Social distancing detection with computer vision techniques

Lee Hoang

BSc Computer Science
City University, London
Date

Contents

Chapter 1

Introduction

Chapter 2

Output Summary

Chapter 3

Background Research

3.1 What is computer vision?

Computer vision (CV) is a scientific discipline that studies how computer scan efficiently perceive, process, and understand information from visual data such as images and videos.

As humans, we can classify three-dimensional objects with ease, whether the pictures are the same object with different colours or angles, we are good at determining the object we are classifying. Computer vision has been developed to detect edges from a pixelated image, face detection, and has been used to develop 3D models from a snapshot yet the technology we have today could be compared to a young child's biological vision.

Computer vision is used in various real world application such as traffic surveillance or medical imaging (SZELISKI, 2020), where people are now able to utilize magnetic resonance imaging (MRI) to safely analysis the heart wall motion where the end result is a 3d model of the heart pumping (Metaxas, 1997).

In recent years, computer vision has been adapting deep learning algorithms to efficiently classify unseen objects within pixel images and videos.

3.2 Deep learning

Deep learning uses artificial intelligence (AI) to try and simulate the choices that a human brain will make. Problems that have regression or classification outputs can be solved by passing data/inputs through artificial neurons which were previously tweaked for the specific problem by training data. There are many different variants of deep learning algorithms such as Artificial Neural Networks (ANN) or Long Short-Term Memory (LSTM) Networks (Hochreiter, 1997) which build onto each other.

3.2.1 Neural networks (NN)

Neural networks is a network formed of interconnected perceptrons which each carry weights and biases. The weights (w) and biases (b) are each represented by a float value which are used to multiply and add to the input respectively. The outcome is then put into an activation function (e.g. ReLU or sigmoid) which determines if the

neuron should be activated. These activations chain together to output a value of what the neural network thinks the solution is.

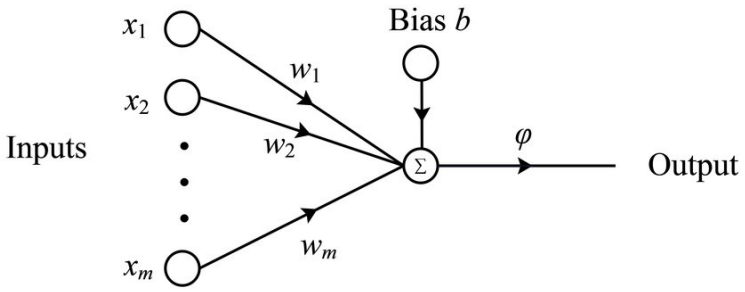

Figure 1: A single perceptron with inputs, weights, and biases

The values of weights and biases are updated when training data is pass through the network using back propagation, adjusting their values depending on the actual and network's solution to the problem. A network can have multiple layers of several perceptrons in order to solve more complex problems, but has the down side of overfitting the accuracy of the output.

3.2.2 Convolutional neural networks (CNN)

CNNs builds upon the NN, specialising in working with grid-structured data such as images or videos. Unlike NN which takes in a 1 dimensional vector as its input, CNN uses tensors (high dimensional vector/matrix) for processing images. The convolutional layer of the network includes kernels/filters (a type of matrix) which efficiently detects features within an image by calculate the sum of the kernel multiplied by a subgrid of the image. These filters are trained to extract details within the images, and to make better predictions over the course of training. Different techniques can be applied such padding to reduce the lost of data when filtering.

The information about the deep learning architectures we have discussed were influenced by taking part in the 'Introduction to AI' (IN3062) and 'Programming and Mathematics for Artificial Intelligence' (IN3063) module at City University.

3.3 WILDTRACK dataset

The 'WILDTRACK' dataset provides footage of pedestrians from 7 different angle, each video having a length of 35 minutes and were filmed using three 'GoPro Hero 4' and four 'GoPro Hero 3' at high resolution (1920x1080). The location of the video took place near a public research university called ETH Zurich, Switzerland. The cameras used were calibrated to allow for precise calculations when wanting the distance between two objects.

While there are other datasets that offer similar features such as the 'EPFL-RLC' dataset, the videos themselves do not contain the same density of pedestrians relative to the 'WILDTRACK' dataset.Footage from other datasets showcase people who are more static, making it less challenging for the project.

3.3.1 Research papers with WILDTRACK dataset

There are many published research papers that have used this dataset for their own project. For example, a research paper that was dedicated to detecting the same pedestrian using all camera footage and creating a shared top down view of the 'point of interest' which indicates the location of the pedestrians (López-Cifuentes A. 2018). What was interesting about this paper was that the author used another dataset that did not include calibrated cameras. When comparing the results at the end, the accuracy of the 'WILDTRACK' was marginally better as the other dataset had calibration errors, therefore the 'point of interest' were at different locations when looking at the shared top down view.

3.4 Existing systems

Many different social distancing detections have been made ever since the outbreak of the corona virus. Most of the systems use deep learning architectures paired with the OpenCV library to help classify pedestrians within a video.

3.4.1 Example (Y. C. Hou, 2020)

This system uses a combination of the YOLO (You Only Look Once) model with the COCO (Common Objects in Context) dataset to train their model. The goal of the project was to produce a top down view of the pedestrians, showing the distance between each person who were identified within the frame. The results of the system were very accurate, as they were able to make use of calibrated cameras. What was very interesting about the project is that some pedestrians were not classified due to hiding behind others. This showcases the limitation of the dataset used, as there were no overlapping footage of the field of view. Further improvements to the system were suggested such as mask and human body temperature detection. The system overall is similar to what this project will achieve with the difference of dataset.

Chapter 4

Method

4.1 Agile development

An agile approach with sprints were used during the development of the project. The approach gave time to reflect and adjust the work being done through out the weeks while allowing focus on the core functionalities of the software being produced.

The outcome of each sprint allowed for early prototypes of the system, easier analysis on code and product, and fixing any bugs within the code. The consequences of only partially completing a sprint's objective can be solved by adding more time for further development. Partially completion of the sprints are inevitable therefore spacing out the sprints instead of one immediately after the other will help allocate more time.

4.2 Management tools

A diary is used to record sprints and work done throughout the week, recording the results produced and research done in order to produce code. Any images/video footage used in the dataset are copied and backed up into another folder for future reference. A simple Gantt chart is used to keep the work flow on track while also updating the chart to allocate time for sprints.

4.2.1 Version control

GitHub was used to store previous versions of the code used for the project. This is in case of any mistakes made during development, backup versions of the project can be restored. GitHub can allow the user to backup their version no matter how small the changes are, providing flexibility throughout the project.

4.3 Deep learning architectures for object detection

Deep learning has been a foundation for modern computer vision, allowing object detection to be 'automatic' by training a CNN which tunes itself after each batch of training data, then being further developed by implementing algorithms for object detection which only requires one pass through the network. This project will specifically be using yolov3 and SSD architectures.

4.3.1 YOLO (You Only Look Once)

YOLO is a object detection architecture that uses convolutional neural networks to divide the image/input into a grid. Each box in the grid is then associated with a high dimensional vector which record data such as: if there is an object within the grid, the predicted position of the bounding box, and the class id of the object. The vector can be expanded if there is more than one object within the box which is called anchor boxes. See appendix 1 for a flowchart of YOLO.

It is possible for the architecture to identify the same object twice within the same box creating redundancy. Since the predictions are based on probability, the architecture chooses the highest probability and uses there bounding box to identify the object. This feature is called 'Non-Max suppression'.

There are many versions of the 'yolo' such as 'yolov2' but during 2018, 'yolov3' was released (REDMON J. 2018). When comparing to previous versions, it offers an increase in speed and efficiency when computing while also providing a better backbone classifier (The core convolutional neural network).

There are also different models of 'yolov3' that perform better speed wise but at the cost of accuracy such as 'yolov3-tiny' which can compute videos at 220 frames per second. This project specifically will use 'yolov3-320' which will detect objects within a video at 45 frames per second while still having good accuracy. The reason for using this model is that realistically, cameras used in offices or public areas for surveillance will most likely be around 30 frames per second.

While this method of object detection is proven to be very effective, it comes with the drawback of not being very flexible when wanting to change the overall structure of the neural network. Its very likely that tweaking the structure if possible could make the outcome of the object detection worst, by also spoils the chances of improving it.

4.3.2 SSD (Single shot detection)

4.4 Pre-trained models (With COCO dataset)

The Microsoft 'COCO' (Common Objects in Context) dataset is a large-scale object detection dataset that contains over 330,000 images and more than 80 different object categories. This project used a pre-trained 'yolov3-320' model with the 'COCO' dataset to detect people within an image. The reasoning for using this is that the dataset is very large, and will take several days to train the network. In comparison, downloading the pre-trained weights (Redmon, 2021) took a few minutes.

Another reason for using 'COCO' is that it is a well documented dataset (Lin et al., 2021), designed to help improve 'image classification' with images that contain multiple objects.

The objects that are in the dataset range from living beings such as 'person' and 'cat' to everyday objects such as 'bottle' and 'cup'. The main focus of the project is to detect 'person', therefore we will need to filter out the other objects within the code, but this means that the forward pass in the neural network will still detect other objects, possibly affecting the time spent to compute a simple image.

4.5 Programming language and libraries

This section discusses the chosen programming language and libraries used in the project, giving examples of functionalities within the coding aspect.

4.5.1 Python

Python was chosen for flexibility when organising and presenting code through out the project. Python allows the user to call modules from different files that contained reusable classes and functions, making the overall structure of the code less cluttered.

Python has compatibility with libraries needed for the project which were installed using 'pip' which is a package management system.

4.5.2 NumPy

NumPy specialises in matrix/array arithmetic, being able compute high dimensional matrix multiplication with ease. For this project, the project use NumPy for generating matrix transformations and to concatenate matrices to generate a side panel for the top down view.

4.5.3 OpenCV

OpenCV is a cross-platform library which can be used to develop real-time computer vision applications. OpenCV can be used to utilise the pre-trained weights previously discussed, allowing to input an image and filter out the objects within the image.

Another functionality of the OpenCV library is image transformation, transforming a plane in an image to a 'flat surface'.

4.5.4 Matplotlib with Seaborn

Matplotlib is a library that specialises in creating plots and graphs. The seaborn library will be primarily used for the presentation of the plots from matplotlib.

4.6 Dataset preparation

The dataset features 7 videos of pedestrians walking in a public space (each video are at different angles) that are each 35 minutes long and are shot at 60 frames per second. It also includes image subsets of the videos, each one taken every tenth of a second from the video with a resolution of 1920x1080. These images are already post-processed to remove distortion.

Due to the large dataset, this project will be utilising samples of screenshots from each camera used to test the capabilities of the 'yolov3' architecture, and will be augmented with artificial noise which allows for extreme test.

The videos are sampled to 5 - 10 second clips using windows video editor. Each clip will contain a variety of pedestrians doing different actions such as being stationary or walking side by side in a group. The clips are still at 1920x1080 resolution, but are

reduced to 30 frames per second. This is because the industry standard for surveillance cameras are 30 frames per second and it will take less time to compute a video.

Figure 4.1 is a table that labels each image/video used in the report, giving a name that will be referenced throughout the report, what camera the video came from, the time the clip started from the original video and length or the frame it came from, and a short description on the main activity in the image/clip.

Image/video name	Camera	Start time /frame	Length (Seconds)	Description
image 1	1	0	N/A	Large crowd in the background with a group of 3 in the middle.
image 2	4	730	N/A	A person in front of the other.
image 3	4	950	N/A	A large crowd of people close to the camera and in the background.
image 4	5	0	N/A	Two people hugging. Two people walking side by side with one blocking the other.
video 1	1	15:02	8	Group of stationary people with a person walking by.
video 2	1	19:54	8	A single person walking past.
video 3	2	23:27	6	Two people walking past each other with close proximity.
video 4	2	16:43	10	Group of four people walking side by side.
video 5	3	1:00	10	Large crowd of people walking past.
video 6	3	32:49	7	Two people making contact with each other.
video 7	6	7:05	8	Two people walking together with their backs to the camera.
video 8	6	22:33	8	A cluster of four people walking with their backs to the camera.
video 9	7	29:05	8	Two people hugging each other.
video 10	7	14:30	8	Stationary group and a moving group side by side.

Figure 4.1: Table of images and video samples used.

4.7 System model

This section discusses the overall flow of the system, how it uses pre-trained model, and the different approaches to detecting close proximity

The flow of the system was model with a activity diagram (figure 4.2), showcasing the step by step process of the overall system and how it interacts with 'yolov3'. The system has three different methods to calculate close proximity of two people and will be later discussed in the results section.

The goal of the system is to mark pedestrians who are close to each other with a red bounding box and the people who are not close with a green bounding box. The system should be able to mark these onto a side panel which represents the top down view of the image. If the input is a video, then the system should calculate all of the step for every frame within the video, the outcome should be the same video but with new annotations included.

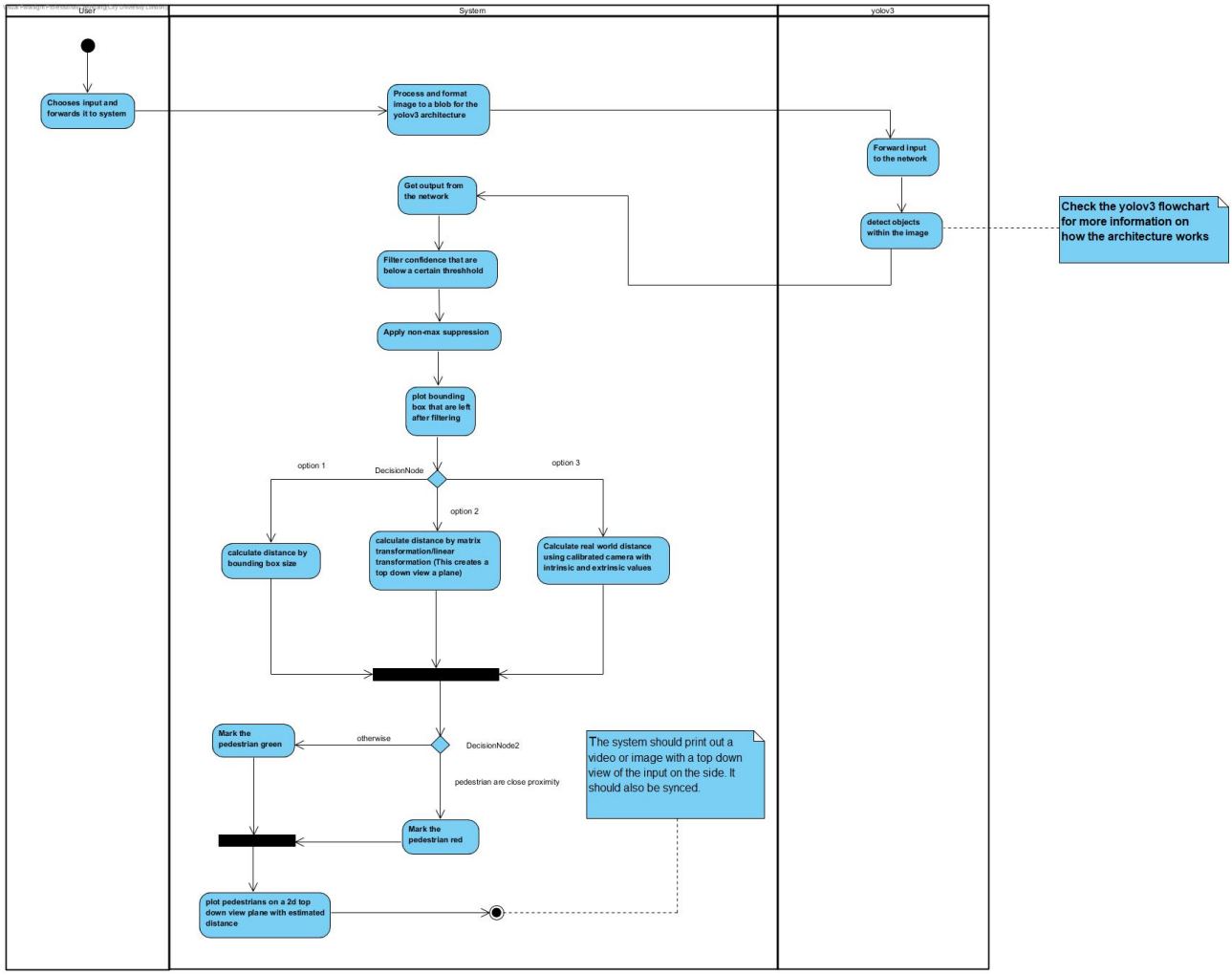

Figure 4.2: Activity diagram describing the system pipeline

A class diagram (appendix 2) was created to show the functions that occur through out the system with the inputs and outputs. Each function also includes the key parameters needed, as well as the function data type.

4.8 Development phases

The project has several development phases to give time to thoroughly test each feature of the system. These phases will be done in chronological order and will each be treated as a sprint.

4.8.1 1: object detection with YOLO

This phase will focus on forwarding images to the network to test how well the object detection works. The images used in the testing will have different density of population in the image, where some images will contain people overlapping another. The same images will also be artificially augmented by adding noise or rotation to image to limit test the architecture. This phase will also be testing videos as inputs, the main measurement of success will be how smooth each individual is tracked through out the processing.

4.8.2 2: Detecting social distancing by bounding box distance

By using the found bounding box in the first phase, we should be able to compute the distance between two people using pixels as a estimated measurement. This will be tested with different camera angles to see how affective the implementation is.

4.8.3 3: Detecting social distancing by matrix transformation

This method can be achieved by using the OpenCV library to calculate a matrix that transforms a quadrilateral in an image that is marked with 4 points to a square image.

4.8.4 4: Detecting social distancing by camera calibration

The dataset provides intrinsic and extrinsic values for the cameras used. This enables the project to convert pixel values to real world measurements, acquiring a very accurate solution to the problem.

4.8.5 5: Plotting the found distances on a top down view of the image

4.8.6

Chapter 5

Results

This section discusses the results found throughout the project.

5.1 Detecting objects with 'yolov3' (Analysis)

OpenCV enabled the use of 'yolov3' with the functions of loading the weights of the neural network and forwarding an image to output the objects detected. The first test that was done was to showcase how well the model detected objects. The model was tested with different image provided by the dataset.

The first image that was forwarded was from the third camera angle (figure 5.1). The main focus of using this specific image was to look at how well the model detected people who were overlapping each other. The outcome of model was over 10,000 objects detected within the image, taking 0.54 seconds to compute. The amount of objects detected might seem unnecessary, but the model was trained to detect 80 different objects, therefore will provide a large number of classifications per image.

Each classification comes with a probability that the object detected is the correct classification. Therefore objects that are 'person' could also have a probability of being identified as a 'car'.

5.1.1 Filtering classification methods

The probabilities can be filtered by tuning the number of objects that can be classified and the probability threshold.