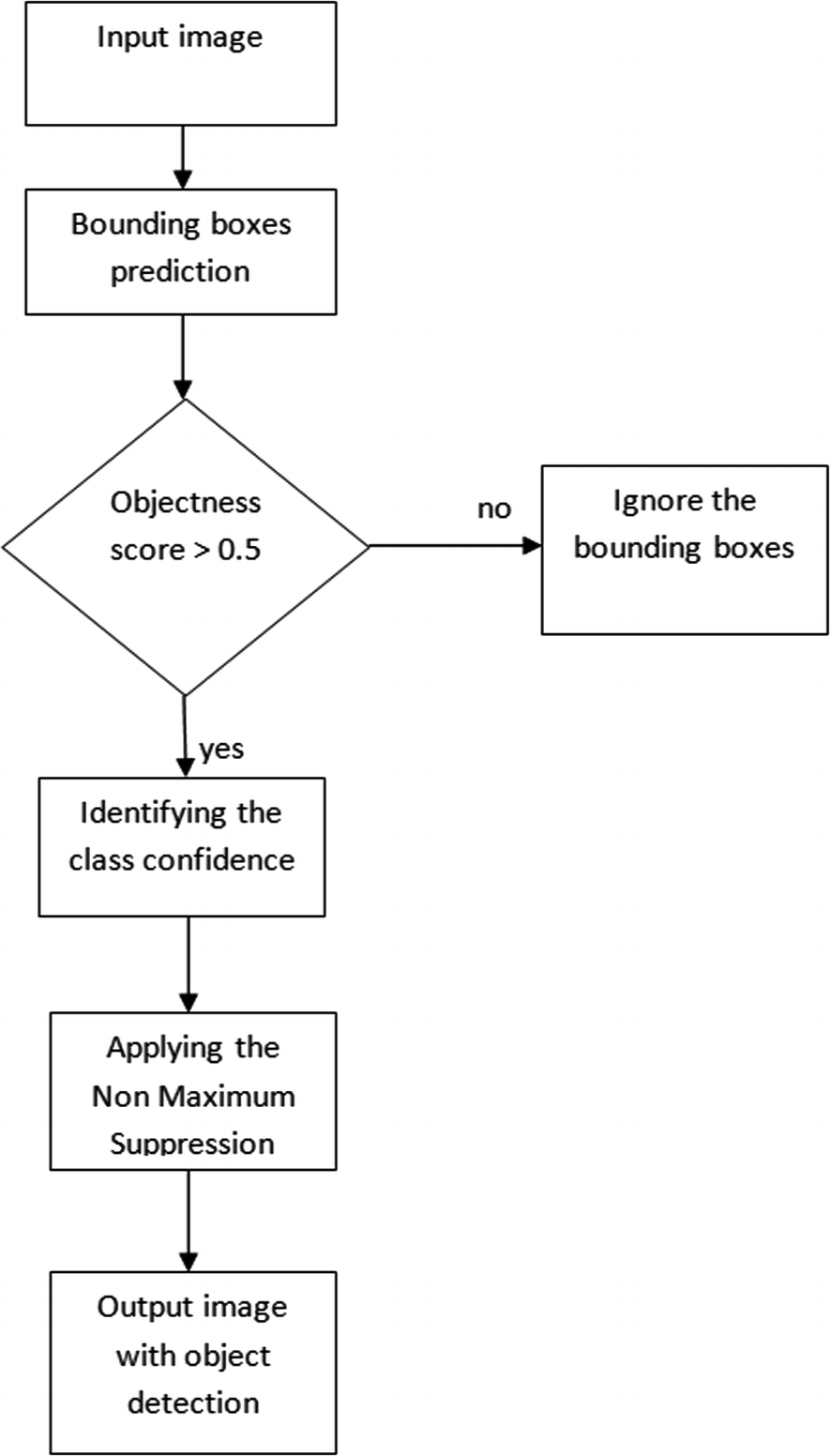
Since there are 13×13 = 169 grid cells and every cell predicts 5 bounding boxes, we find yourself with 845 bounding boxes in total. It seems that the majority of those boxes will have very low confidence scores, so we only keep the boxes whose final score is 30% or more (you can change this threshold counting on how accurate you would like the detector to be).

The final prediction is then:

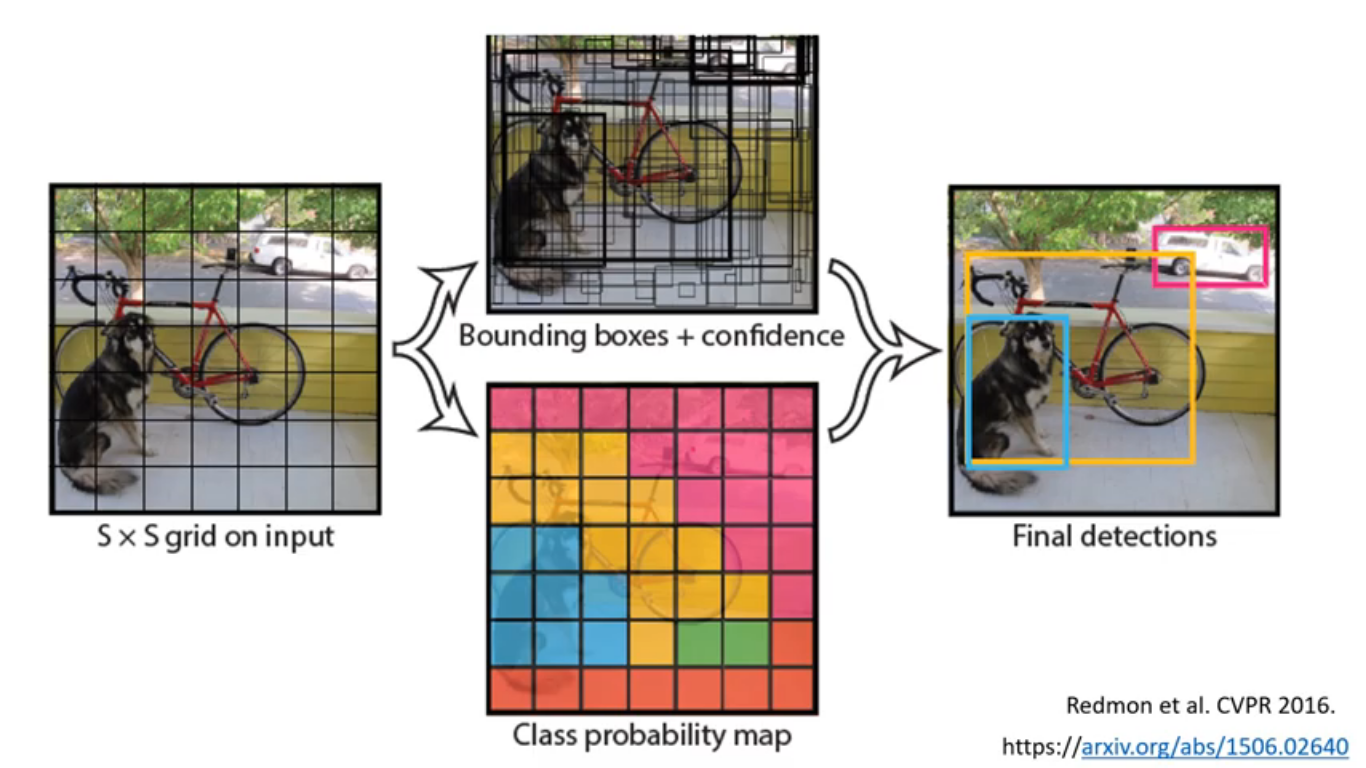


From the 845 total bounding boxes we only kept these three because they gave the best results. But note that even though there were 845 separate predictions, they were all made at the same time — the neural network just ran once. And that’s why YOLO is so powerful and fast.

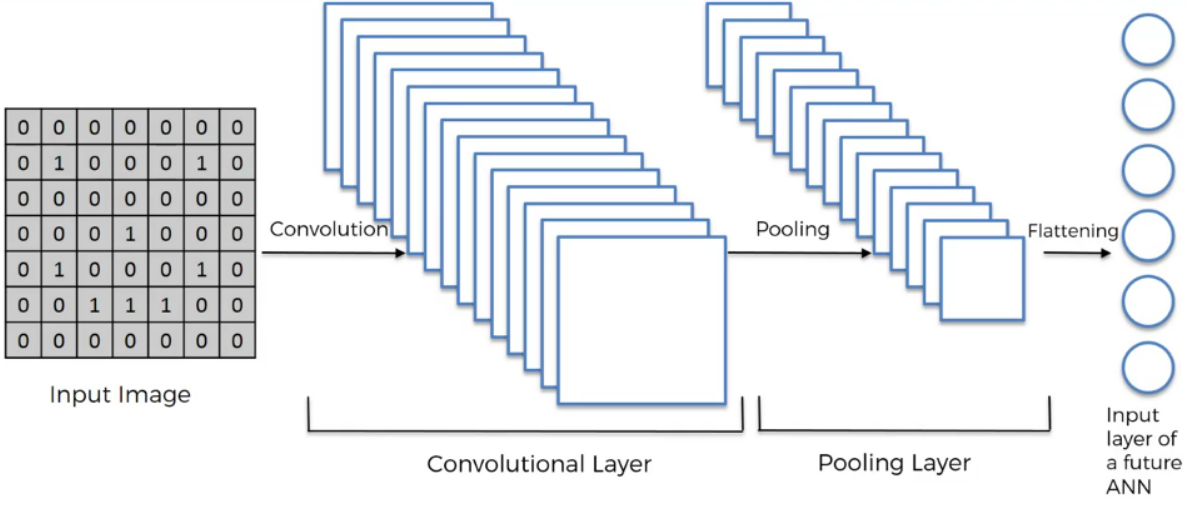
**4.2 Flow Chart:**

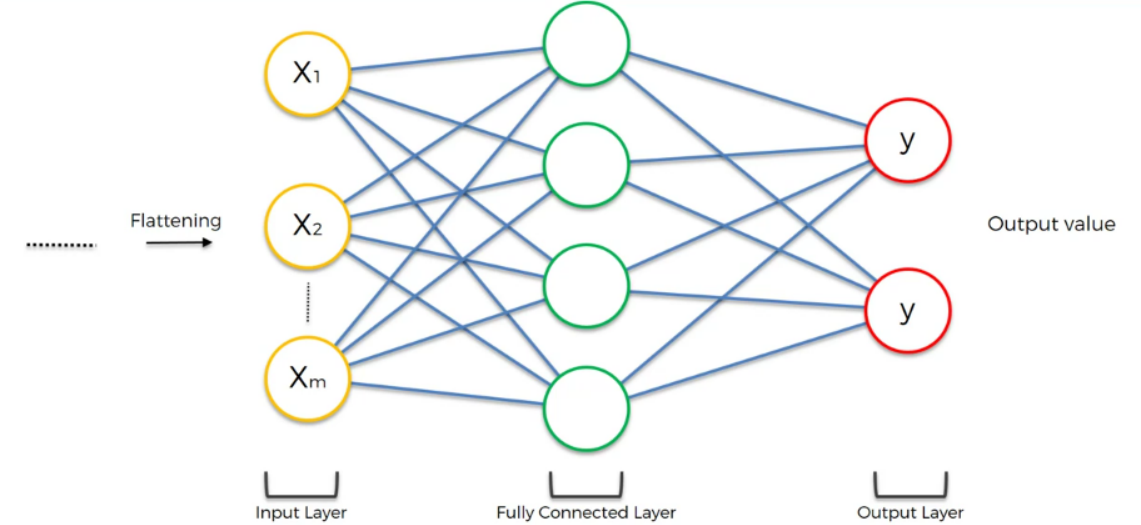
****

**4.3 YOLO Working**

****

**4.4 Convolutional Neural Network**

****

****

**5. Implementation**

**5.1 Dataset**

This system used a COCO dataset to train CNN. COCO stands for Common Objects in Context.

COCO is a large-scale object detection, segmentation, and captioning dataset. COCO has several features:

1. Object segmentation
2. Recognition in context
3. Superpixel stuff segmentation
4. 330K images (>200K labeled)
5. 1.5 million object instances
6. 80 object categories
7. 91 stuff categories
8. 5 captions per image
9. 250,000 people with keypoints

**5.2 Coding**

from absl import flags

import sys

FLAGS = flags.FLAGS

FLAGS(sys.argv)

import time

import numpy as np

import cv2

import matplotlib.pyplot as plt

import tensorflow as tf

from yolov3\_tf2.models import YoloV3

from yolov3\_tf2.dataset import transform\_images

from yolov3\_tf2.utils import convert\_boxes

from deep\_sort import preprocessing

from deep\_sort import nn\_matching

from deep\_sort.detection import Detection

from deep\_sort.tracker import Tracker

from tools import generate\_detections as gdet

import math

from itertools import combinations

####CUSTOM PARAMETERS############################################################

inputVideo='./inputs/peoplewalk.mp4'

# inputVideo='http://192.168.0.25:8080/video'#IP WebCam App

# inputVideo=0

outputVideoName='output'

showClassName=True

showTrackerId=False

showRenderingVideo=True

#\*\*\*\*\*\*\* COUNTING FEATURE (Total Count + Zonal(Band) Count) \*\*\*\*\*\*\*

activateCounting=False

objectsToTrack=["person"]

lineOrientationHorizontal=True

#0.5 Means line will be on middle of video vertically, small is closer to top

bandMidLineWrtHeightOrWidth=0.5

#upperBound is height\*upDownBoundWrtMidLine above mid line, similarly, lowerbound. Bigger the number, bigger is the area

upDownBoundWrtMidLine=0.1

#\*\*\*\*\*\*\* TRACKER TAIL FEATURE \*\*\*\*\*\*\*

activateTrackerTail=True

tailLengthInFrames=50

variableThicknessOfTrackerLine=False

#\*\*\*\*\*\*\* INCOMING OUTGOING FEATURE \*\*\*\*\*\*\*

activateIncomingOutgoing=False

objectsTrackInOut=["truck","car"]

incomingOutgoingLineHorizontal=True

incomingLineWrtHeightOrWidth=0.37

incomingLineThicknessWrtHeightOrWidth=0.025

outgoingLineWrtHeightOrWidth=0.43

outgoingLineThicknessWrtHeightOrWidth=0.025

#\*\*\*\*\*\*\* SOCIAL DISTANCE FEATURE \*\*\*\*\*\*\*

activateSocialDistance=False

distanceTreshold=75#in pixels

#############################################################CUSTOM PARAMETERS###

class\_names = [c.strip() for c in open('./data/labels/coco.names').readlines()]

yolo = YoloV3(classes=len(class\_names))

yolo.load\_weights('./weights/yolov3.tf')

max\_cosine\_distance = 0.5

nn\_budget = None

nms\_max\_overlap = 0.8

model\_filename = 'model\_data/mars-small128.pb'

encoder = gdet.create\_box\_encoder(model\_filename, batch\_size=1)

metric = nn\_matching.NearestNeighborDistanceMetric('cosine', max\_cosine\_distance, nn\_budget)

tracker = Tracker(metric)

vid = cv2.VideoCapture(inputVideo)

codec = cv2.VideoWriter\_fourcc(\*'XVID')

vid\_fps =int(vid.get(cv2.CAP\_PROP\_FPS))

vid\_width,vid\_height = int(vid.get(cv2.CAP\_PROP\_FRAME\_WIDTH)), int(vid.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

out = cv2.VideoWriter('./data/video/'+outputVideoName+'.avi', codec, vid\_fps, (vid\_width, vid\_height))

if activateTrackerTail:

from \_collections import deque

pts = [deque(maxlen=tailLengthInFrames) for \_ in range(1000)]#more the maxlen, more longer is the tracker tail

counter = []

for i in objectsToTrack:

counter.append([-1])#FOR EACH OBJECT LIKE PERSON OR CAR, A TOTAL COUNTER IS CREATED

incomingTrackIdsList=[]

outgoingTrackIdsList=[]

incomingCount=[]#in this list, index 0 will contain counts of objectsTrackInOut[0] object

outgoingCount=[]

for i in objectsTrackInOut:

incomingCount.append(0)

outgoingCount.append(0)

while True:

\_, img = vid.read()

if img is None:

print('COMPLETED')

break

img\_in = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

img\_in = tf.expand\_dims(img\_in, 0)

img\_in = transform\_images(img\_in, 416)

t1 = time.time()

boxes, scores, classes, nums = yolo.predict(img\_in)

classes = classes[0]

names = []

for i in range(len(classes)):

names.append(class\_names[int(classes[i])])

names = np.array(names)

converted\_boxes = convert\_boxes(img, boxes[0])

features = encoder(img, converted\_boxes)

detections = [Detection(bbox, score, class\_name, feature) for bbox, score, class\_name, feature in

zip(converted\_boxes, scores[0], names, features)]

boxs = np.array([d.tlwh for d in detections])

scores = np.array([d.confidence for d in detections])

classes = np.array([d.class\_name for d in detections])

indices = preprocessing.non\_max\_suppression(boxs, classes, nms\_max\_overlap, scores)

detections = [detections[i] for i in indices]

tracker.predict()

tracker.update(detections)

cmap = plt.get\_cmap('tab20b')

colors = [cmap(i)[:3] for i in np.linspace(0,1,20)]

current\_count = [0]\*len(objectsToTrack)#FOR EACH OBJECT LIKE PERSON OR CAR, A CURRENT(Band) COUNTER IS CREATED

centroid\_dict=dict()

for track in tracker.tracks:

if not track.is\_confirmed() or track.time\_since\_update >1:

continue

bbox = track.to\_tlbr()

class\_name= track.get\_class()

color = colors[int(track.track\_id) % len(colors)]

color = [i \* 255 for i in color]

cv2.rectangle(img, (int(bbox[0]),int(bbox[1])), (int(bbox[2]),int(bbox[3])), color, 2)

if showClassName and showTrackerId:

cv2.rectangle(img, (int(bbox[0]), int(bbox[1]-30)), (int(bbox[0])+(len(class\_name)+len(str(track.track\_id)))\*13, int(bbox[1])), color, -1)

cv2.putText(img, class\_name+"-"+str(track.track\_id), (int(bbox[0]), int(bbox[1]-10)), 0, 0.5, (255, 255, 255), 1)

elif showClassName:

cv2.rectangle(img, (int(bbox[0]), int(bbox[1]-30)), (int(bbox[0])+(len(class\_name))\*13, int(bbox[1])), color, -1)

cv2.putText(img, class\_name, (int(bbox[0]), int(bbox[1]-10)), 0, 0.5, (255, 255, 255), 1)

center\_y = int(((bbox[1]) + (bbox[3]))/2)

center\_x = int(((bbox[0]) + (bbox[2]))/2)

height, width, \_ = img.shape

if activateSocialDistance:

if class\_name=="person":

centroid\_dict[track.track\_id]=(int(center\_x), int(center\_y), int(bbox[0]), int(bbox[1]), int(bbox[2]), int(bbox[3]))

#######TRACKER TAIL##################################################################################

if activateTrackerTail:

center = (int(((bbox[0]) + (bbox[2]))/2), int(((bbox[1])+(bbox[3]))/2))

pts[track.track\_id].append(center)#pts stores track ids list and inside that list, it has old centres of objects

for j in range(1, len(pts[track.track\_id])):

if pts[track.track\_id][j-1] is None or pts[track.track\_id][j] is None:

continue

if variableThicknessOfTrackerLine:

thickness = int(np.sqrt(64/float(j+1))\*2)

else:

thickness = 2

cv2.line(img, (pts[track.track\_id][j-1]), (pts[track.track\_id][j]), color, thickness)

################################################################################TRACKER TAIL#########

#######COUNTING##################################################################################

if activateCounting:

if lineOrientationHorizontal:

#CREATE HORIZONTAL LINES (Zone or band)

cv2.line(img, (0, int(bandMidLineWrtHeightOrWidth\*height+upDownBoundWrtMidLine\*height)), (width, int(bandMidLineWrtHeightOrWidth\*height+upDownBoundWrtMidLine\*height)), (0, 255, 0), thickness=2)

cv2.line(img, (0, int(bandMidLineWrtHeightOrWidth\*height-upDownBoundWrtMidLine\*height)), (width, int(bandMidLineWrtHeightOrWidth\*height-upDownBoundWrtMidLine\*height)), (0, 255, 0), thickness=2)

if center\_y <= int(bandMidLineWrtHeightOrWidth\*height+upDownBoundWrtMidLine\*height) and center\_y >= int(bandMidLineWrtHeightOrWidth\*height-upDownBoundWrtMidLine\*height):

if class\_name in objectsToTrack:

index=objectsToTrack.index(class\_name)

current\_count[index] += 1

counter[index].append(int(track.track\_id))

else:

#CREATE VERTICAL LINES (Zone or band)

cv2.line(img, (int(bandMidLineWrtHeightOrWidth\*width+upDownBoundWrtMidLine\*width), 0), (int(bandMidLineWrtHeightOrWidth\*width+upDownBoundWrtMidLine\*width), height), (0, 255, 0), thickness=2)

cv2.line(img, (int(bandMidLineWrtHeightOrWidth\*width-upDownBoundWrtMidLine\*width), 0), (int(bandMidLineWrtHeightOrWidth\*width-upDownBoundWrtMidLine\*width), height), (0, 255, 0), thickness=2)

if center\_x <= int(bandMidLineWrtHeightOrWidth\*width+upDownBoundWrtMidLine\*width) and center\_x >= int(bandMidLineWrtHeightOrWidth\*width-upDownBoundWrtMidLine\*width):

if class\_name in objectsToTrack:

index=objectsToTrack.index(class\_name)

current\_count[index] += 1

counter[index].append(int(track.track\_id))

#################################################################################COUNTING########

###########INCOMING OUTGOING##############################################################################

if activateIncomingOutgoing:

if incomingOutgoingLineHorizontal:

#Horizontal Orientation

cv2.line(img, (0, int(incomingLineWrtHeightOrWidth\*height+incomingLineThicknessWrtHeightOrWidth\*height)), (width, int(incomingLineWrtHeightOrWidth\*height+incomingLineThicknessWrtHeightOrWidth\*height)), (255, 0, 0), thickness=2)

cv2.line(img, (0, int(incomingLineWrtHeightOrWidth\*height-incomingLineThicknessWrtHeightOrWidth\*height)), (width, int(incomingLineWrtHeightOrWidth\*height-incomingLineThicknessWrtHeightOrWidth\*height)), (255, 0, 0), thickness=2)

cv2.line(img, (0, int(outgoingLineWrtHeightOrWidth\*height+outgoingLineThicknessWrtHeightOrWidth\*height)), (width, int(outgoingLineWrtHeightOrWidth\*height+outgoingLineThicknessWrtHeightOrWidth\*height)), (0, 0, 255), thickness=2)

cv2.line(img, (0, int(outgoingLineWrtHeightOrWidth\*height-outgoingLineThicknessWrtHeightOrWidth\*height)), (width, int(outgoingLineWrtHeightOrWidth\*height-outgoingLineThicknessWrtHeightOrWidth\*height)), (0, 0, 255), thickness=2)

if class\_name in objectsTrackInOut:

if center\_y <= int(incomingLineWrtHeightOrWidth\*height+incomingLineThicknessWrtHeightOrWidth\*height) and center\_y >= int(incomingLineWrtHeightOrWidth\*height-incomingLineThicknessWrtHeightOrWidth\*height):

#Incoming zone touched

objectTrackId=int(track.track\_id)

if objectTrackId not in incomingTrackIdsList:#Added because multiple same objectTrackId were appended to incomingTrackIdsList because of frames

incomingTrackIdsList.append(objectTrackId)

if objectTrackId not in outgoingTrackIdsList:

index=objectsTrackInOut.index(class\_name)

incomingCount[index]+=1

outgoingTrackIdsList.append(objectTrackId)#Added so that a person is only counted once

if class\_name in objectsTrackInOut:

if center\_y <= int(outgoingLineWrtHeightOrWidth\*height+outgoingLineThicknessWrtHeightOrWidth\*height) and center\_y >= int(outgoingLineWrtHeightOrWidth\*height-outgoingLineThicknessWrtHeightOrWidth\*height):

#Outgoing zone touched

objectTrackId=int(track.track\_id)

if objectTrackId not in outgoingTrackIdsList:

outgoingTrackIdsList.append(objectTrackId)

if objectTrackId not in incomingTrackIdsList:

index=objectsTrackInOut.index(class\_name)

outgoingCount[index]+=1

incomingTrackIdsList.append(objectTrackId)#Added so that a person is only counted once

else:

#Vertical Orientation

cv2.line(img, (int(incomingLineWrtHeightOrWidth\*width+incomingLineThicknessWrtHeightOrWidth\*width), 0), (int(incomingLineWrtHeightOrWidth\*width+incomingLineThicknessWrtHeightOrWidth\*width), height), (255, 0, 0), thickness=2)

cv2.line(img, (int(incomingLineWrtHeightOrWidth\*width-incomingLineThicknessWrtHeightOrWidth\*width), 0), (int(incomingLineWrtHeightOrWidth\*width-incomingLineThicknessWrtHeightOrWidth\*width), height), (255, 0, 0), thickness=2)

cv2.line(img, (int(outgoingLineWrtHeightOrWidth\*width+outgoingLineThicknessWrtHeightOrWidth\*width), 0), (int(outgoingLineWrtHeightOrWidth\*width+outgoingLineThicknessWrtHeightOrWidth\*width), height), (0, 0, 255), thickness=2)

cv2.line(img, (int(outgoingLineWrtHeightOrWidth\*width-outgoingLineThicknessWrtHeightOrWidth\*width), 0), (int(outgoingLineWrtHeightOrWidth\*width-outgoingLineThicknessWrtHeightOrWidth\*width), height), (0, 0, 255), thickness=2)

if class\_name in objectsTrackInOut:

if center\_x <= int(incomingLineWrtHeightOrWidth\*width+incomingLineThicknessWrtHeightOrWidth\*width) and center\_x >= int(incomingLineWrtHeightOrWidth\*width-incomingLineThicknessWrtHeightOrWidth\*width):

#Incoming zone touched

objectTrackId=int(track.track\_id)

if objectTrackId not in incomingTrackIdsList:#Added because multiple same objectTrackId were appended to incomingTrackIdsList because of frames

incomingTrackIdsList.append(objectTrackId)

if objectTrackId not in outgoingTrackIdsList:

index=objectsTrackInOut.index(class\_name)

incomingCount[index]+=1

outgoingTrackIdsList.append(objectTrackId)#Added so that a person is only counted once

if class\_name in objectsTrackInOut:

if center\_x <= int(outgoingLineWrtHeightOrWidth\*width+outgoingLineThicknessWrtHeightOrWidth\*width) and center\_x >= int(outgoingLineWrtHeightOrWidth\*width-outgoingLineThicknessWrtHeightOrWidth\*width):

#Outgoing zone touched

objectTrackId=int(track.track\_id)

if objectTrackId not in outgoingTrackIdsList:

outgoingTrackIdsList.append(objectTrackId)

if objectTrackId not in incomingTrackIdsList:

index=objectsTrackInOut.index(class\_name)

outgoingCount[index]+=1

incomingTrackIdsList.append(objectTrackId)#Added so that a person is only counted once

initialHeight=60

if activateIncomingOutgoing:

for objectName in objectsTrackInOut:

index=objectsTrackInOut.index(objectName)

cv2.putText(img, "Incoming "+objectName+"s: " + str(incomingCount[index]), (10, initialHeight), 0, 0.8, (0, 0, 255), 2)

initialHeight+=30

cv2.putText(img, "Outgoing "+objectName+"s: " + str(outgoingCount[index]), (10,initialHeight), 0, 0.8, (0,0,255), 2)

initialHeight+=30

###########################################################################INCOMING OUTGOING##############

##########SOCIAL DISTANCING FEATURE###############################################################################

if activateSocialDistance:

red\_social\_dis\_zone\_list=[]

red\_social\_dis\_line\_list=[]

for (id1,p1), (id2,p2) in combinations(centroid\_dict.items(),2):

dx,dy=p1[0]-p2[0], p1[1]-p2[1]

distance = math.sqrt(dx\*dx + dy\*dy)

if distance < distanceTreshold:

if id1 not in red\_social\_dis\_zone\_list:

red\_social\_dis\_zone\_list.append(id1)

red\_social\_dis\_line\_list.append(p1[0:2])

if id2 not in red\_social\_dis\_zone\_list:

red\_social\_dis\_zone\_list.append(id2)

red\_social\_dis\_line\_list.append(p2[0:2])

for idx, box in centroid\_dict.items():

if idx in red\_social\_dis\_zone\_list:

cv2.rectangle(img, (box[2],box[3]),(box[4],box[5]), (0,0,255), 2)

else:

cv2.rectangle(img, (box[2],box[3]),(box[4],box[5]), (0,255,0), 2)

cv2.putText(img, "People at risk: "+str(len(red\_social\_dis\_zone\_list)), (10,initialHeight), 0, 0.8, (0,0,255), 2)

initialHeight+=30

for check in range(0, len(red\_social\_dis\_line\_list)-1):

start\_point=red\_social\_dis\_line\_list[check]

end\_point=red\_social\_dis\_line\_list[check+1]

check\_line\_x=abs(end\_point[0]-start\_point[0])

check\_line\_y=abs(end\_point[1]-start\_point[1])

if (check\_line\_x<distanceTreshold) and (check\_line\_y<distanceTreshold):

cv2.line(img, start\_point, end\_point, (0,0,255), 2)

#############################################################################SOCIAL DISTANCING FEATURE############

#######COUNTING##################################################################################

if activateCounting:

for objectName in objectsToTrack:

index=objectsToTrack.index(objectName)

cv2.putText(img, "Inside Zone "+objectName+" count: " + str(current\_count[index]), (10, initialHeight), 0, 0.8, (0, 0, 255), 2)

initialHeight+=30

total\_count = len(set(counter[index]))-1

cv2.putText(img, "Total "+objectName+" count: " + str(total\_count), (10,initialHeight), 0, 0.8, (0,0,255), 2)

initialHeight+=30

#################################################################################COUNTING########

cv2.putText(img,"Made by Karsatmo", (10,initialHeight), 0, 0.8, (0,0,255), 2)

fps = 1./(time.time()-t1)

cv2.putText(img, "FPS: {:.2f}".format(fps), (10,30), 0, 0.8, (0,0,255), 2)

#cv2.resizeWindow(outputVideoName, 1024, 768)

if showRenderingVideo:

cv2.namedWindow(outputVideoName, cv2.WND\_PROP\_FULLSCREEN)

cv2.setWindowProperty(outputVideoName,cv2.WND\_PROP\_FULLSCREEN,cv2.WINDOW\_FULLSCREEN)

cv2.imshow(outputVideoName, img)

else:

print("FPS: {:.2f}".format(fps))

out.write(img)

if cv2.waitKey(1) == ord('q'):

print("STOPPED")

break

vid.release()

out.release()

cv2.destroyAllWindows()

**6. Conclusion**

This system is useful in understanding the human density in a particular space.It is useful in deriving business insights in an off-line store.This system can do the task of managing the space effectively & efficiently. Thus AI Eye Prognosticator can understand the people's movement pattern in a place & make optimized utilization of space.

33

**7. References**

1. Ya-Li Hou, Student Member, IEEE, and Grantham K. H. Pang, Senior Member, IEEE, “People Counting and Human Detection in a Challenging Situation”, IEEE Transactions on systems, man and cybernetics part A systems and humans, vol. 41, no. 1, January 2011.
2. Manoranjan Paul, Shah M E Haque and Subrata Chakraborty, “Human detection in surveillance videos and its applications - a review”, Paul et al. EURASIP Journal on Advances in Signal Processing 2013, 2013:176 http://asp.eurasipjournals.com/content/2013/1/176.
3. Xiaohang Xu, Dongming Zhang, and Hong Zheng, “Crowd Density Estimation of Scenic Spots Based on Multifeature Ensemble Learning”, Hindawi, Journal of Electrical and Computer Engineering Volume 2017, Article ID 2580860, 12 pages.
4. Joanna M. Burger, Frances E.C. Stewart, John P. Volpe, Jason T. Fisher, A Cole Burton, “Estimating density for species conservation: Comparing camera trap spatial count models to genetic spatial capture-recapture models”, Global Ecology and Conservation, July 2018.
5. G. Sreenu and M. A. Saleem Durai, “Intelligent video surveillance: a review through deep learning techniques for crowd analysis”, Sreenu and Saleem Dural Big Data (2019) 6:48 https://doi.org/10.1186/s40537-019-0212-5.
6. R Schrijvers S. Puttemans, T. Callemein', and T. Goedemé, “Real-time Embedded Person Detection and Tracking for Shopping Behaviour Analysis”, EAVISE, KU Leuven, Jan De Nayerlaan 5, 2860 Sint-Katelijne-Waver, Belgium PXL Smart-ICT, Hogeschool PXL, Elfde Liniestraat 24, 35000 Hasselt, Belgium.
7. Volker Eiselein, Ivo Keller, Hajer Fradi, Thomas Sikora, “Enhancing human detection using crowd density measures and an adaptive correction filter”, Conference Paper August 2013, DOI: 10.1109/AVSS.2013.6636610.
8. Mayur D. Chaudhari, Archana S. Ghotkar, “A Study on Crowd Detection and Density Analysis for Safety Control”, JSCE International Journal of Computer Sciences and Engineering, April,2018.
9. V. Keerthi Kiran, Priyadarsan Parida and Sonali Dash, “Vehicle Detection and Classification: A Review”, Chapter - January 2021.
10. Sriashika Addala, Dept of CSE, LPU Punjab, “Research paper on vehicle detection and recognition”, Experiment Findings May 2020.

34