Deep Learning Based People Counter

Vivian Lopez

A Level Computer Science Programming Project

Exam Centre: 14285

Exam Number: 5183

Contents

Analysis Phase

Problem Definition

There exist solutions to count people using methods such as Haar Cascades which would be an example of traditional object detection, there also exists newer and more recent methods such as deep learning to identify objects. In this project, I hope to leverage deep learning to conduct people detection and give a resulting count. To provide counts of people in a room, users would have to often purchase an expensive product that is designed with many complex features that are often more than what the user needs. I want to develop a solution that is lightweight and at its most basic form, provides a count for people in a room accurately with a clean user interface and doesn't require highly expensive hardware so that it can be implemented in areas where it could be useful without the restriction of cost.

There are current systems with the same functionality however they are often overly expensive with price of over £1000, I am planning on developing a compact system which is significantly less costly and still produces accurate results.

To solve the problem, I am planning on using pre-existing libraries and implementing them so that I can detect people that pass in front of the camera and add them to a count depending on whether they are going in or out. I can check whether they are entering or exiting by tracking detected people and seeing whether they end up at the top or bottom of the camera, thereby indicating whether they are entering or exiting.

Information I need to gather will include what kind of rooms the system will be used in, if there is a dedicated entrance and exit then the system will just need to have a total count of people, and anyone detected passing through the entrance should increment the count and anyone passing through the exit should decrement the count. Another situation could be where a single door is used as an entrance or exit, in this situation, I will have to use tracking to check if people are exiting or entering.

My system will have to be robust and be able to deal with lots of people as well and the different kinds of rooms which it can be implemented, I.e., if there are multiple entrances and exits. I need to find out how many people may be using the rooms, if there are high volumes of people entering or exiting at once then the system will have to be more accurate and efficient in detecting people so may require more training when using a model to detect.

Stakeholders

The end user could be a teacher who wants to see the number of students attending a lesson or a society leader who wants to check the number of people attending a society. The

end user can be anyone who wants to check the number of people present in a room. This could also be businesses and shop owners who want to check how many customers are entering their shop. They can use the data collected to see the peak times when the shop is the most popular and can use this information to improve sales.

The end user, in the case of being a teacher could use the number of students counted to feedback to the school or department which can be used for example when there is a fire drill to quickly check if there are students in rooms. The system can also be used to ensure that people limits aren't met to ensure social distance and provide safety measures in buildings such as schools and workplaces so the owners would be the main users of the system to ensure safety of the building.

Any business or company who would benefit from seeing how many people are entering and exiting an area would benefit from this solution to improve services or ensure safety.

Justification of Computational Methods

Since the problem requires object detection deep learning and the use of neural networks can be used to create a model which can then be used for detection. Libraries in the TensorFlow platform can be used when developing the model using python. Also, OpenCV is another library which can be used to for example draw rectangles around the detected people which can be displayed if necessary or desired. Essentially, deep learning is perfect for this problem as it allows for an efficient way to detect people with high accuracy.

The detections can be directly recorded and an output of detections (showing boxes around detected people passing through) can be passed to an external device to be displayed. The data gained can be analyzed directly and conclusions can be made as a result such as the times at which an entrance is used most often. This means that using object detection is important as other methods that do not use object detection such as using infrared rays and detectors will not be able to give the same output and will not be able to analyze the data to the same level.

Also, the solution requires communication between different hardware device such as the camera, the device conducting inference and the device accessing the information via an app or website. This will allow for the data gathered from the detections to be displayed to external devices such as phones or laptops which would be convenient for users to use instead of checking the device conducting the inference directly, they can check the count of people when in a reasonably close proximity to the system.

Abstraction

Abstraction can be used when conducting the detection, as only the people are needed to be detected, the details of the people themselves are unnecessary, it only matters that they

are people so the model can be trained to classify the objects in the frames of the live video as people or not people. This means that the output of the model is much simpler than if it was looking for different kinds of people like students or teachers. Abstraction can also be used when displaying output to the user, however this would depend on what the user wants, if they want lots of output then other data such as what times people were detected and actual output of the detections with boxes drawn around the people may also be relevant. The specific amount of abstraction depends on what the user wants so can be adjusted to fit the user's preferences. A settings area can be used to tweak preferences to meet user's needs. For example, there could toggles to toggle on live output of the camera showing locations of detections and bounding boxes around the people. There would be more toggles for more features.

Reusability

Certain components can be reused, such as using the trained model to detect people on each frame of the video given by the camera. This means that if the model is trained and is effective, the system will perform well.

Also, the function that carries out the detections can be re-used when carrying out detections on the webcam feed, so that every time a frame is received, it can be processed. This is a particular part where re-using a part of code will be crucial when running the program.

In addition, when creating the model, the section required for image collection can be re-used, specifically the section involving webcam use as this will be needed throughout the project. This includes for example accessing a frame from a webcam, displaying the output of the webcam to the user. Due to this re-occurring component of the project, I will likely be using a library such as OpenCV which is highly useful when working with images and webcam feeds. The code from the library can therefore be re-used at several points.

Problem Decomposition

Decomposition can be applied by splitting the main problem into different sub-problems. These could include:

- Developing the model to detect people in each frame.
- Tracking the detected people and checking whether they pass the entrance or exit.
- Tracking whether people move from top to bottom or bottom to top to see if they are entering or exiting.
- Sending the output to the website to be displayed to the user.
- Developing the website to receive output and display it to the user.

Logical Decisions

The logical decisions would include:

- Checking whether there are people in each frame, and if so, how many, (This would be done using the trained model)
- Checking whether the people are moving from top to bottom or bottom to top to check if they are entering or exiting (This could be done using OpenCV, drawing a dot on the people and checking the location of the dot in the image and if it passes a certain horizontal line (then the pixel y-value can be checked to see if entered the room or exited it).
- Checking if they are any detections of a high enough confidence, i.e., if there is a detection with 99% confidence then it should be included however if there is a detection of 30% confidence then it shouldn't be included as the detection is unlikely to be a person.

Tackling the Problem Concurrently

Some parts can be tackled concurrently:

- If the user has a preference to see output for the detections, then the model can be used to detect people and at the same time, the location of the people can be gained which will allow boxes to be drawn around the people also and this can be sent to the app or web server to display It to the user. Functions for these can be written, i.e., one subroutine for developing the model and one for drawing boxes around the coordinates gained from the detection.
- The app or website can also be developed at the same time as they only require data from the camera and device conducting the inference and this can be gained later on when testing needed.

Investigation

Interview Questions and Responses

Interview Questions:

1) What kinds of features would you like to be able to use when using the system?
2) How many features would be appropriate when displaying output for the system?
3) What kinds of rooms would you use the system in? (i.e., separate entrances and exits or combined entrances and exits)

Q1: I have chosen this as it will give me an indication as to what kinds of features would be necessary in the software so that the users are likely to use them.

Q2: I have chosen this as it will give me an idea of how much the user wants to have the ability to see when getting output on the website/app.

Q3: I have chosen this as it will help me understand if I need to cater the solution to certain environments.

Interview answers:

Teacher:

1) Some features I would like would be to be able to see the immediate count of students as well as give a count of students who arrive past the bell so that I can clearly see if a student is late. Also, I would like to see how many students are late on average. Also, seeing live output would allow me to easily see who is late.
2) I would not like to many features as it should be easy to use so 5 would be more than enough.
3) I teach in a range of different rooms so rooms of both types will be used.

Shop owner:

1) I would like to be able to get fairly detailed reports of the numbers of customers each day as well as the immediate count so I can see how to adjust my store accordingly at different times of the day.
2) I only need analysis and the immediate count so that I don't exceed safety limits for the number of people in my store.
3) I have the same entrance and exit.

Society Organizer:

1) I would like to be able to see the immediate count, and possibly the maximum count so I can see how the numbers change from week to week.
2) Really only limited analysis of the count and the count is needed so not many are required.
3) I am usually in a room with one entrance and exit however sometimes I need to relocate so I might be in different rooms.

Analysis of responses:

Teacher – They wanted the most features but still this was not too high and is therefore doable. They also used different room types so this will have to be considered.

Shop owner – they wanted analysis of the data and the immediate number but nothing apart from this so a few features would be sufficient. Also, they used one type of room, but the entrance and exit were combined so this would have to be considered.

Society Organizer – They required also only analysis and the count so these would be essential. The room types also varied for this person.

Overall – The features that were mentioned first and foremost included the immediate count and then analysis and the live video output. So, the most important component was the count and displaying this accurately. I would have to come up with a method to count people entering and exiting so that the total count could be shown. As opposed to just counting how many people are in the view of the camera as this would not be effective. Another feature that would be conducive to addressing the responses of the users would be incorporating some form of analysis of the data given, even if it is simple, this seems to be something that the stakeholders are looking for.

Essential features:

1 - A model created through deep learning which can detect people in a live video and count them and return a count:

This is needed as without out it; the count can't be obtained. Also, deep learning is the chosen method as it can effectively learn the features of an input which in this case would be the pictures of the people and can efficiently learn where the people are, making effective use of given resources. The effectiveness of the model also depends on the input it is given, I.e., if the input is diverse and varied and there is a large amount of the input then the trained model will perform well. This model can be transferred to another device and can be used for inference once it is already trained which makes the solution portable and not require a desktop computer.

2 - A website/app which can communicate with the device conducting the inference to retrieve the count at regular intervals and update it so that the user can see it:

Either the website or app would be used to act as the front-end which the user can interact with to make use of the model and communicate with the different hardware including the device which is connected to a camera.

3 - Controls on the website that can be used to start and stop inference:

These would be simple and allow the user to easily control the hardware without any unnecessary complexity. They are essential for the user to able to have control over the hardware and start and stop counting the people when they want.

4 – Analysis of the count at time intervals in some form:

This would be part of addressing the stakeholders' interests and would be helpful to them so it should be implemented. It provides a component of analysis of the detections which would certainly be helpful for people like the business owner as this could give an indication of the count at different times and so show when to increase stock.

Also, if the teacher was able to save the count, they could use this as a form of a register count.

Research

Sensor Types Research

When looking at solutions, I came across a variety of different approaches which each solved the problems in different ways using different sensors and provided certain benefits and drawbacks:

People Counting Sensor	Applications, Pros, Cons
Video Camera Sensor	Pro: Gathers precise information on the movement of people within its field of view allowing for a wide range of data collection. Tracks entry and exit data in real-time along with all other engagement metrics such as walk-in data, dwell time, hotspots, and path maps. For most applications, existing security cameras can function as the people counting sensors reducing the cost and complexity of implementation.

	Con: For certain edge cases, existing security cameras may not be able to recognize images. For example, high contrast areas, or areas with poor ambient lighting. In such cases, expensive 3D Cameras may be needed to plug the gaps in coverage. Best use: Any business looking to analyze foot traffic and activity for advanced analytics requirements where people tracking accuracy is critical.
Thermal Sensor	Pro: Low-medium cost, easy installation. Adaptable to complex entrances with multi-directional people movement. Does not capture any personally identifiable information. Works well in darkness, bright areas as well as in places with reflective surfaces or walls. Sensors are often discrete and can track a wide area, translating into fewer sensors. Con: Doesn't allow for in-depth analytics of customers and their behaviors such as dwell times or path maps. Sensors placed outdoors or in harsh environmental situations will deteriorate faster, requiring replacement more often. Best use: Any business looking to analyze foot traffic, maintain capacity limits, or those wanting to maintain the privacy of patrons and customers, such as healthcare clinics and related services.
Infrared Sensor	Pro: Accurate even for large and fast-moving groups of people. Relatively easy installation with one-time calibration or verification.

	No personally identifiable information is captured. Can work in a wide range of lighting conditions, unaffected by shadows and busy patterns on the floors. Con: Requires optimal placement to be effective – ideally above the area to be tracked. Best use: Locations where accuracy matters, such as venues with local capacity limit laws, shopping centers, libraries, museums, parks, and outdoor leisure centers.
Wi-Fi Beacon Sensor	Pro: Tracks customer activity based on their identity; hence it is suited for targeted communications and promotions based on where the customer may be located. Low cost of implementation as fewer Wi-Fi sensor devices are needed. Con: Accuracy is tied to customer mobile device capabilities and settings. Cannot be used for tracking product interactions as position accuracy is limited. Reliance on working Internet connection in the location. Privacy issues as tracking is completely based on customer identity. Best use: Large retailers, shopping malls, logistics warehouses, and other vast spaces where people count is needed and location inaccuracy is not an issue, and Wi-Fi-enabled devices are generally available with most individuals.
Bluetooth Beacon Sensor	Pro: Reasonably priced. Easily scalable for larger locations. Easy self-installation, and battery powered.

	Can push customized messages to customer smartphones. Con: Accuracy depends on customer mobile device capabilities and settings. Technology based on personally identifiable information. Accuracy in tracking the precise location of people is limited. Best use: Locations with a high probability of Bluetooth-enabled devices, such as shopping malls or other retailers.

After looking at all the different types of solutions, I came to the conclusion that the best approach for my specific project would be the use of a video camera as a sensor as I could implement people counting relatively easily with deep learning and since the project is mainly software based, I would be able to use mainly software to carry out the task rather than relying on hardware such as infrared cameras or thermal sensors although, these may have been more accurate, they would have been more complicated to work with in terms of software and the hardware would be expensive during development which Is why using a video camera would be the optimal solution. The camera itself wouldn't be a significant cost and the hardware used to develop it wouldn't need to be extremely specialized.

Existing solutions

After considering the different types of sensors I could use, I looked at solutions that specifically used a video camera and were already on the market.

Irisys SafeCount

After considering the different types of sensors I could use, I looked at a solution that specifically used a video camera and was already on the market.

The solution which I came across was the Irisys SafeCount sensor which seemed highly functional with a range of features that made it effective in tackling the problem.

The analysis seemed very well implemented so I can take points from it such as storing the data concisely however it seemed quite significant so I don't think I would implement this much analysis as this is not the main focus.

The main drawback was the pricing which was exceedingly expensive and the cloud subscription for the analysis would also be very expensive which is something I can improve with a smaller version but much cheaper and more affordable which is more realistic for the stakeholders which I have discussed.

Also, I didn't consider the system as a whole to be lightweight and the detections and counts weren't the main focus which my project will have as the main focus.

However, this solution did give me an idea as to what I was aiming for and gave me an understanding of what would be commercially viable.

Terabee People Counting

Another people counting system which I came across was the Terabee people counting system which has a similar functionality which I want to achieve in my solution, installing a camera connected to a hardware device which would carry out the real time detections and return it to an interface.

The main similarity was that it could be used in entrances/exits which is something that I want my solution to be able to achieve.

In addition to this, it had quite a complex analysis software implemented which looked like something that would be useful for this kind of project, however, this level of depth in analysis is not the focus of my project. But ideally, I can utilize some of the ideas in the analysis such as storing the counts at different times and saving them so that the user can take view them, which would still benefit the user without an overly complicated analysis.

Spectrum Series 3D People Counter

Spectrum Series

Overhead 3D People Counter

The Spectrum Series of People Counters use 3D imaging technology and analytics to provide highly accurate pedestrian traffic data for retail and public spaces. Use the information from these sensors to help plan your operations more efficiently, optimize resources and increase revenue.

Another people counting system which I found was the Spectrum Series 3D people counter which had additional functionalities over the other products, namely being able to distinguish between different types of objects and being able to tell adults and children apart as well as people of different heights which is something that would be beneficial in particular to business when considering the demographic of people who use their stores.

Moreover, it was made to be robust, saving detections in case of power loss, being able to be used remotely, among other features.

This seemed like the most functional product, and it gives me an idea as to what I am aiming for in terms of creating a solution that is viable, particularly the idea of saving detections is something I want to implement in my solution.

Requirements

End User Requirements

Hardware Requirements:

- **Jetson Nano** – Required to be able to run the inference, this is the hardware that will provide the necessary processing power to run the code as running the model requires a certain amount of processing power. If a lower spec device is used, it may not run as fast, and the detections may be less responsive.
- **External Camera** – This is needed so that it can be connected to the Jetson Nano and the inference can be carried out, using the camera to pass a feed so that the software can carry out the detections using the developed deep learning model.
- **Wi-Fi/Ethernet Connection** – This is needed so that any devices that the user wishes to view detections on via the website can connect.

Software Requirements:

- **Up-To-Date OS** - This is needed on the Jetson Nano so that the relevant libraries can be installed as otherwise there may be some compatibility issues.
- **TensorFlow GPU** – This needs to be installed so that the user takes advantage of the GPU in the Jetson Nano so that the detections run quickly.
- **Libraries** – Other libraries part from TensorFlow GPU will be needed, including OpenCV for image processing and Dlib for object tracking.
- **Python** - This will be needed to run the code as I will be developing the solution using python, specifically a python 3.7.x version would be preferable as this will minimize the chances of any compatibility issues between libraries.

Limitations

- One possible limitation is that the hardware required for inference (Jetson Nano) would have to be purchased as well as a camera. This however would be relatively cheap compared to other solutions so it wouldn't be too much of an issue in terms of developing the solution.
- Another possible limitation is that Wi-Fi may have to be available to be able to transfer the data from the camera to the application or website. However, this can be overcome using Bluetooth if the devices are close.
- Another limitation could be that training the mode to be able to detect and count the number of people would require a GPU with enough power so that it wouldn't

take too long. However, this can also be overcome by using free GPUs provided by google, on google Collaboratory which would speed up the process of training the model.

- It may be difficult to find large amounts of data to train the model, this can be overcome using prebuilt models or using existing, already formatted training data.
- Another limitation in terms of the model itself would be that the model would be designed to detect people as opposed to identifying them so would be limit in terms of capability, but this shouldn't affect the program as it will consider if people are entering or exiting the room so even if the same person walks in and out, it wouldn't affect the program.

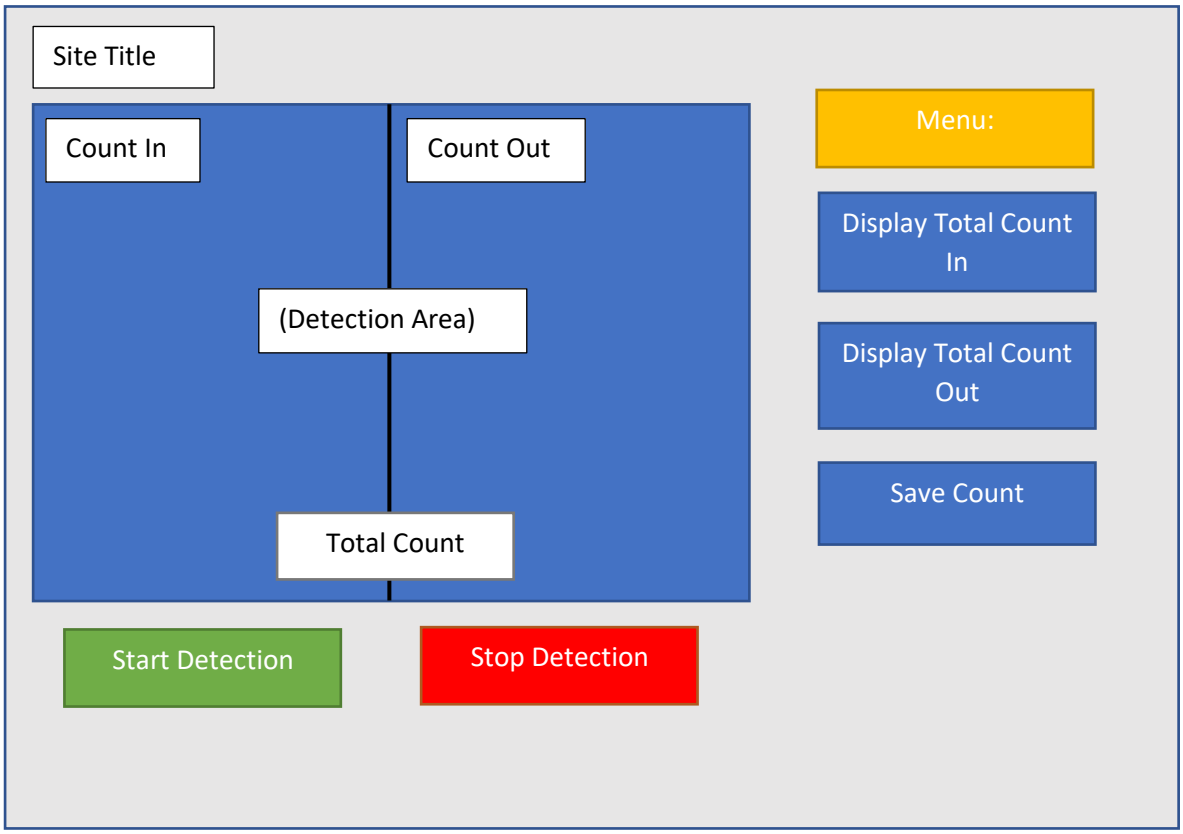
Success Criteria

Criteria	**How it can be evidenced**
1 - Is the correct number of people in a room at any given time given as the minimum output?	Using Realtime testing of people walking from one side of the camera and checking that the correct count is recorded at the end.
2 – Does the website or app give the user the option to toggle different features?	Different buttons that function correctly are shown with the page before and after being clicked shown so that the button can be shown as functioning.
3 - Is the website/app simple and lightweight so the user can easily use it?	User feedback after final stage, a simple interface that functions correctly should be shown with screenshots.
4 - Does the program correctly identify if people are coming into and out of a room?	Both the left and right/up and down counts give correct numbers, screenshots should be provided over a sequence of people passing the camera. Both counts should be correct so that people going in and out are correctly identified.
5 – If the user toggles the live camera feed output with bounding boxes around detected people, is this displayed clearly and correctly?	Screenshots of the start detection button before and after is clicked should be included.
6 - Is there an information menu to direct the user to be able to toggle the different features and allow them to easily use the software with a user-friendly graphical interface?	There should be buttons on the page with screenshots of before and after being clicked.
7 – Is the analysis effective and does it cover all the user's needs?	There should be screenshots of output along with the analysis for a given time of the program running with a certain of number of people passing each side.

Design Phase

UI Design

The solution will involve a UI to be displayed on the front-end web page. I will be using the flask framework to be able to do this. This is how the web page will look:



The real-time detections will be displayed in the center of the screen (blue box) with a count of people that have moved into a room and out of a room, determined by a line on the screen and detected people passing the line.

Total Count - This will be the main component of the UI, displaying the count of the people in the room.

Detection Area - This will be the area which will be used to display the camera feed. The feed will be passed to this as an image which is constantly updated, being given frames from the camera which are processed by the program running the model before being passed back to the webpage via flask.

The different elements on the page apart from the Detection Area will be the following:

- Count In
- Count Out

- Total Count
- Site Title
- Start Detection
- Display Total Count In
- Display Total Count Out

Start Detection:
This is a key button that will allow the client to start the detection, it will display the image with the current settings

Menu:
- Display Total Count In – This is a button which will set the total Count In variable to be shown on the image
- Display Total Count Out – This is a button which will set the total Count In variable to be shown on the image

Count In:
This will show a count of the number of people that have passed the line from the right to left (or left to right depending on the configuration) to give the number of people that have come into the room.

Count Out:
This will show a count of the number of people going the opposite way to count in, and therefore will give a count of the number of people that have left the room.

Total Count:
This will calculate the total number of people in the room by taking away the Count Out from Count In. This will easily allow the client to view the current count at any point in time.

Site Title:
This is simply for display and won't contribute to the functioning of the web app, but it will make the web page display relevant and more appealing.

Stakeholder Input

After designing the main front – end, I decided to get stakeholder input.

I asked the following questions:

1) Does the following design fit your needs? (Image of above design given)
2) Is the design too complex or too simple?
3) Are there any further comments you would like to make on the design?

Responses:

Teacher:

1) The design does facilitate me to easily see the count of students in the class however it doesn't quickly show if a student is late, though I can check the count when the bell goes so this isn't a big deal
2) There are a lot of buttons for me, but this isn't a problem
3) Overall, the design is sufficient and works.

Shop Owner:

1) Yes, it allows me to save the current count which is helpful
2) It could be more complex, but it does what is needed
3) Nothing further to add

Society Organizer:

1) It does exactly what I need
2) It achieves what I want so it isn't too complex or too simple
3) It might be helpful to have the maximum count clearly

Analysis of responses:

Overall, the design seems to fit the client's needs and adding more could make the design clunky and too many buttons may not be appealing. As for the society organizer wanting the maximum count clearly, this can be seen when saving the count at the end of the detection, so saving the count in a clear format will be helpful for the society organizer but this is mainly already addressed. I believe adding more would make the program stray from being lightweight and this could make it harder to run as well so keeping it light with what is needed is what I am planning to do.

Usability Features

I will need to make the interface simple so that users can easily use the web app without needing any special knowledge about what is going on or how to do certain things. This will be done using large buttons so that it is clear to the user what they can do, especially in terms of the menu, since some users may want only some of the items in the menu, so having a large menu with lots of options may take away from the ease of use.

The largest area will be the detection area, containing the live feed with detections appearing on the screen. This will validate to the user that the web app is functioning. The next largest component should be the Total Count as this is the core data, which is required by all users, I can also make it stand out with a bright color such as yellow, so it is clear over the live feed.

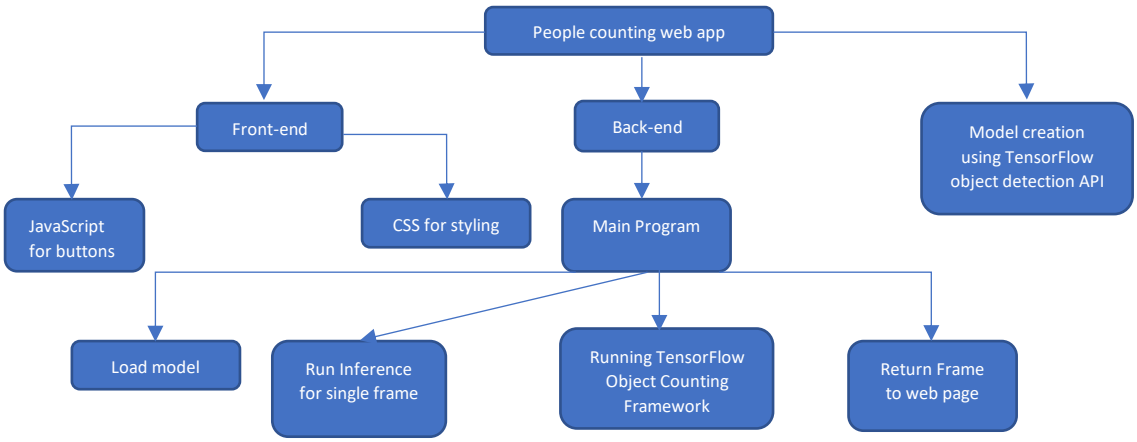
The start and stop detection buttons are very important and must be clear to the user as they are again core parts of the program as they start and stop the detections so must be clearly visible to the user. For this reason, they should be colored as green and red accordingly as shown in the diagram, to make it as user friendly as possible.

The menu does need to be clearly highlighted however the buttons under the menu do not need to be as they could clutter the display. For this reason, the buttons should be relatively small, but not too small to not be seen by the user.

Overall, the usability of the web app should be straightforward as it involves few buttons, but the styling of the page can be adjusted slightly to make it more appealing and friendly to the user.

Problem Decomposition & Structure

Main Decomposition



The task of people counting through a web app can be split into 3 main areas first, as shown in the diagram; Front-end, Back-end and Model Creation which will feed into the back end.

I have chosen the front-end as an area to focus on as this is how as user would utilise the application, and therefore needs to be presented well with all the correct elements, to fulfil their needs. Since I am making a web app, this is an integral part of the problem and needs to be developed correctly so that it can interact with the back end.

Next, I have chosen the back end as an area to focus on as processing that needs to take place is significant, so a back-end is essential to be able to do this correctly as opposed to do all the processing on the front-end (client-side processing would be very slow as the hardware may not be powerful enough and would take up a lot of space). Specifically, I am using a Jetson Nano to carry out the heavy processing due to it having a GPU as well as being light and portable so it can be integrated into environments easily. This will pass the necessary information to the front-end.

Finally, I have chosen model creation as the last main component of the problem. The nature of the problem requires object detection (specifically people detection) to then allow for people counting. There are various methods to do this. For example, there are traditional object detection methods such as using the Viola Jones Object Detection Framework. After research, I found that this kind of framework would not be useful as it is

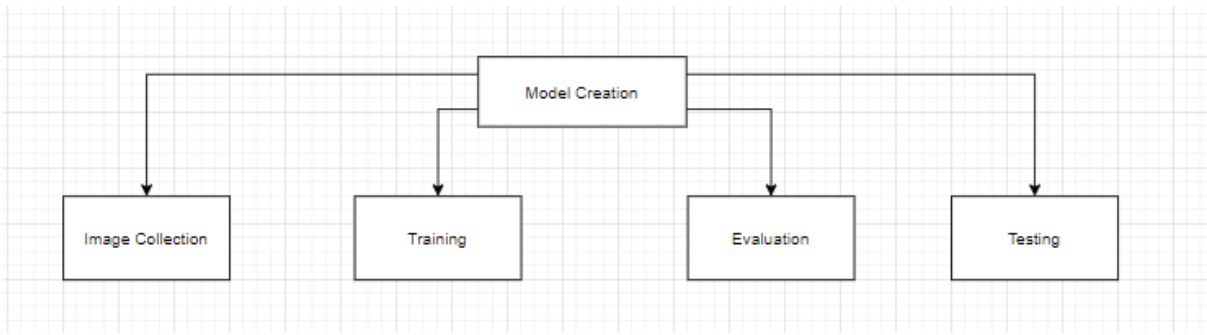
particularly good at detecting faces, but other objects are more difficult and, in this case, people detection is required which is much more difficult compared to face detection as the kinds of images that contain people would be much more varied and matching specific features to them would be difficult. Due to this, I was not keen on using any traditional object detection models due to their specificity to certain cases of object detection and the fact that they aren't very robust. I decided on using deep learning models as they provide the perfect solution to people detection and object detection in general, being able to learn the features automatically after many rounds of training. Due to their high accuracy and quick detection rate, they are highly suited to the problem, and therefore they are part of the solution.

With these three main components coming together, I will be able to create a web app that solves the problem.

Looking further into the decomposition, the front-end is split into HTML, CSS, and JavaScript. These are the building blocks of the webpage. The most important will be the HTML defining the basic structure of the webpage. Next important is the JavaScript as it will be needed when creating buttons for the user to click to pass information to the back end such as specific settings. The CSS is the least important but is required to appeal to the user and make sure that the web page is presentable, however the webpage does not need to be heavily styled as the aim is to create a lightweight solution and the most important thing is that it achieves the goal of people counting.

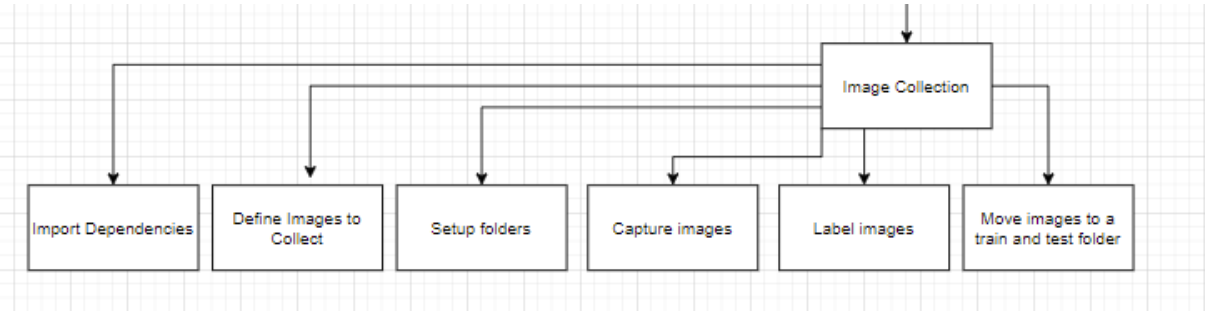
The back end is split into loading the model, running inference on the frame, running TensorFlow Object Counting Framework, and returning the frame to the front-end. Loading the model is the first step, the model is required to carry out detections and therefore needs to be loaded correctly in the proper format so that it can be utilised. Next, running inference on the frame, this is where the actual detections happen, and the coordinates of resulting detections are gained. The TensorFlow Object Counting Framework will be used to carry out both counting as well as tracking and drawing the elements onto the frame, some of which will need to be added and edited to fit the problem, such as calculating a total count of people in a room, given the count of people that have walked in and the count of people that have walked out. Finally, the frame will be sent back to the front-end so that it can be displayed to the user via the webpage.

Model creation is the last main component as shown in the diagram. This component is complex, so I have created a separate decomposition for it:



This is the main section and can be split into four main parts, image collection, training, evaluation, and Testing. These will be repeated in an iterative process till the performance of the model is acceptable.

Model Creation (Image Collection)



1 – Importing Dependencies

There are a few libraries which will be required to be able to collect the images, these will include OpenCV, uuid, os and time.
OpenCV will be used to be able to access the webcam and carry out operations related to the webcam, including taking a picture and saving it to a file.
Uuid will be involved in creating unique names for the images when saving them, so no issues arise with images overwriting other images.
OS will be used when dealing with paths and writing to paths.
Time will be used to help with collecting the images.

2 – Define Images to Collect

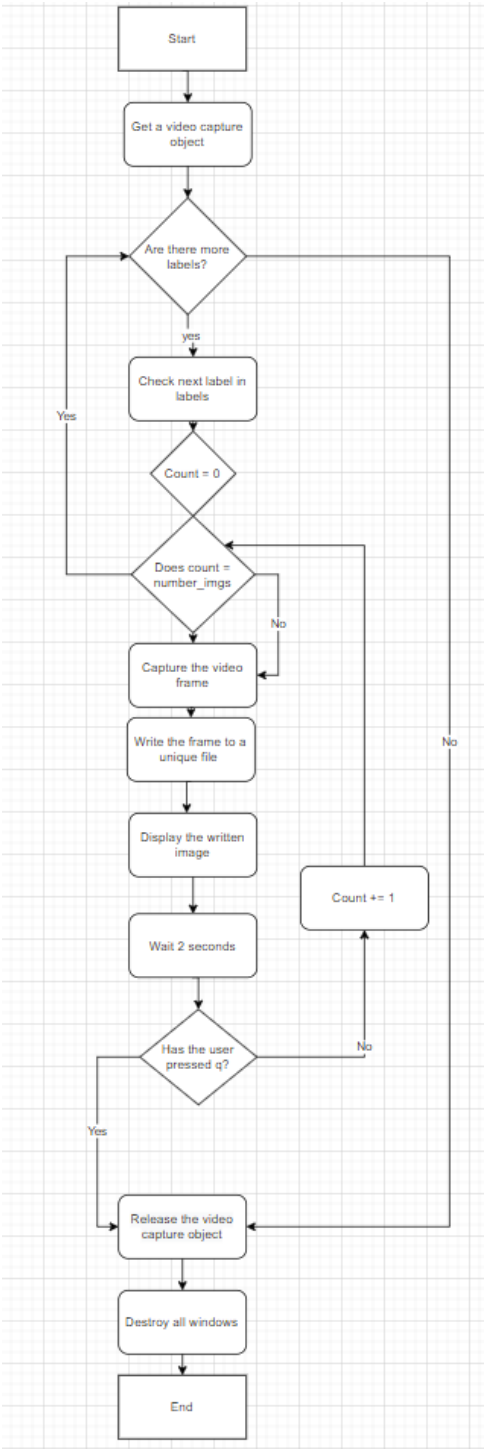
Here I will create the labels which will be used. This will include only one label since I am carrying out object detection for a single object (people). If I was to detect more than one type of object, I would add more labels.

3 – Setup Folders

This is where the OS library comes into play, creating the path where the images will be stored.

4 – Capture Images

OpenCV will now be used to collect the images, it is generalized so if there were multiple labels, it can handle them, this would be useful if a client wanted to detect multiple objects such as detecting dogs as well as people.
Flow Chart:



5 – Label Images

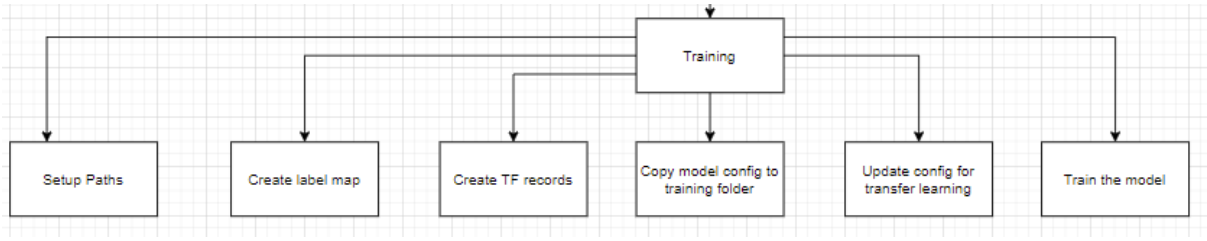
For this, rather than code, I will be using the Labelimg application which creates a .xml file with annotations that indicate the position of the object in the image (the person), as well as the name of the label (person).
This is an example of a .xml file that will be created that will be paired with the corresponding collected image:

```xml
<annotation>
    <folder>label_folder</folder>
    <filename>unique_file_name.jpg</filename>
    <path>path_to_image_here</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>width</width>
        <height>width</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>label_name</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>116</xmin>
            <ymin>193</ymin>
            <xmax>203</xmax>
            <ymax>349</ymax>
        </bndbox>
    </object>
</annotation>
```

6 – Move Images to a train and test folder

This section is self-explanatory, however the number that will be moved to a train folder and the number that will be moved to a test folder will depend on the number of images collected. Common ratios of images used for testing and training are: 70% train, 30% test. 80% train, 20% test. 60% train, 40% test. Most of the images will be used for training, so I am likely to use an 80% train, 20% test ratio.

Model Creation (Training)



This section will involve using a pre-trained model and taking advantage of transfer learning to create a model that has a high enough performance that it works well with smaller amounts of data which is better than having to collect lots of data beforehand.

1 – Setup Paths

In this section, I'm defining variable names which will be useful later, particularly when working with directories and saving files.
Since I will be taking advantage of transfer learning, I would specify the URL of the pretrained model from the TensorFlow Model Zoo, which contains a range of different models, some with higher MAP (mean average precision) and some with higher speeds. The

one I have chosen, the SSD MobileNet V2 FPNLite 320x320, has a speed of 22ms and a MAP of 22.2. This gives a good tradeoff between accuracy and speed, but this architecture can be changed later when performance tuning if needed.

2 – Create Label Map

This section involves creating a file that contains the different labels and assigns them ids. This will be used when training, making it easy to identify the labels. It is also used during inference.
This is how it will look like:

```
item {
    name:'Person'
    id:1
}
```

3 – Create TF records

TensorFlow Records are the format in which the test images and train image will be stored so that they can be used when training and conducting inference.

4 – Copy model config to training folder

Again, this section is self-explanatory, the pretrained model config is copied to the new checkpoint path so that it can be used and built on when training.

5 – Update config for transfer learning

This section also involves setting up the paths correctly in the new config so that it is ready for training.

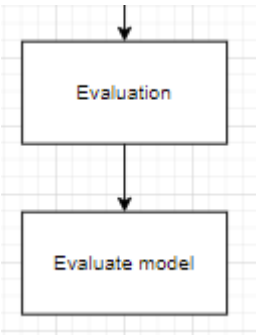
6 – Train the model

Now that everything is setup, this section will use a command to start the training. Hyperparameters such as number of epochs can be configured here which are particularly important as they determine the performance of the model.

Model Creation (Evaluating)

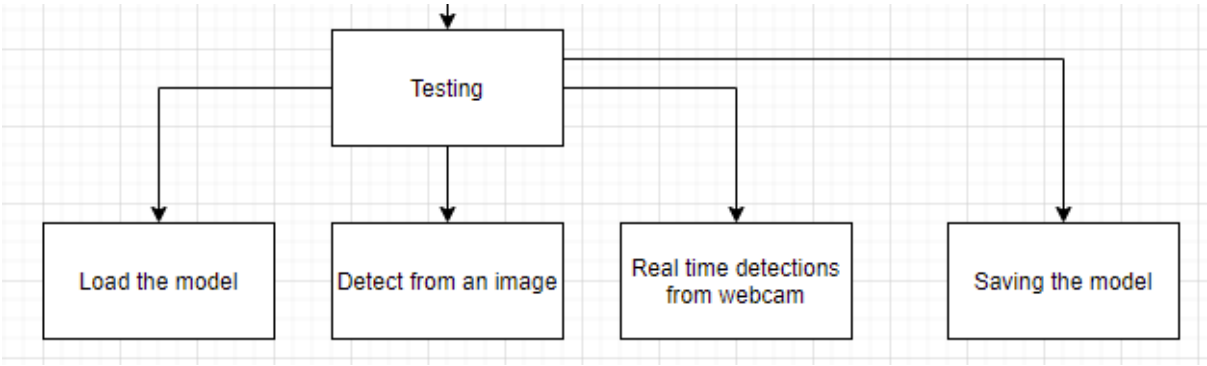
Evaluating the model will involve finding specific metrics that measure how well the model performs against test data, i.e., how well it performs at detecting images as well as to what accuracy.

1 – Evaluate the model



A command is also used to provide metrics such as loss and precision to see how well the model performs on the test data. This will use the initial validation data that was split so that the model can be tested for the specified metrics. This will give an indication of how well the model is performing and whether the model needs to be refined.

Model Creation (Testing)



Testing the model itself will mean checking the model loads properly, so it has been saved correctly as well as using it to detect from an image and to detect from a webcam and detecting from a webcam is the most important as it is the one that will be used in the final application.

1 – Loading the model

First the required modules must be imported:

Next, I need to load the config which will allow me to build a detection model, then I can restore the most recent checkpoint from training and finally create a function to carry out the detections themselves, given an image which would be in the form of a tensor array. This function will be key when carrying out detections, both for a single image and for real-time object detection.

2 – Detect from an image

The modules required for this section need to be imported first, then the main detection can be carried out.

3 – Realtime detections from webcam

OpenCV will be utilized to capture webcam input and then detections can be carried out on each frame and the resulting bounding boxes can be drawn on the frame then displayed. This will allow for the model to be checked for accuracy and speed.

4 – Saving the model

The model needs to be saved as a .pb format so that it can be used in the main program with the TensorFlow Object Counting Framework.

Post-Creation Model Testing:

To test the built model itself, I will make use of the TensorFlow Object Detection APIs testing tools such as tensorboard to display things like loss metrics (how badly the model performs against training epochs). With this I can gain metrics such as mean average precision (MAP) and loss which I can use to gauge how well the model is performing. Of course, I need test data with which I can evaluate the model with, this would have already been partitioned earlier. I can also collect additional test images of people on the internet to test the model, making sure I have a wide range of images and styles so that the model is checked to be robust and able to handle different types of backgrounds.

After creating evaluation metrics for the model, I can re-train the model with additional images, and using tensorboard, I can see the images that the model performed poorly on and collect more images of that style.

I can repeat this process of evaluating the model and re-training it iteratively until the metrics for the model are reasonable and consistently above a certain threshold, I.e., an accuracy of 85%.

If retraining on images doesn't work, I can change the epoch number to train for longer, I can also change the model architecture if needed, with different models giving different accuracies paired with different speeds.

To test for the speed of the model, I can run the model via flask on a web page, to see if the speed of the detections is sufficient. If the speed is too slow, I may need to change the model architecture.

Algorithms for the back end

The algorithms required in the back end are required to feed the correct data to the front-end so that the web app functions correctly, to meet the users' needs. I will be utilizing the flask framework to make the connection between the front and back end. It will allow me to create a flask server which I can use to allow my webpage to be accessed on the same network. The back-end algorithms will be stored on the device running the inference and

the subroutines will be stored in the flask framework so that they are run when the web page

Generate Frames

The flask framework consists of a template folder which contains all the front-end related files such as the HTML file. The other component is the python files. To get a frame from the front-end, an image source is used which refers to a specific function in the python file. This returns a response to the web server (which is hosted on the network) and the response calls a generator function which takes webcam input and returns the frame after it is processed, this then goes back to the image element in the webpage. This is how the frame will be returned to the webpage.

The first algorithm that I will be using is for generating frames so that they can be displayed to the webpage. This allows the live feed to be displayed on the webpage for the user to see. I will be able to build on this and involve detections later on.

```
Function Generate Frames:

      While true

            Read frame from camera

            If frame not successfully read, then
                  Exit while loop

            Encode frame

      Yield frame
```

The subroutine takes a frame from the already captured input, exits the loop if the frame is not read correctly, encodes the frame (This is needed so that the frame can be displayed to the webpage) then finally the keyword yield is used for the frame as when using flask, this is needed so that the function is run again in a loop.

Detection on Single Image

The next crucial algorithm I will be using is for running detections on an image. This will allow me to run the detections on the frame before it is encoded and returned to the webpage so that the final image that is returned to the webpage is an annotated image with bounding boxes around detections.

It firstly converts the image (which would a frame from the webcam feed), into a numpy array, so that it can be converted to a tensor, and then a tensor with a new axis which is the format needed for the image to be processed by the model. Then inference can be run, giving the output to a dictionary. The dictionary is then sliced to give the correct detections, which are converted to integers then the dictionary is returned. This algorithm is very important as it retrieves the detections, using the model that has been created. This will then be later used when counting detections.

```
Function Run_Inference_for_Single_Image(Model, Image):

      Convert image to numpy array

      Convert numpy array to tensor

      Output_dict = Model(image)

      Set detections from output_dict to detections dictionary

      Return detections dictionary
```

Loading Model

This is a simple but important function that will take the model path as input and will load the model using a method in the TensorFlow module, then it will return this model in a format which can be used for inference.

```
Function Load Model ( Model Path ):

      Set model to tf.saved_mode.load( Model Path )

      Return model
```

Algorithm Used in TensorFlow Object Counting Framework

The parts of the framework that will be used consists of 3 files. The first is the tensorflow_cummulative_object_counting.py file. The core function Run_Inference is the main function that is used, and will be needed to run both the detections, tracking, and drawing of output on the frame. It makes use of object tracking with the Dlib library, using the 2 other files, centroidtracker.py and trackable_object.py. Both are needed when applying object tracking. The framework will be integrated into the main flask application and therefore will be amended so that it functions correctly. I chose to use this framework as it provides a simple way to carry out cumulative object counting as well as displaying necessary output such as identifying new objects in a frame.

This is a link to the github repository where the particular object counting framework is found:
https://github.com/TannerGilbert/Tensorflow-2-Object-Counting (accessed 20/07/21)

The following is the pseudocode for the Run_Inference Function in the function, separated with each part explained. I will be adapting it during the development phase so that it can be used with the flask framework and I will be removing unneded sections and adding sections to fit my project.

This first section creates the function, including a range of parameters. The first is the model which wil be the model that I create in development which will be responsible for carrrying out the detections. The next is the category_index which is created using the label_map for

the model, which I have described previously. The next is cap, this is the captured video feed. The next is labels, this will be just people since I am only detecting people. The next is roi_position, this is where the line that people must pass to be counted is located, it's value is a 0 for all the way to one side and 1 for all the way to the other side. The next value is threshold, this is the minimum confidence that the model has in the detections for them to be used. A higher value means that only higher accuracy detections are included whereas a lowever value will mean that less accurate detections will be included and these may not be people so this is set to a reasonable value of 0.8 here. The x_axis parameter is for the positioning of the line, it is vertical if x_axis is True and horizontal if x_axis is false. Skip_frames is the number of frames skipped between new detections being made. This increases performance as it reduces the load on the processors as fewer detections need to be made however, it still has the ability to perform well as people aren't going to be moving very quickly. The next is save_path, this is for saving the feed however this is something I am not planning on doing, rather I want to save the count so this is something I will change. The final parameter show indicates whether or not the feed is displayed.

The next line declares a list counter which holds the value of the count left,right, up and down so can be used whatever the value of the parameters are.

The total_frames variable is used because of the skip_frames parameter so that the detection can be carried out every time i.e. 20 frames have passed.

The next line declares a variable called ct which creates an object using a class in the separate centroid_tracker.py file in the framework. This is tracks the centre of the bounding boxes of detected people and is essential when tracking them.

Finally, a trackers list and trackableObjects dictionary are defined to record the current trackers used to track objects and the trackable objects that exist so that they can be kept track of.

```
Function Run_Inference( model, category_index, cap, labels,
roi_position=0.6, threshold=0.8, x_axis=True, skip_frames=20, save_path='',
show=True ):

    counter = [0, 0, 0, 0]  # count left, right, up, down

    total_frames = 0 # Records the number of frames since start

    ct = CentroidTracker(maxDisappeared=40, maxDistance=50)

    trackers = []

    trackableObjects = {}
```

The next section first does a check to see if the save_path parameter is true, then it records the width height fps then creates an object which can be used to save the video to the specified path later on.

```
    # Check if results should be saved
    if save_path:

        width = int(cap.get(3))
```

```
        height = int(cap.get(4))

        fps = cap.get(cv2.CAP_PROP_FPS)

        out = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(
            'M', 'J', 'P', 'G'), fps, (width, height))
```

This section starts the while loop in which the image will be have tracking and detection carried out if the number of skip frames are skipped. First, we are entering the loop checking that the video feed is available, then the frame is read and set to the image_np variable which will be the frame which will be edited and then displayed. Then we are getting information about the frame such as height width and color.

```
    while cap.isOpened():

        ret, image_np = cap.read()

        if not ret:
            break

        height, width, _ = image_np.shape

        rgb = cv2.cvtColor(image_np, cv2.COLOR_BGR2RGB)
```

Now initially the status is put as waiting and initalise a rects list to store the rectangles which would be the boudning boxes of the detections.

```
        status = "Waiting"

        rects = []
```

The next section is the detection section where the required skip frames are skipped, checked using total_frames modules skip_frames == 0 and after this, the previous detection for a single image function is run and we update the trackers accordingly.

```
        if total_frames % skip_frames == 0:

            status = "Detecting"
            trackers = []

            # Actual detection.
            output_dict = run_inference_for_single_image(model, image_np)

            for i, (y_min, x_min, y_max, x_max) in
enumerate(output_dict['detection_boxes']):

                if output_dict['detection_scores'][i] > threshold and
(labels == None or
category_index[output_dict['detection_classes'][i]]['name'] in labels):

                    tracker = dlib.correlation_tracker()

                    rect = dlib.rectangle(
                        int(x_min * width), int(y_min * height), int(x_max
* width), int(y_max * height))
```

```
                    tracker.start_track(rgb, rect)

                    trackers.append(tracker)
```

Now if the skip frames have not been met, the tracking section is run where the trackers are updated, and centroids (the centre of the bounding boxes) are calculatted and as well as this, the direction of the objects is found. If the objects pass the line then count is also updated in this section.

```
        else:

            status = "Tracking"

            for tracker in trackers:

                # update the tracker and grab the updated position

                tracker.update(rgb)

                pos = tracker.get_position()

                # unpack the position object

                x_min, y_min, x_max, y_max =
int(pos.left()),int(pos.top()), int(pos.right()), int(pos.bottom())

                # add the bounding box coordinates to the rectangles list
                rects.append((x_min, y_min, x_max, y_max))

        objects = ct.update(rects)

        for (objectID, centroid) in objects.items():

            to = trackableObjects.get(objectID, None)

            if to is None:

                to = TrackableObject(objectID, centroid)

            else:

                if x_axis and not to.counted:

                    x = [c[0] for c in to.centroids]
                    direction = centroid[0] - np.mean(x)

                    if centroid[0] > roi_position*width and direction > 0
and np.mean(x) < args.roi_position*width:

                        counter[1] += 1
                        to.counted = True

                    elif centroid[0] < roi_position*width and direction < 0
and np.mean(x) > args.roi_position*width:

                        counter[0] += 1
                        to.counted = True

                elif not x_axis and not to.counted:
```

```
                    y = [c[1] for c in to.centroids]
                    direction = centroid[1] - np.mean(y)

                    if centroid[1] > roi_position*height and direction > 0
and np.mean(y) < args.roi_position*height:

                        counter[3] += 1
                        to.counted = True

                    elif centroid[1] < roi_position*height and direction <
0 and np.mean(y) > args.roi_position*height:

                        counter[2] += 1
                        to.counted = True

                to.centroids.append(centroid)

            trackableObjects[objectID] = to
```

This final section is where all the results to be ouptut are set up and if the show variable is true, then they are displayed.

```
            text = "ID {}".format(objectID)
            cv2.putText(image_np, text, (centroid[0] - 10, centroid[1] -
10),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)

            cv2.circle(image_np, (centroid[0], centroid[1]), 4, (255, 255,
255), -1)

        # Draw ROI line
        if x_axis:
            cv2.line(image_np, (int(roi_position*width), 0),
(int(roi_position*width), height), (0xFF, 0, 0), 5)

        else:

            cv2.line(image_np, (0, int(roi_position*height)), width,
int(roi_position*height)), (0xFF, 0, 0), 5)

        # display count and status
        font = cv2.FONT_HERSHEY_SIMPLEX

        if x_axis:
            cv2.putText(image_np, f'Left: {counter[0]}; Right:
{counter[1]}', (10, 35), font, 0.8, (0, 0xFF, 0xFF), 2,
cv2.FONT_HERSHEY_SIMPLEX)

        else:
            cv2.putText(image_np, f'Up: {counter[2]}; Down: {counter[3]}',
(10, 35), font, 0.8, (0, 0xFF, 0xFF), 2, cv2.FONT_HERSHEY_SIMPLEX)

        cv2.putText(image_np, 'Status: ' + status, (10, 70), font, 0.8, (0,
0xFF, 0xFF), 2, cv2.FONT_HERSHEY_SIMPLEX)

        if show:

            cv2.imshow('cumulative_object_counting', image_np)

            if cv2.waitKey(25) & 0xFF == ord('q'):
                break
```

```
        if save_path:
            out.write(image_np)
        total_frames += 1
    cap.release()
    if save_path:
        out.release()
    cv2.destroyAllWindows()
```

These algorithms show how the logic and basic structure of the solution will be developed as they show how the detections themselves will be carried along with how they will be used to return a count to the user which is the basic functionality of the program. They also describe how the back-end will pass information to the front, using the flask framework. The front-end components will be developed to work with these algorithms, so they aren't as significant. The algorithms that form the backbone of the solution are described here and they will be highly important when I come to actually coding the solution.

Testing Algorithms and Validation

Testing the load model function is straightforward, and it only needs to be checked for loading the model correctly. This can be done by using the model after. For validation, the model path can be checked to be correct, if not an appropriate error message can be given so that during development, it is easy to identify if an incorrect model path is given.

Testing running inference for a single frame needs to be done using test images which would have been created earlier, or data from the internet can be taken of images with people and without people, the output dictionary should have very low scores for no people and have high scores for images with people. Using trace tables in this kind of testing situation wouldn't be appropriate due to the use of models. Much of the model testing would have been done earlier as well when creating the model so this essentially just needs to be checked to be giving the correct output, provided with a correct model and image. However, In terms of validation, the frame can be checked, if the frame is empty then this could cause issues however, this is normally checked for when opening the webcam so this would already be checked in the generate frames function.

Since the run inference function is already part of the TensorFlow Object Counting Framework, it is already known to be working correctly so it can be safely implemented, however when adapting I to allow for it to be run alongside the flask framework in a webpage, this will need to be taken into account, therefore there will be testing during the development stage that addresses this specifically. It will be in the form of testing the program as a whole and checking whether it gives the desired webcam feed in the browser with the correct detections or not.

Testing returning the frame to the front-end would involve simply passing live feed to the front-end without and image processing and therefore would only need to be checked initially. Issues could arise if the frame is being passed to slowly and image processing is taking too long however, this can be fixed through improving the speed of the model by changing the architecture and other hyperparameters.

Inputs and Outputs

These will be the main inputs, with their corresponding processes and outputs.

Input	Process	Output
Frame from webcam	Object Counting Framework algorithms	Frame fully annotated with user's settings given
Menu buttons for settings	Will return a Boolean variable to the main program for each button whether to i.e., show a counter or save a count	Confirmation that a setting has been activated through text displayed after the button is clicked/Action of button carried out (i.e., count shown)
Start/Stop button	Start/stop the video by calling the video function which will generate the frames or terminate this function.	The frames from the webcam will be displayed or removed

Key Variables

These variables will be the most important, allowing the correct output to be given.

Name	Type	Use
output_dict	Dictionary	Contains the locations of detections in a frame so that they can be drawn on and used for the count
counter	List	To store the cumulative count of people that have passed to certain parts of the screen, it is in the format: [left, right, up, down], only two of these indexes will be used depending on the orientation of the camera and the setup
Trackers	List	Contains the current trackers used from the Dlib library, used for each object (person) in the frame
Detection_model	An object from the TensorFlow model class	Used to carry out detections, using its methods
Category_index	Category index object from label map util class	Used to detect the correct object (people), created from the label map
Display_total_count_in	Boolean	Used to check if the user has pressed the button to display total count in or not

Display_total_count_out	Boolean	Used to check if the user has pressed the button to display total count out or not
Save_count	Boolean	Used to
Camera	Object from OpenCV	Stores the camera feed
Image_np	Numpy array	Stores the frame image as a numpy array so that it can be processed
Total_frames	Integer	Stores the total frames since the webcam feed has run. Needed for carrying out detections when some frames are skipped

Testing Checklist

Component to test	Working?
Model - does it give an accuracy of above 85% for correct detections?	
Model - does it give an appropriate speed for detections in real-time with the webcam?	
Web page – does it provide all the necessary elements?	
Does the save count button work (is the correct count saved to file)?	
Does the show count in button work?	
Does the show count out button work?	
Does the start button work?	
Does the stop button work?	
Is the webcam correctly displayed to the webpage?	
Is the correct count given at any time?	

After all the components are tested in the checklist, the program will be completed.

Development & Testing Phase

Stage 1 - Basic Flask App setup

Code, Explanation and Justification

This initial phase will create a basic webpage using the Flask Framework and, as a placeholder, I will be using haarcascades to conduct face detection which will act as the 'frame processing' for now. Later in development, I will be able to replace this with a functioning deep learning model with a good accuracy that can carry out detection of people in the frame.

The following code will be the starting point for creating a webpage in flask and it will be loading a html file called index.html which is also shown below. The HTML is already setup for adding a video to output on the page which is represented by an image source that will be given to the page via a function called video later.

First, I am creating a starting point for the project by creating a webpage in flask which will load a html file called index.html.

```
1   # Importing the necessary libraries including flask and openCV
2   from flask import Flask, render_template, Response
3   import cv2
4
5   # Creating the flask app
6   app = Flask(__name__)
7
8   # Creating variable a camera which will store input from the webcam
9   camera = cv2.VideoCapture(0)
10
11  # Using a decorator and a function to render the html page located in the template folder
12  @app.route('/')
13  def index():
14      return render_template('index.html')
55  # The flask app itself is run in the following section with the host ip address and port specified
56  if __name__ == '__main__':
57      app.run(host='127.0.0.1', debug=False, port=9999)
```

I am setting the HTML up to allow for a video to be output through the page, using an image source that will be given to the page via a function in the flask framework called video. I will develop the video function after creating some other functions first.

templates > index.html > ...

```
<!DOCTYPE html>
<html>
  <body>
    <h1>Live streaming</h1>
    <div>
        <img src="{{ url_for('video') }}" width="50%"/>
    </div>
    </body>
</html>
```

The next function I will create is the generate_frames() function which is where the majority of the frame processing will occur and here I am taking advantage of the OpenCV library to read frames from the earlier declared camera variable and then move onto processing the frame by adding bounding boxes and a count which again will act as a placeholder for the deep learning model however OpenCV will still be needed as it is very useful when handling images so will be used in conjunction with the model to produce output.

```
# A function which takes the global camera variable and uses it to return a modified frame with a bounding box around faces face
def generate_frames():
    while True:

        # read a frame from the camera
        success,frame=camera.read()

        # Check is made whether the camera input was successfully read or not
        if not success:
            break
        else:

            # The following section uses haarcascades to detect faces and count them but this will be replaced by a deep learning model later on

            face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
            # Read the input image
            # Convert into grayscale
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            # Detect faces
            faces = face_cascade.detectMultiScale(gray, 1.1, 9)
            count = str(len(faces))
            font = cv2.FONT_HERSHEY_SIMPLEX
            cv2.putText(frame, count, (10,450), font, 3, (0, 255, 0), 2, cv2.LINE_AA)
            # Draw rectangle around the faces
            for (x, y, w, h) in faces:
                cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)

            # The resulting frame is encoded and converted to bytes
            ret,buffer=cv2.imencode('.jpg',frame)
            frame=buffer.tobytes()

        # Yield is used so that the generator function can be continually called
        yield(b'--frame\r\n'b'Content-Type: frame/jpeg\r\n\r\n' + frame + b'\r\n')
```

Now I will be developing the video function which is a simple function that calls the generator function so that the image in the HTML page is constantly updated with the newest frame from the webcam feed. It uses a response function which is built into the flask framework to allow for a response to be sent to a web page, which In this case is an image.

```
# Here a decorator is used to start the generator function to allow for the video to be taken and processed and then returned to the webpage
@app.route('/video')
def video():
    return Response(generate_frames(),mimetype='multipart/x-mixed-replace; boundary=frame')
```

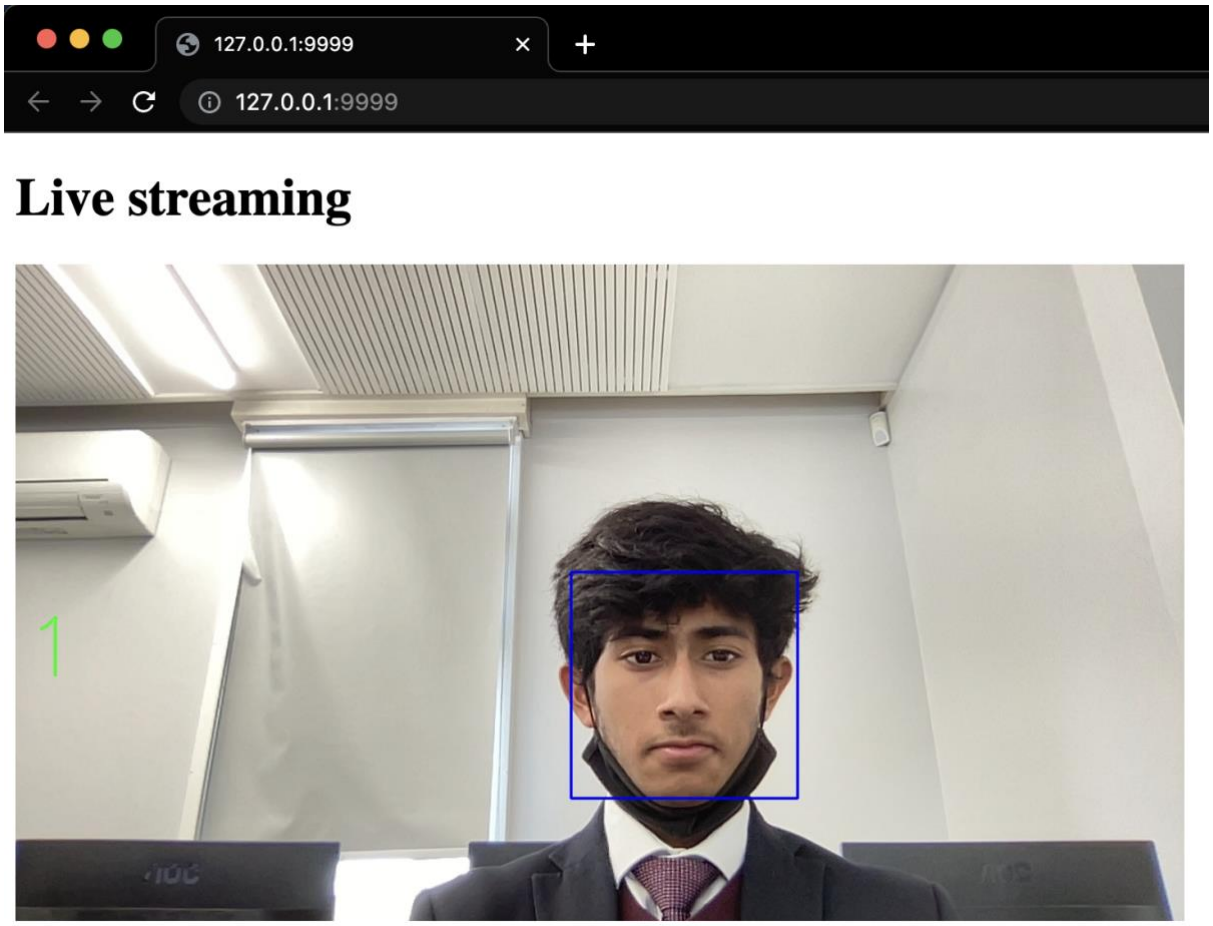
Testing

This stage is more so a template for the solution as the actual methods I will use for the detections will be different, the part that will be continued is the flask framework usage so the passing of frames to the browser is something that will be used in the final solution.

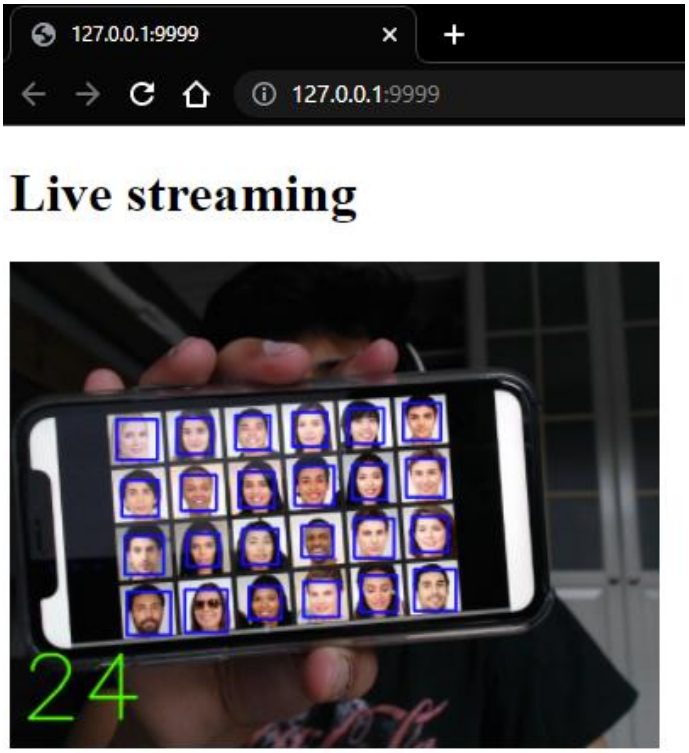
Therefore, the testing of this section is mainly of the code using the flask framework as opposed to the detections and count themselves.

The below screenshot shows the video being correctly passed to the browser and the output being given correctly. This can be accessed on multiple devices provided they give the same IP address and port number for the device running the detections.

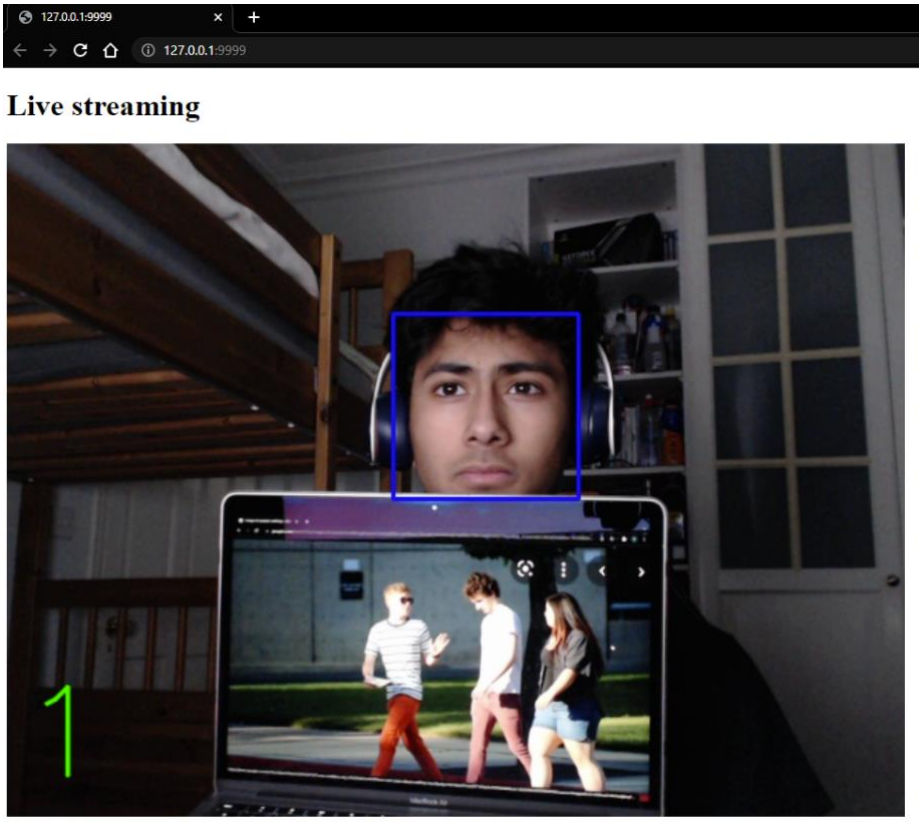
The count is given in green on the left side of the video output.



The screenshot below shows that the program can currently deal with multiple objects accurately, being able to detect all 24 people.



However, when tested using images of full people as opposed to faces, the correct count was not returned, this is likely due to the frontal face haar-cascade being less accurately when given full people rather than only faces facing forward. This will be addressed when the deep learning model is implemented which would be able to detect full people without issues.



User Feedback

After giving it to the different users, I asked the following questions:

1. What do you think of the app so far?
2. Does it function as required?
3. Are there any elements you would particularly like to be developed?

Responses:

Teacher:

1) The initial app seems to demonstrate basic functionality and it looks like it can be built up from
2) It only detects the front of faces which is not sufficient so this will need to be developed
3) As mentioned, the detection needs to be developed, apart from this, I would like to see the buttons in an upcoming stage

Shop Owner:

1) It seems to be a good initial development however clearly the detections need to be worked on and the page should be improved
2) If the detections were accurate, it would count the people in it's view rather than the people going in and out, so it doesn't
3) I would like to see the people in and out being developed as this is the most important thing I expect to see

Society organizer:

1) The app at its current state is clearly not ready but it gives a good sense of what it will be like so as long as the features are developed on this, it will fit my needs.
2) It doesn't but that Is something I believe you should be soon implementing
3) I particularly want to see some buttons added soon

The general feedback was to continue with the development and keep the primary features such as the detections and giving a count in and out a priority so this is what I will do. In the next stage I will likely work on the model creation to address the detections then go onto work onto integrating the counting.

Review

In this stage I have:

- Created a basic template using the flask framework which allows me to get a webcam freed in the browser and display it.
- Conducted basic face counting using haar cascades which will act as a filler for the deep learning part of the model.
- Setup the html for the webpage to be rendered using the flask framework and templates folder.

This stage demonstrates the basic functionality of the app, providing a webcam feed, detecting faces, drawing boxes around them and displaying a count of the number of detections. This achieves the basic purpose of the app, and it can be used to implement the additional features and the deep learning component can replace the haar cascades which will make the app more accurate.

This stage is more so a template for the solution as the actual methods I will use for the detections will be different, the part that will be continued is the flask framework usage so the passing of frames to the browser is something that will be used in the final solution.

Addressing the Success Criteria:

1. The correct number of people is given as minimum output only if the frontal face is clearly visible so this is partially met however the goal of this stage is not to get the detections completely correct so this will be fully addressed in a later stage
2. Toggling different features - this is not met in this stage
3. The app as a whole in this stage is extremely lightweight and is easily used, the program is run, and a count is given, despite it not being completely accurate.
4. Identifying people coming in and out of a room - this is not met in this stage
5. Bounding boxes displayed with live feed – this is met in this stage however will be made accurate with the detection model
6. Information menu – not met in this stage
7. Effective analysis – not met in this stage

So, Criteria 1 is partially met, criteria 3 and 5 are met and the rest will be addressed in later stages.

Stage 2 - Model Creation

Code, Explanation and Justification + Testing

First, I will be creating a virtual environment to work with so that all the dependencies, i.e., the libraries I need can be installed directly into that environment and this will avoid any dependency conflicts.

To do this, I will use Windows Command Prompt

First, I am creating a virtual environment called tfod (TensorFlow object detection):

```
C:\Users\vivia\Documents\Stage_2_v2\TFODCourse>python -m venv tfod
```

Then, I am activating the environment:

```
C:\Users\vivia\Documents\Stage_2_v2\TFODCourse>.\tfod\Scripts\activate

(tfod) C:\Users\vivia\Documents\Stage_2_v2\TFODCourse>
```

The next step I am taking is installing some libraries which I will need, these include Ipykernel and Jupyter notebook. Jupyter notebook will be used so I can run code straight away in cells rather than having to run whole files. This will be helpful when I am working with libraries and executing commands. Ipykernel will be needed because it allows me to associate a Jupyter Notebook to my virtual environment.

```
(tfod) C:\Users\vivia\Documents\Stage_2_v2\TFODCourse>pip install ipykernel
(tfod) C:\Users\vivia\Documents\Stage_2_v2\TFODCourse>pip install notebook
```

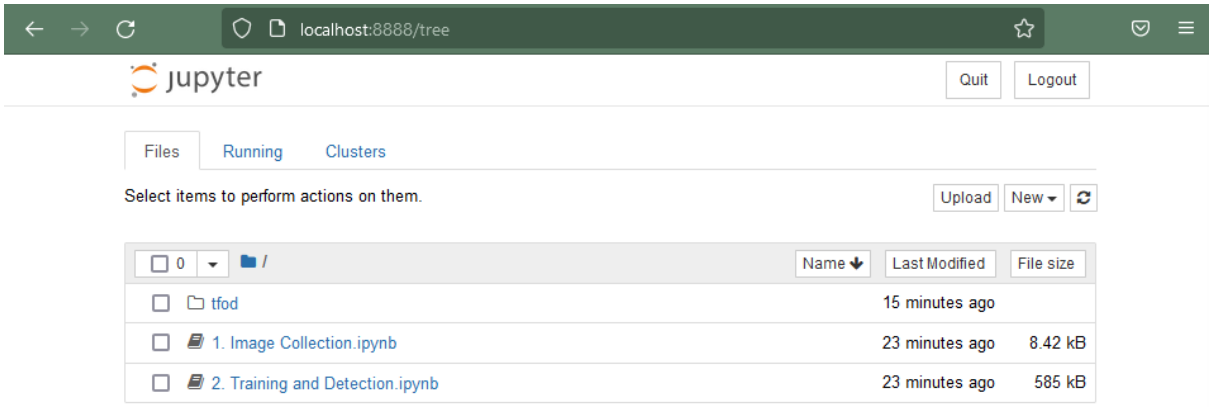
Next I can actuallly associate the notebook with the enviroment with the following command:

```
(tfod) C:\Users\vivia\Documents\Stage_2_v2\TFODCourse>python -m ipykernel install --user --name=tfod
Installed kernelspec tfod in C:\Users\vivia\AppData\Roaming\jupyter\kernels\tfod
```

At this point I have setup the virtual environment and associated it to the Jupyter notebooks, when I open Jupyter notebook with this command:

```
(tfod) C:\Users\vivia\Documents\Stage_2_v2\TFODCourse>jupyter notebook
```

This is what I see:



In it, there are 2 notebooks which I will add code to, one which I will be using for collecting the images I will be using to train my model and the other I will be using to execute the training and I will also include cells that will allow for the model to be used to carry out detections both in real time and on images so that I can easily test my model.

At this stage I am going to move onto collecting images and preparing them so that I can use them to train my model.

The first cell I have is installing OpenCV which will be important later on in the notebook. It is a computer vision library which will allow me to take advantage of the webcams and video input and I can do Realtime detections using it. Using this library will allow me to easily manipulate images and work with the webcam.

In [1]:
```
!pip install opencv-python
```

```
Collecting opencv-python
  Downloading opencv_python-4.5.5.62-cp36-abi3-win_amd64.whl (35.4 MB)
     ---------------------------------------- 35.4/35.4 MB 5.7 MB/s eta 0:0
0:00
Collecting numpy>=1.14.5
  Downloading numpy-1.21.5-cp37-cp37m-win_amd64.whl (14.0 MB)
     ---------------------------------------- 14.0/14.0 MB 8.4 MB/s eta 0:0
0:00
Installing collected packages: numpy, opencv-python
Successfully installed numpy-1.21.5 opencv-python-4.5.5.62
```

Next, I am importing some key libraries that will be needed for the rest of the notebook:

In [2]:
```
# Import opencv
import cv2

# Import uuid
import uuid

# Import Operating System
import os

# Import time
import time
```

I have already discussed OpenCV (cv2), the next library is uuid, which is a library that will allow me to give images unique names so that when they are saved, they will have unique names which I am using because it will prevent conflicts between file names in case, they are the same.

Os is needed so that I can work with the file system and save files into different directories. This will include saving the images I collect into certain locations.

Time will be used to pause between taking images.

In the next cell, I am defining the person label I will be using, and I will be storing it in an array as it would allow for other labels to be added if needed in the future.

In [3]:
```
# Label list defined to hold the label which we will use to define the object we want to detect
labels = ['person']
# variable defined to determine how many images will be collected later on
number_imgs = 5
```

Next, I am setting up an images file path in which I will be storing the images:

In [4]:
```
# Defines variable to store path where images will be stored
IMAGES_PATH = os.path.join('Tensorflow', 'workspace', 'images', 'collectedimages')
```

Then I am creating the path using a command and creating a folder for each label, right now this will just be a folder for the person label.

In [5]:
```
# Command to create the images path
!mkdir {IMAGES_PATH}

# Using for loop to create a folder for each label (Only creates a path for the person label)
for label in labels:
    path = os.path.join(IMAGES_PATH, label)
    if not os.path.exists(path):
        !mkdir {path}
```

Next, I am creating the option of collecting images myself using OpenCV:

In [7]:
```
for label in labels:
    cap = cv2.VideoCapture(1)# captures the video input
    print('Collecting images for {}'.format(label))# Iterates for all labels (only one for people)
    time.sleep(5)#Stops the program for 5 seconds so that there is time between images being taken
    for imgnum in range(number_imgs):# Iterates 5 times to get 5 images
        print('Collecting image {}'.format(imgnum))
        ret, frame = cap.read()#Captures a frame from the webcam input
        # Creates a unique image name, making use of uuid
        imgname = os.path.join(IMAGES_PATH,label,label+'.'+'{}.jpg'.format(str(uuid.uuid1())))
        #Uses OpenCV to write the image to the path just created for it
        cv2.imwrite(imgname, frame)
        cv2.imshow('frame', frame)#Displays the image that was taken
        time.sleep(2) #Stops the program for 2 second so that there is time between images being taken

        if cv2.waitKey(1) & 0xFF == ord('q'):#Checks if q is pressed at which point, the code would be halted
            break
cap.release()
cv2.destroyAllWindows()
```

However, when I run the code, the following error is given as output:

```
Collecting images for person
Collecting image 0

---------------------------------------------------------------------------
error                                     Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_19068\191152643.py in <module>
      9         imgname = os.path.join(IMAGES_PATH,label,label+'.'+'{}.jpg'.format(str(uuid.uuid1())))
     10         #Uses OpenCV to write the image to the path just created for it
---> 11         cv2.imwrite(imgname, frame)
     12         cv2.imshow('frame', frame)#Displays the image that was taken
     13         time.sleep(2) #Stops the program for 2 second so that there is time between images being taken

error: OpenCV(4.5.5) D:\a\opencv-python\opencv-python\opencv\modules\imgcodecs\src\loadsave.cpp:801: error: (-215:Ass
ertion failed) !_img.empty() in function 'cv::imwrite'
```

The code ran up to the point where the frame from the webcam input was read. This indicated to me that there is something wrong with the webcam input. To amend this, I decided to first try and change the port that was being used for the input from 1 to 0, so I implemented this change (on line 2). The amended code with the output is shown next:

```
In [6]:
for label in labels:
    cap = cv2.VideoCapture(0)# captures the video input
    print('Collecting images for {}'.format(label))# Iterates for all labels (only one for people)
    time.sleep(5)#Stops the program for 5 seconds so that there is time between images being taken
    for imgnum in range(number_imgs):# Iterates 5 times to get 5 images
        print('Collecting image {}'.format(imgnum))
        ret, frame = cap.read()#Captures a frame from the webcam input
        # Creates a unique image name, making use of uuid
        imgname = os.path.join(IMAGES_PATH,label,label+'.'+'{}.jpg'.format(str(uuid.uuid1())))
        #Uses OpenCV to write the image to the path just created for it
        cv2.imwrite(imgname, frame)
        cv2.imshow('frame', frame)#Displays the image that was taken
        time.sleep(2) #Stops the program for 2 second so that there is time between images being taken

        if cv2.waitKey(1) & 0xFF == ord('q'):#Checks if q is pressed at which point, the code would be halted
            break
cap.release()
cv2.destroyAllWindows()
```

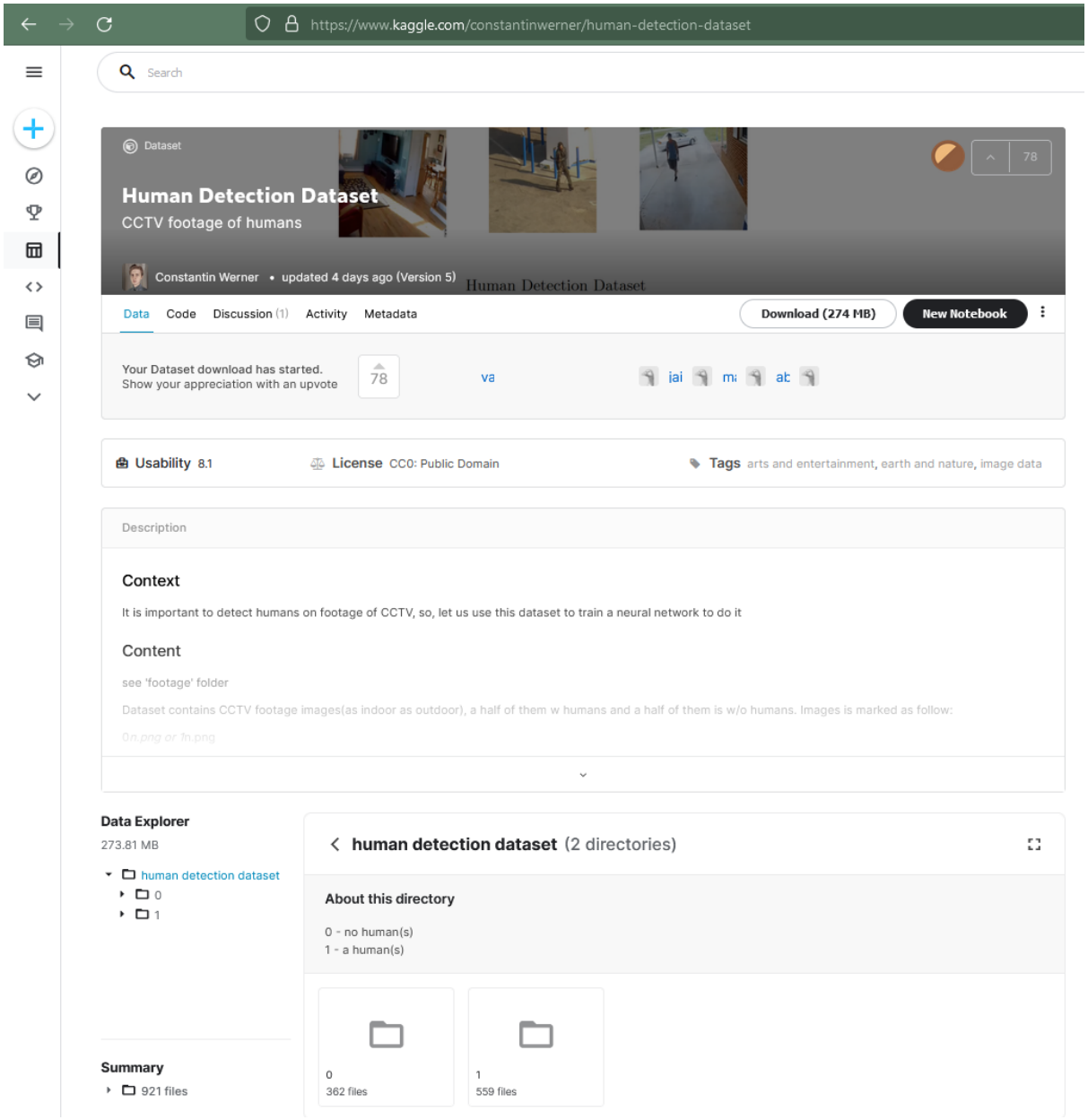
```
Collecting images for person
Collecting image 0
Collecting image 1
Collecting image 2
Collecting image 3
Collecting image 4
```

The code now ran fine which means that OpenCV recognizes my webcam using port 0 rather than 1 which is something I have to keep in mind when using OpenCV later on in development.

At this point, I decided that continually taking images using this code wouldn't be as effective as using a prebuilt dataset of images as those images would be of a higher quality and there would be many more images, I could choose from to include in my training. The quality would be higher as there would be a wider range of environments that the images would be taken in compared to what I am capable of taking. Also, this would save a significant amount of time as I wouldn't have to collect all the images myself. Therefore, I came to the conclusion of using a dataset which I had found on Kaggle (A machine learning website) that had over 900 images, ones with humans and ones without.

This is the URL for the dataset, (accessed on 16/02/2022):
https://www.kaggle.com/constantinwerner/human-detection-dataset

This is what the webpage that provided the dataset looks like:



After downloading the dataset, I first selected a few good images which I thought would allow the model to achieve a reasonable accuracy. This is so that I can start off training and see how accurate the model can be initially. Later on, when performance tuning, I can continue training the model on a wider set of data and change other components of the model if needed to maximise the performance of the model.

These are the images I will initially be using to train my model:

Next, I have to label the images so that the person is labelled in the image, and they can be identified by the model so that the correct part of the image is being used for training.

To do this, I will be using an application called Label_Img.

First, I have to install some libraries that will be needed for this:

```
In [9]:  1 !pip install --upgrade pyqt5 lxml
```

```
Collecting pyqt5
  Downloading PyQt5-5.15.6-cp36-abi3-win_amd64.whl (6.7 MB)
     ---------------------------------------- 6.7/6.7 MB 6.6 MB/s eta 0:0
0:00
Collecting lxml
  Downloading lxml-4.7.1-cp37-cp37m-win_amd64.whl (3.7 MB)
     ---------------------------------------- 3.7/3.7 MB 9.3 MB/s eta 0:0
0:00
Collecting PyQt5-Qt5>=5.15.2
  Using cached PyQt5_Qt5-5.15.2-py3-none-win_amd64.whl (50.1 MB)
Collecting PyQt5-sip<13,>=12.8
  Downloading PyQt5_sip-12.9.1-cp37-cp37m-win_amd64.whl (76 kB)
     ---------------------------------------- 76.9/76.9 KB 4.4 MB/s eta
0:00:00
Installing collected packages: PyQt5-Qt5, PyQt5-sip, lxml, pyqt5
Successfully installed PyQt5-Qt5-5.15.2 PyQt5-sip-12.9.1 lxml-4.7.1 pyqt5
-5.15.6
```

Before I can install it, I have to define the path for it to be downloaded into:

```
In [10]:  1 # Creating the path where the Labelimg application will be stored
          2 LABELIMG_PATH = os.path.join('Tensorflow', 'labelimg')
```

Now I can actually create the path and git clone the files for the application into the defined path then install it.

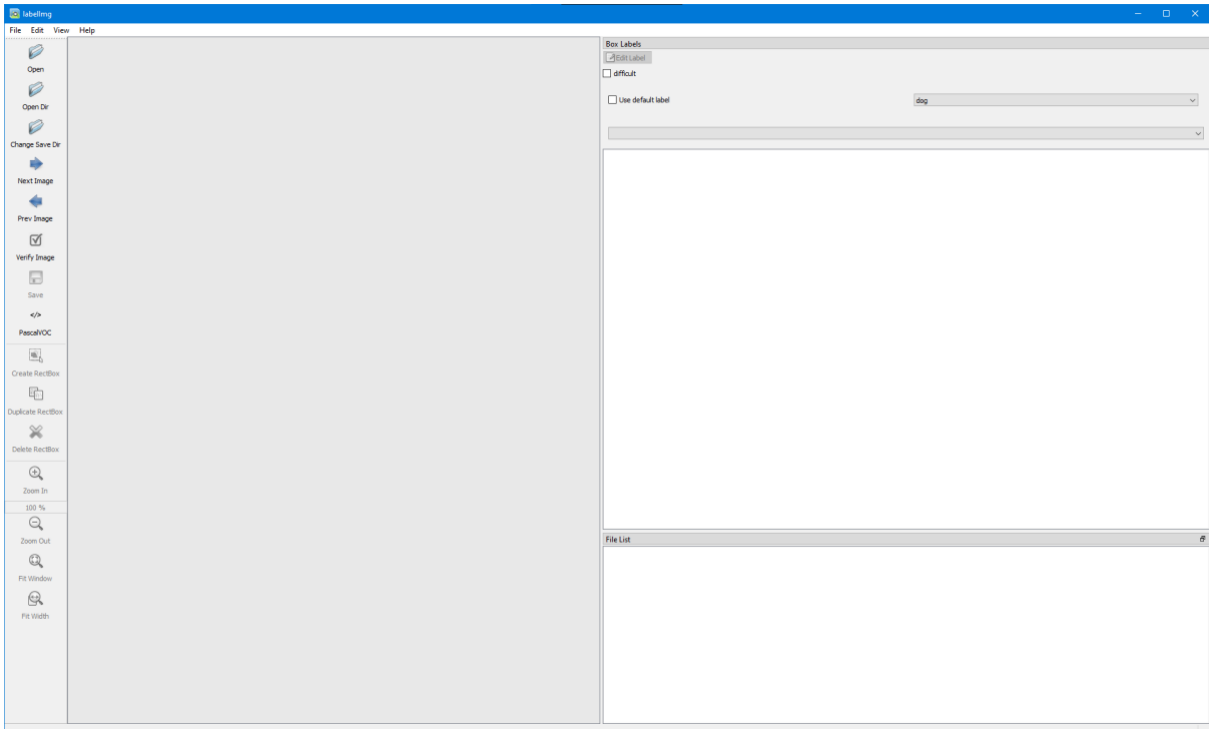
```
In [12]:  1 !mkdir {LABELIMG_PATH}
          2 !git clone https://github.com/tzutalin/labelImg {LABELIMG_PATH}
```

```
In [13]:
# Installing labelimg
!cd {LABELIMG_PATH} && pyrcc5 -o libs/resources.py resources.qrc
```

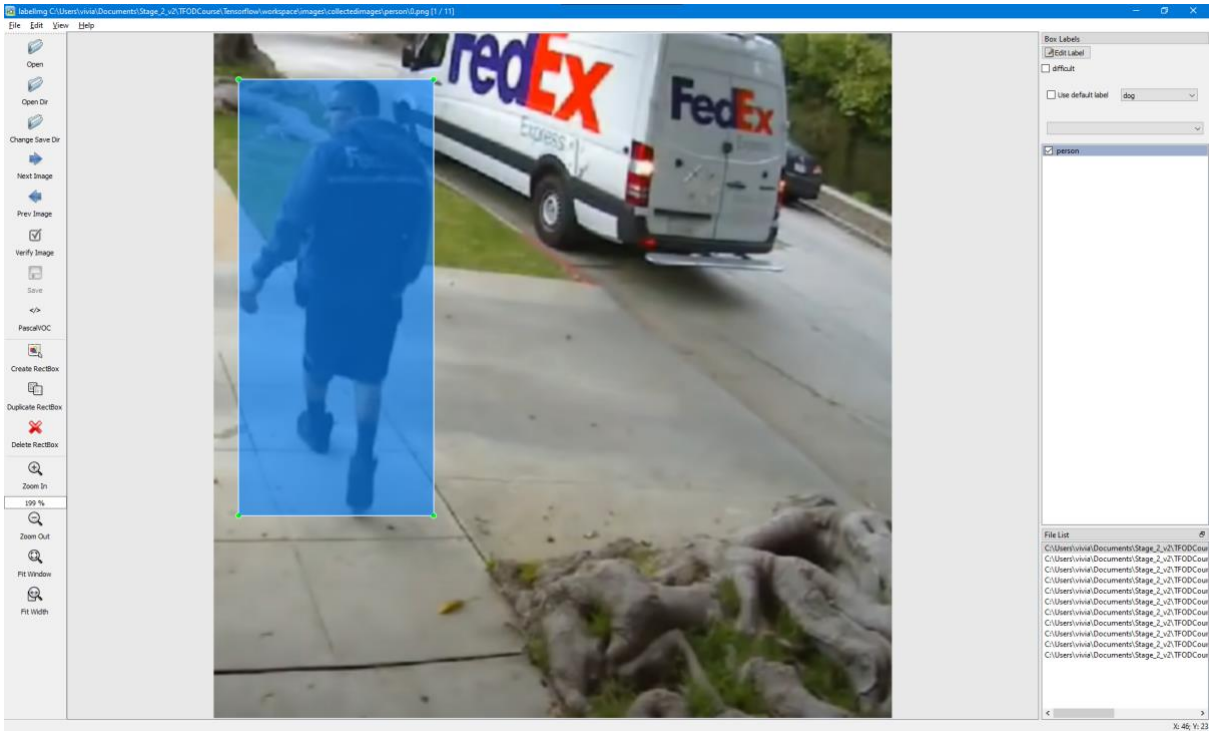
Now that I have installed LabelImg, I can start using it. This is the command to open it:

```
In [*]:
# Opening LabelImg
!cd {LABELIMG_PATH} && python labelImg.py
```

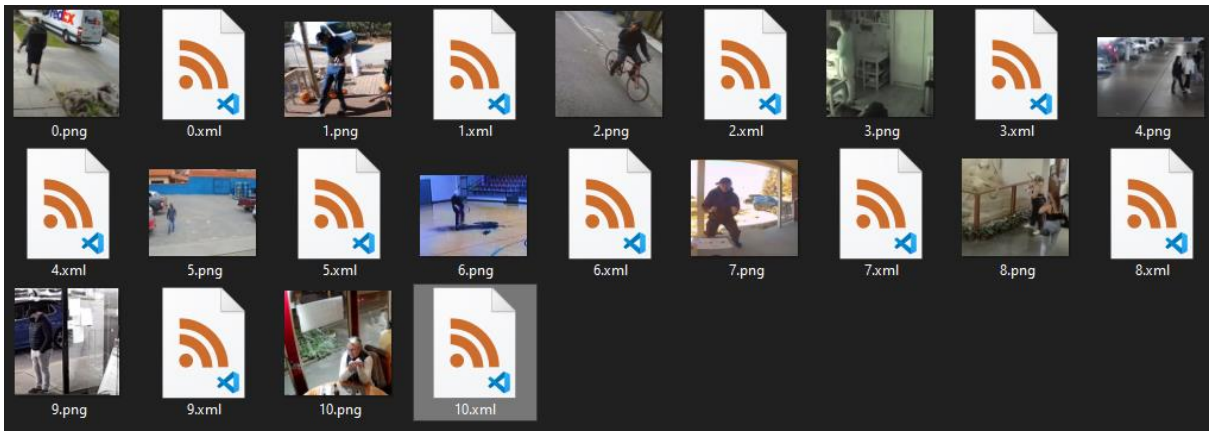
And this is the interface:



When adding the bounding boxes, this is what the interface looks like:



Once I have labelled all the images, I have the annotations saved in a separate file that has the same name as the image:



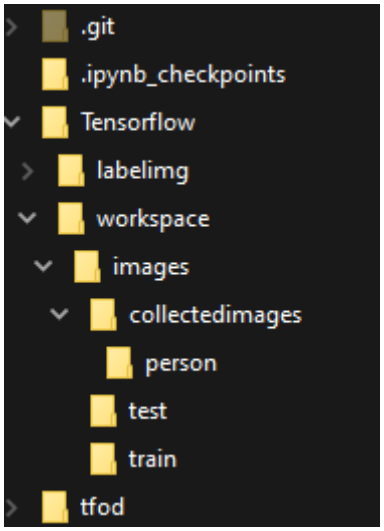
An example of how the annotation of an image looks like is:

```xml
<annotation>
    <folder>person</folder>
    <filename>10.png</filename>
    <path>C:\Users\vivia\Documents\Stage_2_v2\TFODCourse\Tensorflow\workspace\images\collectedimages\person\10.png</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>480</width>
        <height>471</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>person</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>243</xmin>
            <ymin>207</ymin>
            <xmax>406</xmax>
            <ymax>431</ymax>
        </bndbox>
    </object>
</annotation>
```

The final step for preparing the images is splitting them into a training and testing partition. The reason I am doing this is because I want to train the model with most of the images but some of them, I want to save so that when I am evaluating the model, I can use images that it has never seen before which will be a good test of the model's accuracy, and it will also give me an indication as to how the model will perform when I start using it as part of the web page.

In total I have 10 images currently, I am going to use 7 for training and 3 for testing. I have chosen this 7:3 ratio as I want the majority of my data to be used to train the model and only a few to test it. If this ratio doesn't work well for me then I can always add more to test and train when I am performance tuning.

This is my current folder structure:



Now I can move onto the training notebook as I have finished collecting and preparing images:

2. Training and Detection.ipynb

The first cell is once again importing the OS library as I will be using it extensively in this section of the model creation as well.

In [1]:
```
# Importing OS library
import os
```

After this, I am defining some key paths and names which will be crucial when executing commands during training. The reason I am defining a pre-trained model name and URL is because I will be taking advantage of transfer learning from an already successful architecture from the TensorFlow model Zoo. The model I have initially chosen is an SSD_Mobilenet model which has as speed of 22 milliseconds and a mean average precision (mAP) of 22.2. When compared to the other models, this specific model displayed a good trade-off between speed and accuracy, and I need both for my project to work well and deliver on stakeholders needs.

In [2]:
```
# Name of my model
CUSTOM_MODEL_NAME = 'my_ssd_mobnet'
# Name of the pre-trained model I am using
PRETRAINED_MODEL_NAME = 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8'
# URL of the pre-trained model I am using
PRETRAINED_MODEL_URL = 'http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_fpnlite
# This is a pre-made script that will generate a specific file I need
TF_RECORD_SCRIPT_NAME = 'generate_tfrecord.py'
# This will be the name of the label map that is needed when detecting with thte model
LABEL_MAP_NAME = 'label_map.pbtxt'
```

In [2]:
```
ction/tf2/20200711/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz'

th thte model
```

The next thing I'm doing is creating a dictionary called paths and creating key value pairs for different paths that I will be using throughout the rest of the code.

In [3]:

```
paths = {
    'WORKSPACE_PATH': os.path.join('Tensorflow', 'workspace'),
    'SCRIPTS_PATH': os.path.join('Tensorflow','scripts'),
    'APIMODEL_PATH': os.path.join('Tensorflow','models'),
    'ANNOTATION_PATH': os.path.join('Tensorflow', 'workspace','annotati
    'IMAGE_PATH': os.path.join('Tensorflow', 'workspace','images'),
    'MODEL_PATH': os.path.join('Tensorflow', 'workspace','models'),
    'PRETRAINED_MODEL_PATH': os.path.join('Tensorflow', 'workspace','pr
    'CHECKPOINT_PATH': os.path.join('Tensorflow', 'workspace','models',
    'OUTPUT_PATH': os.path.join('Tensorflow', 'workspace','models',CUST
    'TFJS_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTOM_
    'TFLITE_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTO
    'PROTOC_PATH':os.path.join('Tensorflow','protoc')
 }
```

In [3]:

```

oin('Tensorflow', 'workspace'),
.n('Tensorflow','scripts'),
)in('Tensorflow','models'),
 join('Tensorflow', 'workspace','annotations'),
['Tensorflow', 'workspace','images'),
['Tensorflow', 'workspace','models'),
.path.join('Tensorflow', 'workspace','pre-trained-models'),
 join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME),
('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'export'),
ensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'tfjsexport'),
['Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'tfliteexport'),
['Tensorflow','protoc')

```

Similarly, I am defining a files dictionary to hold certain file locations including the pipeline config (which will be created later on), the TF record script which will be needed before actual training can be startedm and the label map path which is what is needed when running the model.

In [4]:

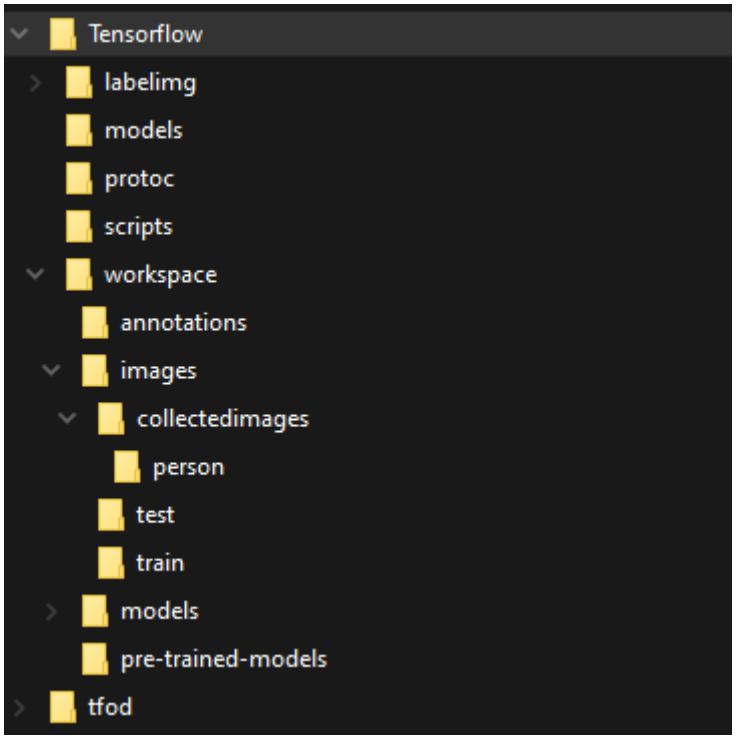
```
files = {
    'PIPELINE_CONFIG':os.path.join('Tensorflow', 'workspace','models',
    'TF_RECORD_SCRIPT': os.path.join(paths['SCRIPTS_PATH'], TF_RECORD_S
    'LABELMAP': os.path.join(paths['ANNOTATION_PATH'], LABEL_MAP_NAME)
}
```

Now, I am creating the directories that I defined earlier in the paths dictionary.

In [5]:

```
# Looping over each path value in the paths dictionary
for path in paths.values():
    # Checking if the path already exists
    if not os.path.exists(path):
        #Creating the path
        !mkdir {path}
```

After creating all these directories, this is my folder structure:



Each folder is needed to store files that will be needed for the training process.

The next stage involves making sure I have the correct libraries for training setup. The first I am installing is ‘wget’ which will help in retrieving the model architecture that I will be using.

In [6]:
```
!pip install wget
import wget
```

```
Collecting wget
  Using cached wget-3.2.zip (10 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Using legacy 'setup.py install' for wget, since package 'wheel' is not in
stalled.
Installing collected packages: wget
  Running setup.py install for wget: started
  Running setup.py install for wget: finished with status 'done'
Successfully installed wget-3.2
```

After this, I am downloading the TensorFlow Object Detection API which is a major set of files that I will need for handling the model. Specifically, I will be using the object detection section of the API.

In [7]:
```
# Checking if the path for the API to be downloaded in exists
if not os.path.exists(os.path.join(paths['APIMODEL_PATH'], 'research',
    # Downloads the API files using git clone
    !git clone https://github.com/tensorflow/models {paths['APIMODEL_PA
```

```
In [7]:
1 ch for the API to be downloaded in exists
2 (os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection')):
3 API files using git clone
4 //github.com/tensorflow/models {paths['APIMODEL_PATH']}
```

Next, I am installing TensorFlow Object Detection, first specifying the URL then using wget to download the URL. After this, I am executing commands to move files and enter the correct directories so that the install can be carried out.

```
In [9]:
1 # Install Tensorflow Object Detection
2 url="https://github.com/protocolbuffers/protobuf/releases/download/v3.1
3 wget.download(url)
4 !move protoc-3.15.6-win64.zip {paths['PROTOC_PATH']}
5 !cd {paths['PROTOC_PATH']} && tar -xf protoc-3.15.6-win64.zip
6 os.environ['PATH'] += os.pathsep + os.path.abspath(os.path.join(paths['
7 !cd Tensorflow/models/research && protoc object_detection/protos/*.prot
8 !cd Tensorflow/models/research/slim && pip install -e .
```

```
In [9]:
1
2 5.6/protoc-3.15.6-win64.zip"
3
4
5
6 PROTOC_PATH'], 'bin'))
7 o --python_out=. && copy object_detection\\packages\\tf2\\setup.py setup
8
```

```
In [9]:
1
2
3
4
5
6
7 :\\setup.py setup.py && python setup.py build && python setup.py install
8
```

Now I am running a verification script to make sure that TensorFlow is installed correctly:

```
In [10]:
1 VERIFICATION_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research',
2 # Verify Installation
3 !python {VERIFICATION_SCRIPT}
```

```
Traceback (most recent call last):
  File "Tensorflow\models\research\object_detection\builders\model_builde
r_tf2_test.py", line 22, in <module>
    import tensorflow.compat.v1 as tf
ModuleNotFoundError: No module named 'tensorflow'
```

However, this came up with an error, so I tried the following command first:

```
In [15]:
1 !pip install tensorflow --upgrade
```

Then, various libraries that TensorFlow depended on were listed with ‘no module found’ errors, these errors were easily fixed by installing the required modules. After this was done, the verification script gave the following output:

```
Ran 24 tests in 40.600s

OK (skipped=1)
```

Additionally, I am using the following command to test if all the packages I want are installed:

```
In [7]: 1 !pip list
```

```
Package                       Version             Editable project locatio
n
----------------------------- --------------------
-------------------------------------------------------------------------
-----
absl-py                       1.0.0
argon2-cffi                   21.3.0
argon2-cffi-bindings          21.2.0
astunparse                    1.6.3
attrs                         21.4.0
backcall                      0.2.0
bleach                        4.1.0
cached-property               1.5.2
cachetools                    5.0.0
certifi                       2021.10.8
cffi                          1.15.0
charset-normalizer            2.0.12
colorama                      0.4.4
cycler                        0.11.0
debugpy                       1.5.1
decorator                     5.1.1
defusedxml                    0.7.1
entrypoints                   0.4
flatbuffers                   2.0
gast                          0.5.3
google-auth                   2.6.0
google-auth-oauthlib          0.4.6
google-pasta                  0.2.0
grpcio                        1.44.0
h5py                          3.6.0
idna                          3.3
importlib-metadata            4.11.1
importlib-resources           5.4.0
ipykernel                     6.9.1
ipython                       7.31.1
ipython-genutils              0.2.0
jedi                          0.18.1
Jinja2                        3.0.3
jsonschema                    4.4.0
jupyter-client                7.1.2
```

It gives a long list of packages, and I can easily check if a package is installed or if I needed to install it.

This means that TensorFlow is successfully installed and now I can move on.

The next step I am taking is to import the object_detection module which I had previously downloaded so I could use it in the notebook.

```
In [17]:
1 import object_detection
```

And the final part of setting up the downloaded files is to add the pre-trained model file which I am going to use as part of the custom training process to a folder I have already created. The following code shows how I am doing this:

```
In [10]:
1 wget.download(PRETRAINED_MODEL_URL)
2 !move {PRETRAINED_MODEL_NAME+'.tar.gz'} {paths['PRETRAINED_MODEL_PATH']
3 !cd {paths['PRETRAINED_MODEL_PATH']} && tar -zxvf {PRETRAINED_MODEL_NAM
```

```
In [10]:
1 d(PRETRAINED_MODEL_URL)
2 AINED_MODEL_NAME+'.tar.gz'} {paths['PRETRAINED_MODEL_PATH']}
3 PRETRAINED_MODEL_PATH']} && tar -zxvf {PRETRAINED_MODEL_NAME+'.tar.gz'}
```

The next step is to create a label map, this is needed when conducting inference, so when using the model, the label map is needed to relate the detections to certain classes and display them correctly.

```
In [16]:
# Creating a list containing a dictionary fo each label (only person in my case)
labels = [{'name':'Person', 'id':1}]

# Creating the labelmap file in the already created labelmap folder
with open(files['LABELMAP'], 'w') as f:
    # Looping through each label (only goes through person)
    for label in labels:
        # Writes the required content for the label with the name and id
        # anything written is in the correct format
        f.write('item { \n')
        f.write('\tname:\'{}\'\n'.format(label['name']))
        f.write('\tid:{}\n'.format(label['id']))
        f.write('}\n')
```

This is what the created label map file looks like:

```
item {
    name:'person'
    id:1
}
```

Now I am going to create a specific file in a specific file format needed for training, this will be called TFRecords and are a binary file format for storing data. Using this format will help speed up training for my custom object detection model.

First, I am getting a TensorFlow script that will convert files to TFRecords, using git clone.

```
In [12]:
if not os.path.exists(files['TF_RECORD_SCRIPT']):
    !git clone https://github.com/tf/GenerateTFRecord {paths['SCRIPTS_PATH']}
```

```
Cloning into 'Tensorflow\scripts'...
```

Next, I am running a command to use the script, passing certain parameters such as the image path for train and test which I earlier created, the label map, and the path that the annotations should be stored in as well as the name of the TFRecords that should be saved.

However, when running this I came across this error:

```
KeyError: 'person'
```

After careful inspection of my code, I realized I had made a case error when creating the label map, the name should have been 'person' as this is how I labelled each image with the labelimg application however I used 'Person' which caused the record generation to give this KeyError. I amended the code for the label generation:

```
In [11]:
# Creating a list containing a dictionary fo each label (only person in my case)
labels = [{'name':'person', 'id':1}]

# Creating the labelmap file in the already created labelmap folder
with open(files['LABELMAP'], 'w') as f:
    # Looping through each label (only goes through person)
    for label in labels:
        # Writes the required content for the label with the name and id
        # anything written is in the correct format
        f.write('item { \n')
        f.write('\tname:\'{}\'\n'.format(label['name']))
        f.write('\tid:{}\n'.format(label['id']))
        f.write('}\n')
```

And it gave me the following output successfully:

```
In [15]:
!python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'train')} -l {file
!python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'test')} -l {files
```

```
Successfully created the TFRecord file: Tensorflow\workspace\annotations\train.record
Successfully created the TFRecord file: Tensorflow\workspace\annotations\test.record
```

```
In [15]:
train')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'], 'train.record')}
test')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'], 'test.record')}
```

```
Successfully created the TFRecord file: Tensorflow\workspace\annotations\train.record
Successfully created the TFRecord file: Tensorflow\workspace\annotations\test.record
```

The next thing I will be looking at is the pipeline.config file for my chosen model architecture which is a mobilenet model. This file specifies the architectures of the model itself, showing the different operations it will carry out, such as data augmentation where each image will be augmented to correctly fit the model as input. I will be editing this file so that training will work for my specific setup.

This is what the pipeline.config path looks like:

```
model {
  ssd {
    num_classes: 90
    image_resizer {
      fixed_shape_resizer {
        height: 320
        width: 320
      }
    }
    feature_extractor {
      type: "ssd_mobilenet_v2_fpn_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 3.9999998989515007e-05
          }
        }
        initializer {
          random_normal_initializer {
            mean: 0.0
            stddev: 0.009999999776482582
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.996999979019165
          scale: true
          epsilon: 0.0010000000474974513
        }
      }
      use_depthwise: true
      override_base_feature_extractor_hyperparams: true
      fpn {
        min_level: 3
        max_level: 7
        additional_layer_depth: 128
      }
    }
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
        use_matmul_gather: true
```

Firstly, I am copying the pipeline.config file to the checkpoint path I previously defined.

```
In [20]: 1 !copy {os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'pipeline.config'
         1 file(s) copied.
In [20]: 1 _PATH'], PRETRAINED_MODEL_NAME, 'pipeline.config')} {os.path.join(paths['CHECKPOINT_PATH'])}
         1 file(s) copied.
```

Next, I am updating the file itself, completely using code as it is more efficient, and it will edit the file accurately so that training runs smoothly.

The first step in updating the config is importing the libraries that will be needed, these include tensorflow, object detection config_util, pipeline_pb2 and text_fprmat from google.protobuf.

Next, I will be assigning the configs of the pipeline.config file to a config variable.

```
In [22]: 1 # Assigning config of pipleline.config to config variable
         2 config = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
```

When outputting the current config, this is what is output:

```
In [23]: 1 config
```

```
    }
    use_moving_average: false
  }
  fine_tune_checkpoint: "PATH_TO_BE_CONFIGURED"
  num_steps: 50000
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "classification"
  fine_tune_checkpoint_version: V2,
  'train_input_config': label_map_path: "PATH_TO_BE_CONFIGURED"
  tf_record_input_reader {
    input_path: "PATH_TO_BE_CONFIGURED"
  },
  'eval_config': metrics_set: "coco_detection_metrics"
  use_moving_averages: false,
  'eval_input_configs': [label_map_path: "PATH_TO_BE_CONFIGURED"
  shuffle: false
```

There are several ‘PATH_TO_BE_CONFIGURED’ sections which need to be replaced. These paths include paths to the label maps and TFRecords which I earlier created.

Next, I am reading the config file and assigning the text from it to the pipeline_config variable.

In [25]:
```
# Assigning the number of classes (1 for person) to the section in the pipeline config
pipeline_config.model.ssd.num_classes = len(labels)
# Assinging the batch size which is a parameter used when training
pipeline_config.train_config.batch_size = 4
# Assigning the pretrained model path
pipeline_config.train_config.fine_tune_checkpoint = os.path.join(paths['PRETRAINED_MODEL_PAT
# Changing a checkpoint type to detection
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
# Assignign the label map path
pipeline_config.train_input_reader.label_map_path= files['LABELMAP']
# Assinging the train TFRecord path
pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] = [os.path.join(path
# Assinging the label map to be used when evaluating
pipeline_config.eval_input_reader[0].label_map_path = files['LABELMAP']
# Assinging the test TFRecord path
pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:] = [os.path.join(pa
```

In [25]:
```
 the section in the pipeline config

d when training


 os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'checkpoint', 'ckpt-0')

ype = "detection"

files['LABELMAP']

reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'], 'train.record')]
g
 = files['LABELMAP']

t_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'], 'test.record')]
```

Finally, I am writing the paths to the config file and thereby updating it so that it can be used for training.

In [26]:
```
# Turning the pipeline_config into text that can be written
config_text = text_format.MessageToString(pipeline_config)
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "wb") as f:
    # Writing the additional paths to the config
    f.write(config_text)
```

Now, I am able to start training. I will do this using a command. To create the command, I am going to create a path to the training script first.

In [27]:
```
# Creating path to training script
TRAINING_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'mode
```

In [27]:
```
to training script
= os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'model_main_tf2.py')
```

Then I am creating the command by assigning a string to a command variable. I am formatting the string with firstly the training script then various arguments that will be passed to the training script. These include the location of the model, the location of the pipeline config and the number of training steps (epochs). The number of epochs is a hyperparameter which is something that I can performance tune to gain better performance. For now, I will leave it at 2000 training steps.

```
In [28]: 1 command = "python {} --model_dir={} --pipeline_config_path={} --num_train_steps=2000".format
```

```
In [28]: 1 train_steps=2000".format(TRAINING_SCRIPT, paths['CHECKPOINT_PATH'],files['PIPELINE_CONFIG'])
```

This is what the command looks like after it has been created:

```
In [29]: 1 print(command)

python Tensorflow\models\research\object_detection\model_main_tf2.py --model_dir=Tensorflow\w
orkspace\models\my_ssd_mobnet --pipeline_config_path=Tensorflow\workspace\models\my_ssd_mobne
t\pipeline.config --num_train_steps=2000
```

Now I am able to execute it. I will be executing it in a separate command prompt window so that I can see the output of the progress of the training.

When trying to execute, I got another 'no module found' error:

```
ModuleNotFoundError: No module named 'pycocotools'
```

I fixed this by installing it as before.

This happened for a few other modules, so I spent some time installing the required modules.

When the script started running properly, I noticed it was taking a significant amount of time to start training, then I realized that I didn't have TensorFlow-GPU installed. Since I have a GPU, and want to take advantage of it for training, I am going to install the GPU version of TensorFlow so that training will be quicker.

Once installed, I ran the command for training. The output for one step in the training process is shown:

```
INFO:tensorflow:Step 100 per-step time 0.661s
I0220 12:41:53.997941 13844 model_lib_v2.py:707] Step 100 per-step time 0.661s
INFO:tensorflow:{'Loss/classification_loss': 0.19592775,
 'Loss/localization_loss': 0.39158416,
 'Loss/regularization_loss': 0.15619433,
 'Loss/total_loss': 0.7437062,
 'learning_rate': 0.0319994}
I0220 12:41:54.007445 13844 model_lib_v2.py:708] {'Loss/classification_loss': 0.19592775,
 'Loss/localization_loss': 0.39158416,
 'Loss/regularization_loss': 0.15619433,
 'Loss/total_loss': 0.7437062,
 'learning_rate': 0.0319994}
```

This shows the loss at each stage, and this is something that the deep learning model is trying to minimize through a backpropagation algorithm.

The final stage gave the following output:

```
INFO:tensorflow:Step 2000 per-step time 0.183s
I0220 12:47:44.954508 13844 model_lib_v2.py:707] Step 2000 per-step time 0.183s
INFO:tensorflow:{'Loss/classification_loss': 0.049984515,
 'Loss/localization_loss': 0.019881459,
 'Loss/regularization_loss': 0.14634904,
 'Loss/total_loss': 0.21621501,
 'learning_rate': 0.07991781}
I0220 12:47:44.955513 13844 model_lib_v2.py:708] {'Loss/classification_loss': 0.049984515,
 'Loss/localization_loss': 0.019881459,
 'Loss/regularization_loss': 0.14634904,
 'Loss/total_loss': 0.21621501,
 'learning_rate': 0.07991781}
```

This shows that the loss decreased from 0.195…. to 0.0499… which is significant and indicates that training went well. Now I am going to move onto evaluating the model. This means I will find out some metrics such as accuracy and precision about the model.

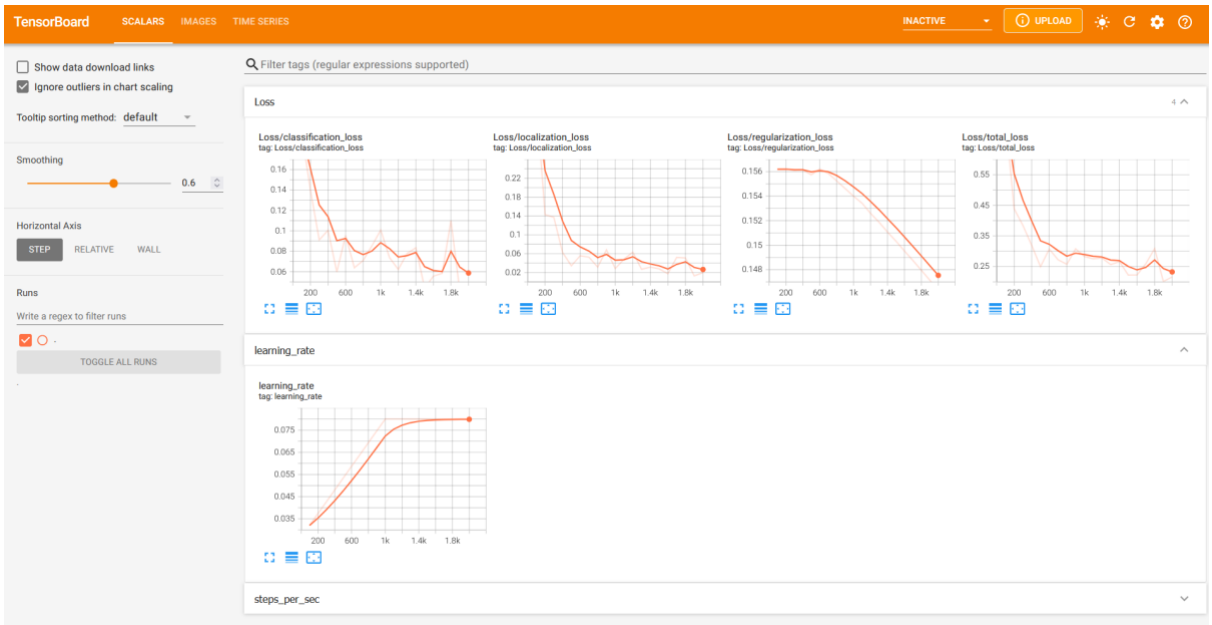
I will follow the same process in creating a command for evaluating as I did for training:

```
In [15]:
command = "python {} --model_dir={} --pipeline_config_path={} --checkpoint_dir={}".format(TRAINING_SCRIPT, paths['CH

In [15]:
ckpoint_dir={}".format(TRAINING_SCRIPT, paths['CHECKPOINT_PATH'],files['PIPELINE_CONFIG'], paths['CHECKPOINT_PATH'])
```

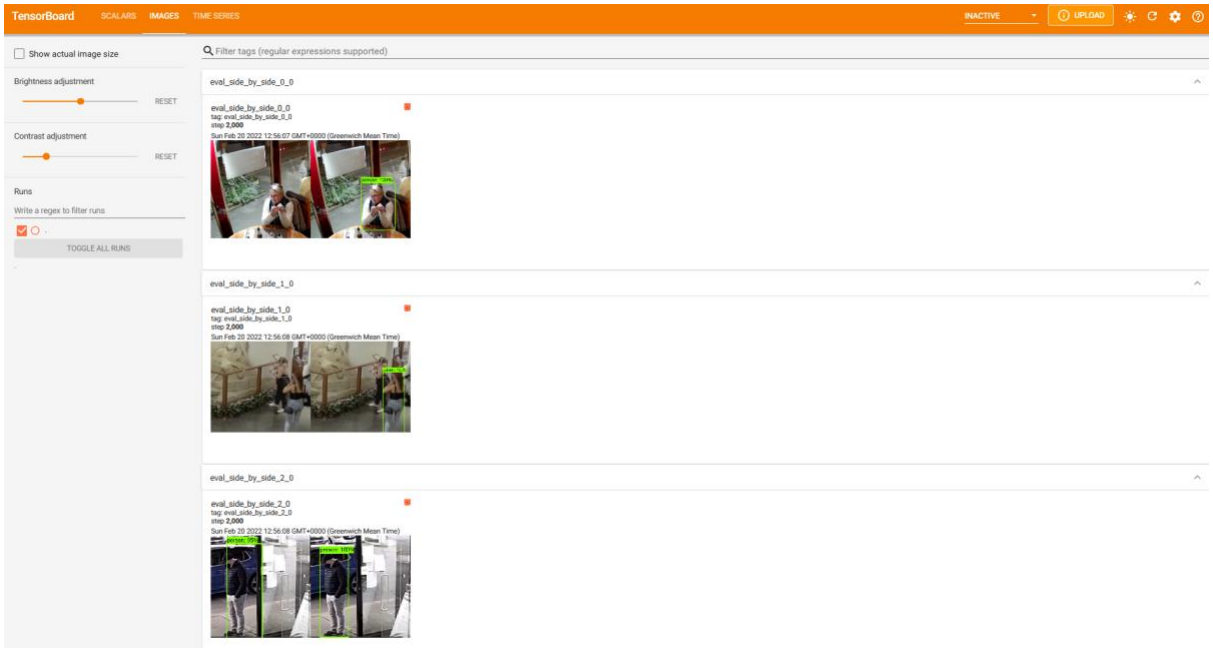
This is the command I will be using:

```
In [16]:
print(command)

python Tensorflow\models\research\object_detection\model_main_tf2.py --model_dir=Tensorflow\workspace\models\my_ssd_m
obnet --pipeline_config_path=Tensorflow\workspace\models\my_ssd_mobnet\pipeline.config --checkpoint_dir=Tensorflow\wo
rkspace\models\my_ssd_mobnet
```

Next, I will be using tensor board to visualize the learning and help with seeing how the model trained:



I am also using tensor board to see how the evaluation went, using the test images, I created from earlier, this is how the model performed:



So, the model detected the person with high accuracy in the last image but wasn't able to detect the first 2. This means I will need to train it further, which is something I can address in performance tuning.

I am going to now move onto doing testing of the model, first on an image I have saved for testing. First, I am again importing modules necessary for using the model:

In [1]:
```
# Importing Modules for testing
import os
import tensorflow as tf
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
from object_detection.utils import config_util
```

Then, I am loading the pipeline config file which I mentioned earlier and building a model and storing it as detection_model. Then, I am restoring the most recent checkpoint from training (checkpoint 4). Finally, I am defining a function to return detections as a dictionary.

In [11]:
```
# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
detection_model = model_builder.build(model_config=configs['model'], is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(paths['CHECKPOINT_PATH'], 'ckpt-4')).expect_partial()

@tf.function
def detect_fn(image):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections
```

Now, I am importing modules needed for image detection:

In [6]:
```
# Importing modules for image detection
import cv2
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
```

Here I am creating a category index which is something that is needed to detect the classes from a model so this would be needed to detect people and is created from the label map.

In [7]:
```
# Creating a category_index, needed for detections, created from label map
category_index = label_map_util.create_category_index_from_labelmap(files['LABELMAP'])
```

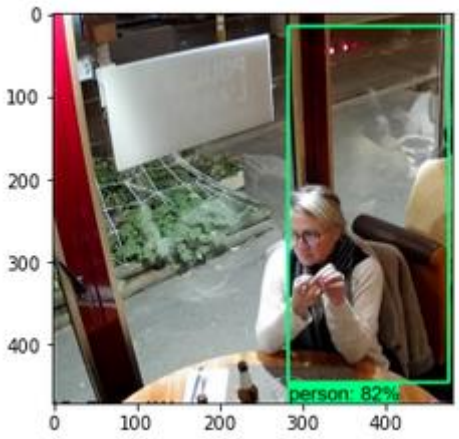
Next, I am selecting an image to test the model on.

In [13]:
```
# This is the path of the image which the model is going to be tested on
IMAGE_PATH = os.path.join(paths['IMAGE_PATH'], 'test', '10.png')
```

Finally, I am utilizing the function I defined earlier to carry out the detections. I am using OpenCV to read in the image, then I convert it to a NumPy array so it can be worked with. Then I am running the detect_fn function to retrieve the detections in the form of a dictionary. Then I am displaying the detections on a pyplot.

In [14]:
```
# First reading in the image and converting it to a numpy array so it can be worked with
img = cv2.imread(IMAGE_PATH)
image_np = np.array(img)

input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
# detection carried out on image and retrurned as a dictionary
detections = detect_fn(input_tensor)

# Finding the detections as well as the number of detections from the dictionary
num_detections = int(detections.pop('num_detections'))
detections = {key: value[0, :num_detections].numpy()
              for key, value in detections.items()}
detections['num_detections'] = num_detections

# detection_classes should be ints.
detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

label_id_offset = 1
image_np_with_detections = image_np.copy()

# Plotting the detection with pyplot
viz_utils.visualize_boxes_and_labels_on_image_array(
            image_np_with_detections,
            detections['detection_boxes'],
            detections['detection_classes']+label_id_offset,
            detections['detection_scores'],
            category_index,
            use_normalized_coordinates=True,
            max_boxes_to_draw=5,
            min_score_thresh=.8,
            agnostic_mode=False)

# Displaying the image with the detection
plt.imshow(cv2.cvtColor(image_np_with_detections, cv2.COLOR_BGR2RGB))
plt.show()
```

This is the output:



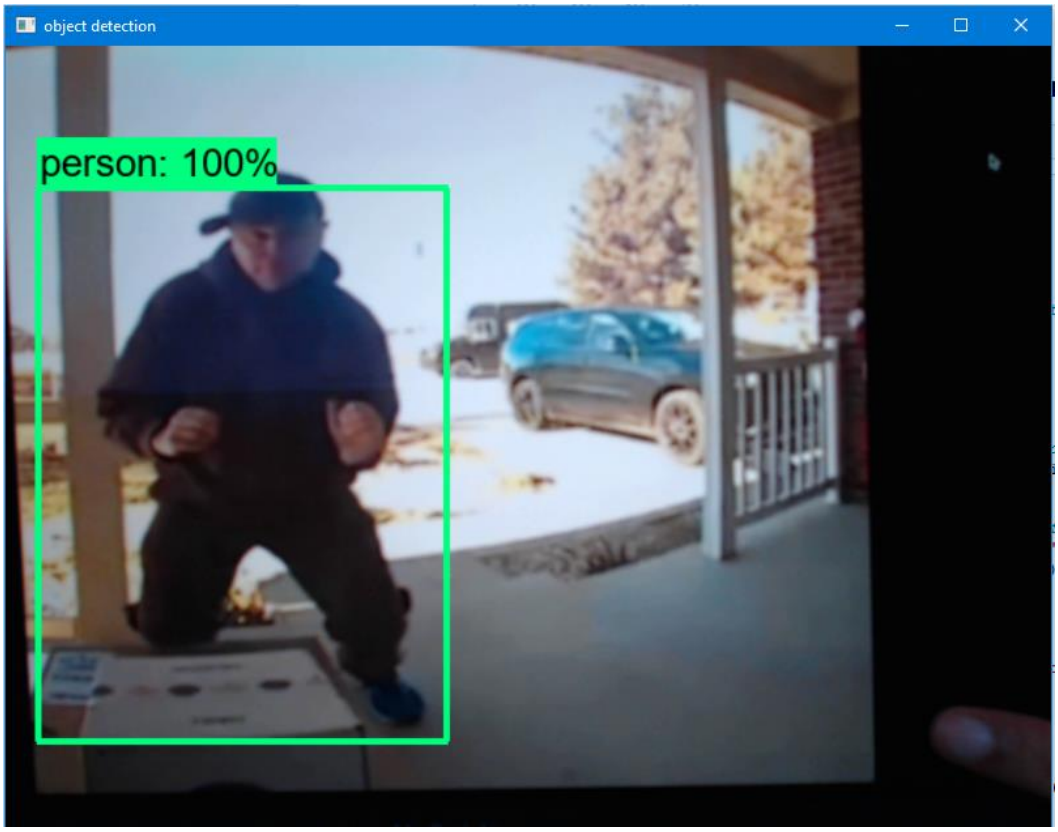
The next part of testing is Realtime testing using a webcam, this is similar to using the model to detect from an image however, I have to use the OpenCV library more to be able to take advantage of the camera, as I have done in stage 1.

In [17]:

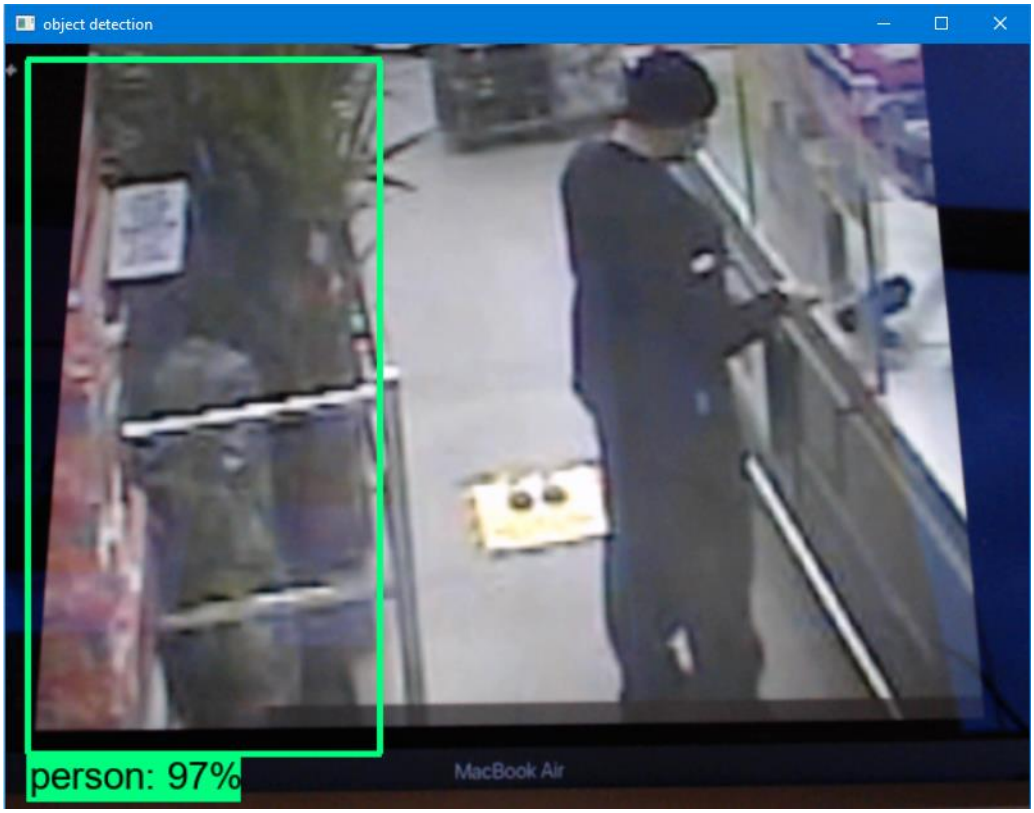
```
# Capturing the video from the camera
cap = cv2.VideoCapture(0)
# Getting the dimensions of the video
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# Starting a loop so that the inference will run till it is exited
while cap.isOpened():
    # Reading the frame from the video input
    ret, frame = cap.read()
    # Converting the frame to a numpy array
    image_np = np.array(frame)

    # Converting the image to a tensor so that the detect_fn function can be used on it
    input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
    detections = detect_fn(input_tensor)

    # Getting the detections and number of detectyions from the dictionary detections
    num_detections = int(detections.pop('num_detections'))
    detections = {key: value[0, :num_detections].numpy()
                  for key, value in detections.items()}
    detections['num_detections'] = num_detections

    # detection_classes should be ints.
    detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

    label_id_offset = 1
    image_np_with_detections = image_np.copy()

    # Creating an image to display
    viz_utils.visualize_boxes_and_labels_on_image_array(
                image_np_with_detections,
                detections['detection_boxes'],
                detections['detection_classes']+label_id_offset,
                detections['detection_scores'],
                category_index,
                use_normalized_coordinates=True,
                max_boxes_to_draw=5,
                min_score_thresh=.8,
                agnostic_mode=False)

    # Displaying the image with the detections and resized
    cv2.imshow('object detection',  cv2.resize(image_np_with_detections, (800, 600)))

    # This is the flag to exit the loop, clicking q exits the loop
    if cv2.waitKey(10) & 0xFF == ord('q'):
        cap.release()
        cv2.destroyAllWindows()
        break
```

Running the testing was partially successful, I was able to get some correct bounding boxes however, they were not always displayed.

An example of successful output:



An example of unsuccessful output:

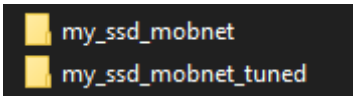


In the successful output, the person was detected with 100% accuracy however, in the unsuccessful output, the shelf was classified as a person with 97% accuracy. This means that the model requires performance tuning as it isn't ready to be used in the web app yet.

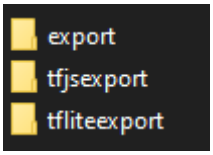
Onto performance tuning, now I am selecting more images and labelling them. The main change I will be making is changing the custom model's name so that a new model is saved, I am now calling it my_ssd_mobnet_tuned:

```
In [3]:  1 # Name of my model
         2 CUSTOM_MODEL_NAME = 'my_ssd_mobnet_tuned'
```

After running the cells again, I now have another folder:



Which has the same folders inside:



As well as adding images, I am bumping the training time up by increasing the number of epochs from 2000 to 3000, both of these combined should hopefully increase the performance of the model.

```
In [40]:  1 command = "python {} --model_dir={} --pipeline_config_path={} --num_train_steps=3000"
```

After running the training script again, this was the final stage output:

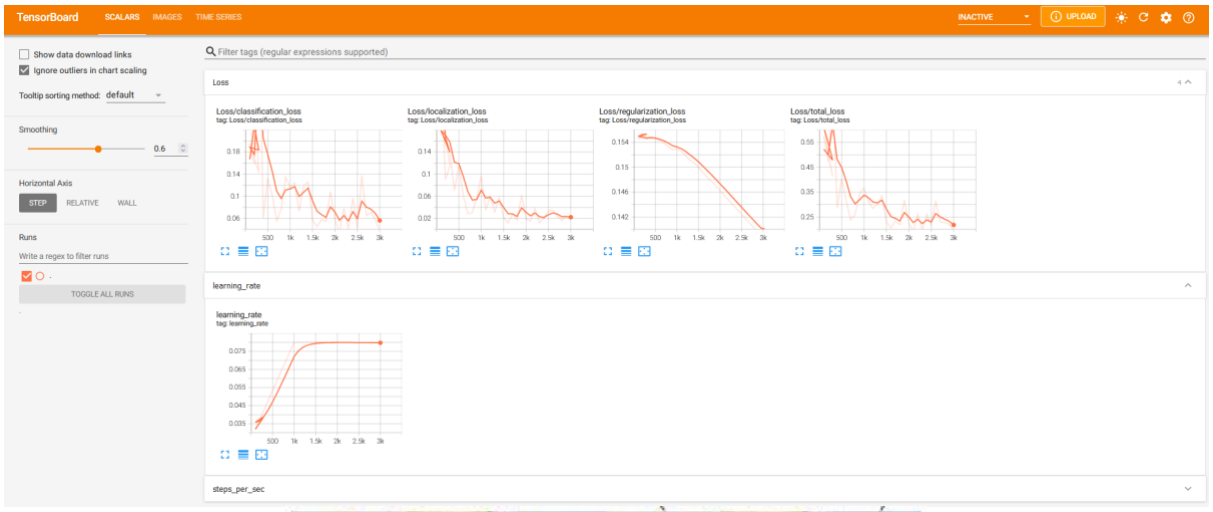
```
INFO:tensorflow:Step 3000 per-step time 0.180s
I0220 20:45:26.596947 23108 model_lib_v2.py:707] Step 3000 per-step time 0.180s
INFO:tensorflow:{'Loss/classification_loss': 0.036574397,
 'Loss/localization_loss': 0.020453712,
 'Loss/regularization_loss': 0.13887617,
 'Loss/total_loss': 0.19590428,
 'learning_rate': 0.0796716}
I0220 20:45:26.600444 23108 model_lib_v2.py:708] {'Loss/classification_loss': 0.036574397,
 'Loss/localization_loss': 0.020453712,
 'Loss/regularization_loss': 0.13887617,
 'Loss/total_loss': 0.19590428,
 'learning_rate': 0.0796716}
```

The loss reached was lower, indicating an improvement in performance.

After running the evaluation script, it also gave a higher precision and recall than before.

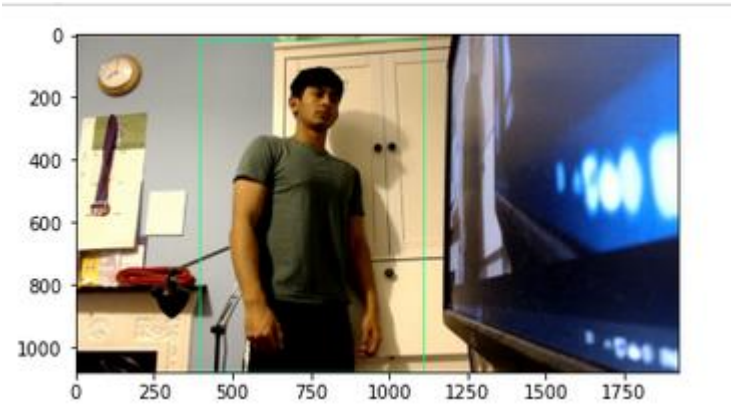
```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.308
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.608
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.410
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.308
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.300
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.300
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.400
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.400
```

Using tensor board to visualize the results:



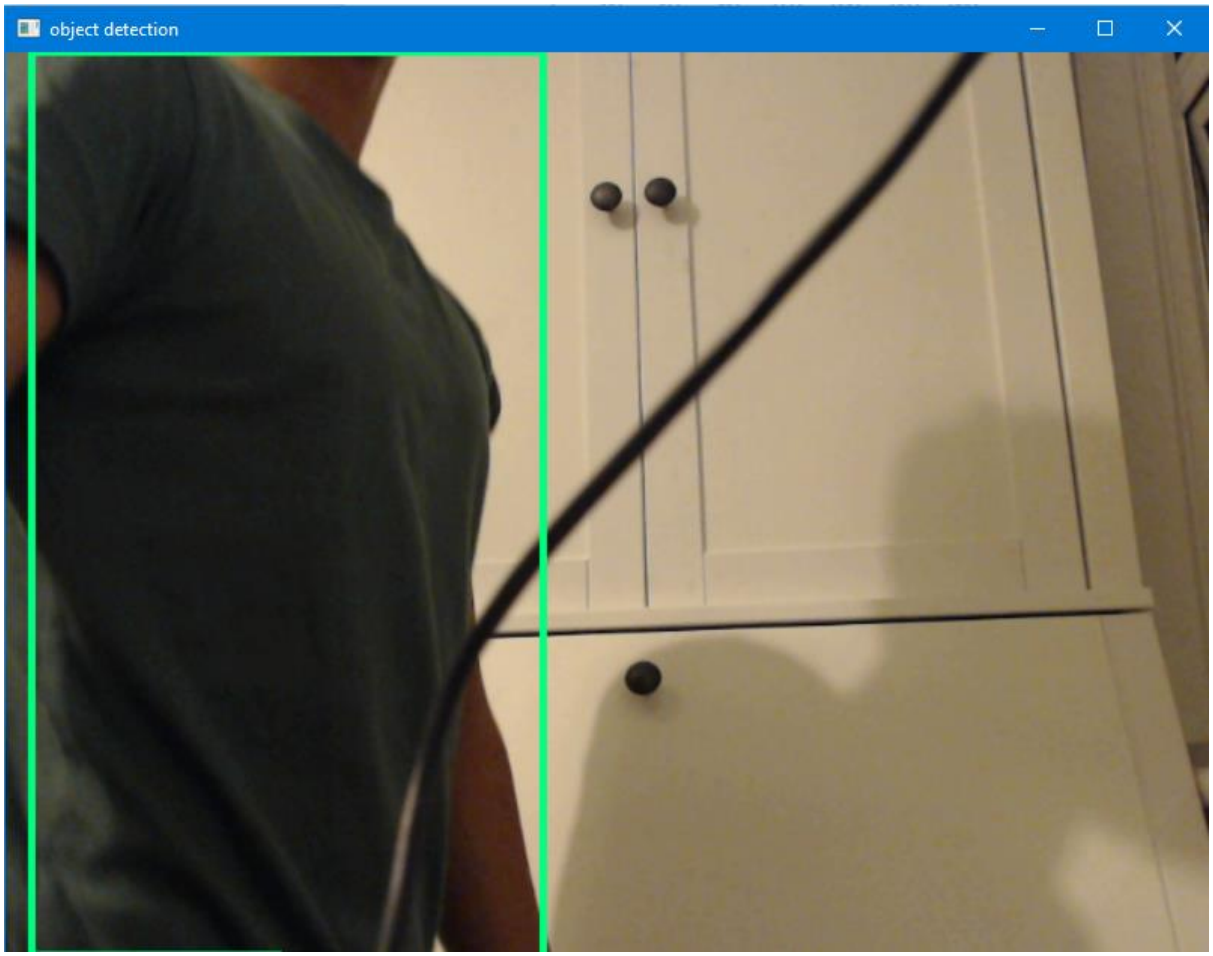
The results seemed much more promising with the model able to detect the people more accurately and producing lower loss and higher accuracy.

When running the image testing, this was the output:



This was a successful detection therefore showing the performance tuning worked and improved the model's performance.

In addition, when running the live detections through the camera, it performed well:



User Feedback

In this stage, I asked the users whether the model would be sufficient, and with the evidence provided, they were happy with it. This stage didn't need significant user feedback as it could mainly be tested by me and the result of this testing was a good indication of the performance of the model which was positive and now since the stakeholders were happy with it, I am ready to move onto the next stage.

Review

In this stage I have:

- Created a model which I can use for people detections
- Checked the model and obtained a result of > 85% accuracy in successful detections

This stage clearly achieves the goal of obtaining an accurate model and which I can make use of when working with the flask app to give accurate detections. Although the initial round of training didn't provide an accurate model, the performance tuning allowed for this after I saw through testing that the model didn't perform as desired.

Addressing the Success Criteria:

1. This stage will contribute to the correct number of people being detected and this can be established in later stages when the model and object counting framework is integrated
2. This is not addressed in this stage
3. The model itself, despite being a deep learning model is fairly lightweight due to its architectures and therefore contributes to the program being lightweight.
4. This is not addressed in this stage
5. This will allow for this to be met in later stages
6. This is not addressed in this stage
7. This is not addressed in this stage

Overall, the criteria that are essentially addressed in this stage are the criteria relating to the detections themselves as they are dependent on the model therefore the model is crucial in that aspect. The model can now be utilized to properly develop the app.

Stage 3 Integrating TensorFlow Object Counting Framework

Code, Explanation, and Justification

To be able to count the number of people in a room, I have chosen to use the TensorFlow Object Counting Framework which cumulatively counts objects using a chosen model. The main component of incorporating it into the solution is integrating it with my current flask code. This will be the main challenge because the flask code must pass the images to the hosting server, using a generator function as opposed to just running straight away in a loop as is done in the TensorFlow Object Counting Framework.

The app is split into different files, some of which are from the Object Counting Framework:

- app.py
- centroid_tracker.py
- trackable_object.py
- The model files
- The label map file

Before anything, I am firstly importing everything I need.

app.py > ...

```
1   # Importing the necessary libraries including flask and openCV
2   from flask import Flask, render_template, Response
3   import cv2
4   import cv2
5   import numpy as np
6   import argparse
7   import tensorflow as tf
8   import dlib
9
10  from object_detection.utils import label_map_util
11  from object_detection.utils import ops as utils_ops
12
13  from trackable_object import TrackableObject
14  from centroidtracker import CentroidTracker
```

Next, I need to make sure that the compatibility issues with TensorFlow 1 and 2 are taken care of which I am doing with the following code:

```
16  utils_ops.tf = tf.compat.v1
17
18  # Patch the location of gfile
19  tf.gfile = tf.io.gfile
```

And next I can begin using my previous flask code, which I have already explained in stage1:

```
app = Flask(__name__)

# Creating variable a camera which will store input from the webcam
cap = cv2.VideoCapture(0)

# Using a decorator and a function to render the html page located in the template folder
@app.route('/')
def index():
    return render_template('index.html')
```

The crucial function now is the generate_frames() function.

This is where the key part of the framework, the cumulative object counting component will be integrated.

I am doing this by taking a key function from it; run_inference() and I will be adapting it to fit the flask framework. It must be integrated into the generate frames function as this is the function that returns the frames to the webpage so the processing of the frame must be done in this function. Previously, I used haar cascades, this is the point at which I can replace it with my model and use the framework to do cumulative object counting so that the correct count is given when getting the room count.

```
def generate_frames(total_frames,model, category_index, cap, labels, roi_position=0.6, threshold=0.5, x_axis=True, skip_frames=20, save_path='',):

    while cap.isOpened():
        ret, image_np = cap.read()
        if not ret:
            break
```

As shown in the above screenshot, the function uses many parameters:

- The first is total_frames
    - This tracks the frames that have passed since the start, it is used later when calculating when to carry out a detection
- The second is category_index
    - This is what is taken from the label_map that will be used for the detection, so it is needed for the model to work and detect people
- The next is cap
    - Short for video capture which is the webcam feed captured by OpenCV so this is what we will be using to get extract the frame which can then be processed.
- The next is labels
    - It is the people label map which is needed to use the correct label when detecting with the model.
- The next is Roi_position
    - It is the positioning of the line which people would have to cross to be detected, this can be adjusted accordingly to where the door or entrance is so that it allows for the count to be correctly given depending on the location.
- The next is threshold
    - This will be used when the model is used to detect people, if they are above the threshold then the detection will count, otherwise the detection will be

disregarded, this ensures that only accurate detections are registered and links to the success criteria to makes sure that the count is given accurately

- The next is x-axis
    - This is a Boolean value and is used to make the detection line that people have to cross to be counted on the x-axis as opposed to the y-axis.
- The final parameter used is the skip-frames parameter
    - This is used so that detections don't have to be carried out every frame as this would be very costly and as a result would take a longer time, even if the model is very efficient. It skips a certain number of frames before detecting again, it essentially increases performance with a slight trade-off with accuracy.
- The last parameter is save_path
    - This would be used to save the video to a particular path however, this is not required in this stage, so it is not used

The last few lines are starting a loop, reading a frame from the webcam feed using OpenCV as before and checking if the frame was correctly read. If it was not correctly read, the loop will be exited.

I am now setting up key variables as well as converting the image to RGB. This is an important part of the inference process as well as the tracking component as both height and width are needed to work with the coordinates when drawing boxes and the status contributes to user experience and usability as it can be outputted to show the user what is currently happening in the program and improves transparency as they know what is going on. The rects list will allow for the different objects being tracked to be updated when detection isn't occurring and it allows new objects to be added when they are detected, the objects of course would be the people with high accuracy.

```
39      # First find the height and width of the image, since these are the only variables needed, an underscore is used for the extra variable
40      height, width, _ = image_np.shape
41
42      # Converts the image to RGB if not already so that it can be worked with easily
43      rgb = cv2.cvtColor(image_np, cv2.COLOR_BGR2RGB)
44
45      # Status is used to display the current action being taken, i.e. waiting, detecting, or tracking
46      status = "Waiting"
47
48      # rects is a list which will hold the detcted boxes
49      rects = []
```

The following screenshot is the first section of the loop and makes use of the skip_frames variable so that detection can be carried out at a certain frame interval.
The trackers variable is also initialised, and this will directly be used by the dlib library to hold the different trackers in use, so the different people detected can be tracked and later on checked for whether they have crossed the line to indicate they have entered or exited a room.

The output_dict is a dictionary which makes use of a certain function called 'run_inference_for_single_image'. This function is already defined and is covered later but for now, the details can be abstracted, and it can be explained simply, it carries out the detection on the current frame, using the created model, and all detections are held in a dictionary format which is saved to the output_dict variable.

```
        # This section changes the status to detecting if the skip frames amount has been reached, using the total and a modulus to check and starts detection
        if total_frames % skip_frames == 0:
            status = "Detecting"
            trackers = []

            # Actual detection for the image given, returns the coordinates of the detections in a dictionary
            output_dict = run_inference_for_single_image(model, image_np)
```

The next part of the detecting section is where the detections carried out are initially processed by comparing their scores to the threshold score given as an argument to the function, if they are above and their label is people then we can move onto the next stage.

Next, I am creating a tracker for each object that the model considers to be a person (above the threshold), as well as a rectangle for the person and the coordinates corresponding to that person in the dictionary are used to create a rectangle and add it to the rectangle list. Then the start_track method can be used to track the person using their rectangle which was created. The tracker is also added to the list of trackers.

```
            # Goes through each coordinate given and checks if the scores is greater than the threshold argument given so that only accurate detections given
            # Also checks for the correct label being detected
            for i, (y_min, x_min, y_max, x_max) in enumerate(output_dict['detection_boxes']):
                if output_dict['detection_scores'][i] > threshold and (labels == None or category_index[output_dict['detection_classes'][i]]['name'] in labels):

                    # A dlib correlation tracker is used to track the specific object which is detected
                    # This is done for each object detected above the threshold and has the label of person
                    tracker = dlib.correlation_tracker()
                    rect = dlib.rectangle(
                        int(x_min * width), int(y_min * height), int(x_max * width), int(y_max * height))
                    tracker.start_track(rgb, rect)
                    trackers.append(tracker)
```

Next, I am moving onto the tracking section, and this is the time between the skip frames, so for example if skip frames parameter is 20, this will include the 1st to the 19th frame. This is where the tracking stage of the program is carried out, so we already have the objects, and we now need to update their positions. This is done through the update method which takes rgb as an argument due to the image being rgb. Now we need to update the rectangle in the rects list, so this is done by getting the now current position and adding it to the rects list.

```
        else:
            status = "Tracking"
            for tracker in trackers:
                # update the tracker and grab the updated position
                tracker.update(rgb)
                pos = tracker.get_position()

                # unpack the position object
                x_min, y_min, x_max, y_max = int(pos.left()), int(
                    pos.top()), int(pos.right()), int(pos.bottom())

                # add the bounding box coordinates to the rectangles list
                rects.append((x_min, y_min, x_max, y_max))
```

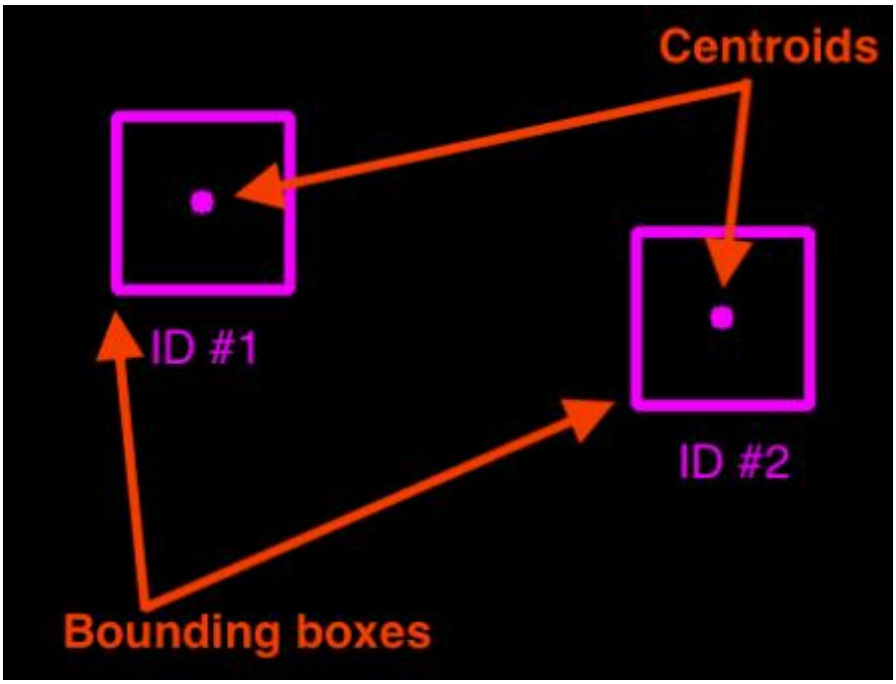
The next section makes use of tracking with centroids:

```
# use the centroid tracker to associate the (1) old object
# centroids with (2) the newly computed object centroids
objects = ct.update(rects)
```

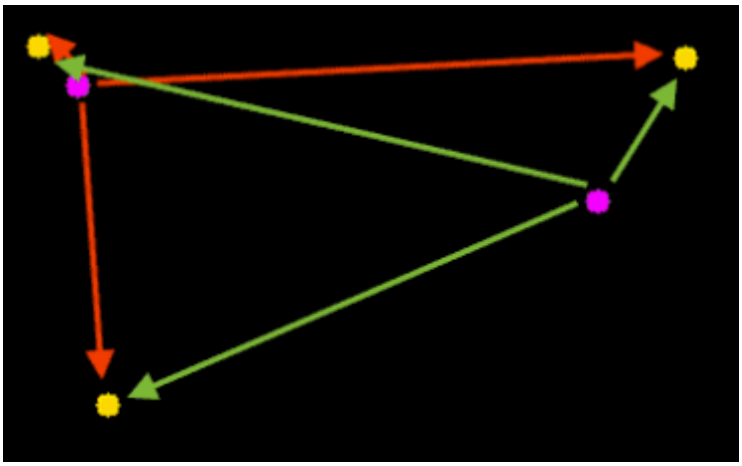
A short explanation of the use of centroids:

Centroids are the centre coordinate of the detected objects. They are what are being used for the object tracking.

This is the first stage in the tracking:

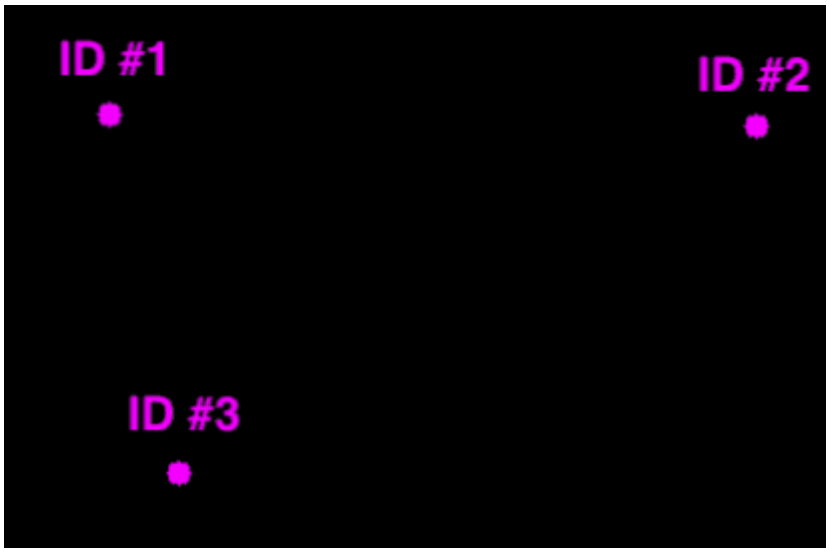


The next stage involves calculating the Euclidean distances between any new centroids and existing centroids.



Once I have the Euclidean distances, I can attempt to associate the object IDs

Any new objects which I don't have a distance for I will assign new IDs:



The next section of code continues off the initial ct.update() line:

It first starts a for loop to iterate over all the objects. Then it checks for any new objects and creates a trackable object for them. Before this, a get method id used to retrieve the trackable objects.

```
89            # loop over the tracked objects
90            for (objectID, centroid) in objects.items():
91                to = trackableObjects.get(objectID, None)
92                # check to see if a trackable object exists for the current object ID
93                if to is None:
94                    # if there is no existing trackable object, create one
95                    to = TrackableObject(objectID, centroid)
96                # otherwise, there is a trackable object so we can utilize it
97                # to determine direction
```

After this, the other situation is considered, i.e., finding which way an object is moving and checking if it has passed our pre-defined line so that the count can be incremented.

```
else:
    # the difference between the x-coordinate (or y-coordinate) of the *current*
    # centroid and the mean of *previous* centroids will tell
    # in which direction the object is moving (negative for
    # 'up' and positive for 'down')

    # Additionally check to see if the object has been counted or not
    if x_axis and not to.counted:
        x = [c[0] for c in to.centroids]
        direction = centroid[0] - np.mean(x)
```

This section of code first if the object has been counted and if the x-axis parameter in this generate_frames() function is true. If it is then it can continue to calculate the mean centroid and a direction for the current object can be calculated using it.

After the direction variable has been calculated for the current object, it can be used to see if the object has passed the ROI line and if so, the counter for that direction can be incremented. The following code specifically checks if the object is moving down and past the line. Also, the counted attribute of the object is set to True so that when the algorithm is run again, it would know that this object has been checked already.

```
# if the direction is positive (indicating the object
# is moving down) AND the centroid is below the
# center line, count the object
if centroid[0] > roi_position*width and direction > 0 and np.mean(x) < roi_position*width:
    counter[1] += 1
    to.counted = True
```

The following section does the same as above but checks for an object moving upwards:

```
# if the direction is negative (indicating the object
# is moving up) AND the centroid is above the center
# line, count the object
elif centroid[0] < roi_position*width and direction < 0 and np.mean(x) > roi_position*width:
    counter[0] += 1
    to.counted = True
```

However, this is all using the x-axis, and if the user has not specific the x-axis to be used then the program wouldn't work which is why it is used again for the y-axis:

```
# Same process above is done if x-axis not specified so y-axis used instead
elif not x_axis and not to.counted:
    y = [c[1] for c in to.centroids]
    direction = centroid[1] - np.mean(y)

    if centroid[1] > roi_position*height and direction > 0 and np.mean(y) < roi_position*height:
        counter[3] += 1
        to.counted = True
    elif centroid[1] < roi_position*height and direction < 0 and np.mean(y) > roi_position*height:
        counter[2] += 1
        to.counted = True
```

And after this is checked, I can add the centroid to the trackable object attribute centroids:

```
to.centroids.append(centroid)
```

And finally, I can get to drawing the lines and circles that will clearly show the detections and tracking:

```
# draw both the ID of the object and the centroid of the
# object on the output frame
text = "ID {}".format(objectID)
cv2.putText(image_np, text, (centroid[0] - 10, centroid[1] - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
cv2.circle(
    image_np, (centroid[0], centroid[1]), 4, (255, 255, 255), -1)
```

This draws both the ID number, as well as the circle on the centroid and this identifies the tracked object clearly to the user.

The next part draws the count and status, the count uses the counter list for left right up and down:

```
# display count and status
font = cv2.FONT_HERSHEY_SIMPLEX
if x_axis:
    cv2.putText(image_np, f'Left: {counter[0]}; Right: {counter[1]}', (
        10, 35), font, 0.8, (0, 0xFF, 0xFF), 2, cv2.FONT_HERSHEY_SIMPLEX)
else:
    cv2.putText(image_np, f'Up: {counter[2]}; Down: {counter[3]}', (
        10, 35), font, 0.8, (0, 0xFF, 0xFF), 2, cv2.FONT_HERSHEY_SIMPLEX)
cv2.putText(image_np, 'Status: ' + status, (10, 70), font,
            0.8, (0, 0xFF, 0xFF), 2, cv2.FONT_HERSHEY_SIMPLEX)
```

And of course, the total_frames variable needs to be incremented:

```
# Add one to the total frames after one frame is processed
total_frames += 1
```

The last part to finish of the generate_frames() function is to encode the image once we have drawn everything on it using openCV. Then I can use yield as previously described in stage 1:

```
# Encode the image so that it can be displayed to the webpage
ret, buffer = cv2.imencode('.jpg', image_np)
frame = buffer.tobytes()
yield (b'--frame\r\n'
       b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
```

The next screenshot includes the video function which was demonstrated in stage 1 but has been amended in stage 2 to accommodate the changes in the generate_frames() function. Namely the inclusion of arguments being passed to the function so that the parameters are used correctly. The arguments passed use the correct variables which are either defined in the main function or given directly as a value with the example of threshold which is given to be 0.5.

```
# Here a decorator is used to start the generator function to allow for the video to be taken and processed and then returned to the webpage
@app.route('/video')
def video():
    return Response(
        generate_frames(detections_made, total_frames,detection_model, category_index, cap, labels="label_map.pbtxt", roi_position=0.6, threshold=0.5, x_axis=True, skip_frames=20),
    mimetype='multipart/x-mixed-replace; boundary=frame')
```

The next function which is implemented in this stage is the inference function which carries out the detections on a single image, given the image and a model as arguments. I have chosen to split the program using a single function specifically for inference on a single image as this is a reusable component. The function needs to be called every time a detection needs to be made so, with the example of skip_frames being equal to 20, This function would be called every 20 frames during the detection section in the generator function. Therefore, using a function here is a sensible and highly useful approach.

The first few lines are concerned with getting the image into a NumPy array format so that it can be converted to a tensor and then a tensor with a batch so that the model can understand it as this form is required for the model to be able to run the inference.

```
def run_inference_for_single_image(model, image):
    image = np.asarray(image)
    # The input needs to be a tensor, convert it using `tf.convert_to_tensor`.
    input_tensor = tf.convert_to_tensor(image)
    # The model expects a batch of images, so add an axis with `tf.newaxis`.
    input_tensor = input_tensor[tf.newaxis, ...]
```

After this, I have everything I need to run the model on the image so this can be carried out:

```
    # Run inference
    output_dict = model(input_tensor)
```

The result produces a dictionary which contains all the detections that were made, I can sort this later as seen in the generator function, using a threshold value to take into account only certain detections which have a required degree of accuracy or ‘confidence’.

Following this, I have to convert the image back to NumPy arrays so that I can work with them later in the program, i.e., in the generator function. I also need to adjust certain

properties of the dictionary so that they are useful in the generator function, i.e., the first num_detections.

```
193     # All outputs are batches tensors.
194     # Convert to numpy arrays, and take index [0] to remove the batch dimension.
195     # We're only interested in the first num_detections.
196     num_detections = int(output_dict.pop('num_detections'))
197     output_dict = {key: value[0, :num_detections].numpy()
198                    for key, value in output_dict.items()}
199     output_dict['num_detections'] = num_detections
```

Additionally, I want to make sure that the detection classes are integers:

```
201     # detection_classes should be ints.
202     output_dict['detection_classes'] = output_dict['detection_classes'].astype(
203         np.int64)
```

The final section in the inference function is designed to handle models with masks which may be useful If I decide to change the model architecture I am using for better performance. This is something that was part of the TensorFlow object counting framework, so it doesn't require editing.

```
205     # Handle models with masks:
206     if 'detection_masks' in output_dict:
207         # Reframe the the bbox mask to the image size.
208         detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
209             output_dict['detection_masks'], output_dict['detection_boxes'],
210             image.shape[0], image.shape[1])
211         dete (variable) detection_masks_reframed: Any
212             detection_masks_reframed > 0.5, tf.uint8)
213         output_dict['detection_masks_reframed'] = detection_masks_reframed.numpy()
```

And finally, once the output_dict dictionary has been edited fully and adjusted it can be returned:

```
215     return output_dict
```

I have also included a separate load_model() function just to make the code more maintainable and easier to understand, so I can abstract some of the details when going over my code (It is easy to understand what load_model() does from the name hence why I have included it as a separate function).

The function takes the model_path as an argument, clears a TensorFlow session and loads the model, then it finally returns the loaded model. The model is loaded in a format that I can use.

```
217    def load_model(model_path):
218        tf.keras.backend.clear_session()
219        model = tf.saved_model.load(model_path)
220        return model
```

The final section in the main app.py file is the main section and is run to initialize are the variables which are required, and any objects needed are instantiated. After everything is essentially setup, the app can be run, and the html page is rendered, and the previously described functions are used to provide the functionality behind the webpage.

```
221  # The flask app itself is run in the following section with the host ip address and port specified
222  if __name__ == '__main__':
223      # Loading the model
224      detection_model = load_model("my_ssd_mobnet_tuned\export\saved_model")
225
226      # Extracting the category_index which is needed for detections
227      category_index = label_map_util.create_category_index_from_labelmap(
228          "label_map.pbtxt", use_display_name=True)
229
230      # The counter list is defined so that the counts of people can be recorded
231      counter = [0, 0, 0, 0]  # left, right, up, down
232
233      # The total frames variable is deinfed to allow skip_frames to work later on
234      total_frames = 0
235
236      # The centroid tracker object is instantiated with specified arguments that will allow for object tracking
237      ct = CentroidTracker(maxDisappeared=40, maxDistance=50)
238
239      # Both the trackers and trackable objects are defined so that they can be added to and worked with later on in the program
240      trackers = []
241      trackableObjects = {}
242
243      # The flask app is run and the host is set using a specified IP and a port is also included
244      app.run(host='192.168.1.81', debug=False, port=9999)
```

The other two files involved in this stage are provided by the TensorFlow Object Counting Framework and are shown in the following screenshots:

The first is the centroidtracker.py file:

It first imports needed modules and libraries including NumPy and SciPy.

```
1  # import the necessary packages
2  from scipy.spatial import distance as dist
3  from collections import OrderedDict
4  import numpy as np
```

Next it creates a class, initializing the attributes needed for a centroid tracker.

```
class CentroidTracker:
    def __init__(self, maxDisappeared=50, maxDistance=50):
        # initialize the next unique object ID along with two ordered
        # dictionaries used to keep track of mapping a given object
        # ID to its centroid and number of consecutive frames it has
        # been marked as "disappeared", respectively
        self.nextObjectID = 0
        self.objects = OrderedDict()
        self.disappeared = OrderedDict()

        # store the number of maximum consecutive frames a given
        # object is allowed to be marked as "disappeared" until we
        # need to deregister the object from tracking
        self.maxDisappeared = maxDisappeared

        # store the maximum distance between centroids to associate
        # an object -- if the distance is larger than this maximum
        # distance we'll start to mark the object as "disappeared"
        self.maxDistance = maxDistance
```

Now it defines several methods which will be needed when handling the centroid tracker. The first of which is register and this adds the new centroid to the objects, increments the next object ID, and changes the disappeared attribute to 0.

```
    def register(self, centroid):
        # when registering an object we use the next available object
        # ID to store the centroid
        self.objects[self.nextObjectID] = centroid
        self.disappeared[self.nextObjectID] = 0
        self.nextObjectID += 1
```

The next method is de-register which removes the object once it has moved off the screen and amends the attributes accordingly.

```
    def deregister(self, objectID):
        # to deregister an object ID we delete the object ID from
        # both of our respective dictionaries
        del self.objects[objectID]
        del self.disappeared[objectID]
```

The next method is update and this is the most important method as it updates the attributes of the current tracker and calculates the new centroids which will be needed for tracking the objects. This function is crucial in allowing the position of the new objects to be found and makes use of the centroid concepts explained previously.

```python
    def update(self, rects):
        # check to see if the list of input bounding box rectangles
        # is empty
        if len(rects) == 0:
            # loop over any existing tracked objects and mark them
            # as disappeared
            for objectID in list(self.disappeared.keys()):
                self.disappeared[objectID] += 1

                # if we have reached a maximum number of consecutive
                # frames where a given object has been marked as
                # missing, deregister it
                if self.disappeared[objectID] > self.maxDisappeared:
                    self.deregister(objectID)

            # return early as there are no centroids or tracking info
            # to update
            return self.objects

        # initialize an array of input centroids for the current frame
        inputCentroids = np.zeros((len(rects), 2), dtype="int")

        # loop over the bounding box rectangles
        for (i, (startX, startY, endX, endY)) in enumerate(rects):
            # use the bounding box coordinates to derive the centroid
            cX = int((startX + endX) / 2.0)
            cY = int((startY + endY) / 2.0)
            inputCentroids[i] = (cX, cY)

        # if we are currently not tracking any objects take the input
        # centroids and register each of them
        if len(self.objects) == 0:
            for i in range(0, len(inputCentroids)):
                self.register(inputCentroids[i])

```

```
        # otherwise, are are currently tracking objects so we need to
        # try to match the input centroids to existing object
        # centroids
        else:
            # grab the set of object IDs and corresponding centroids
            objectIDs = list(self.objects.keys())
            objectCentroids = list(self.objects.values())

            # compute the distance between each pair of object
            # centroids and input centroids, respectively -- our
            # goal will be to match an input centroid to an existing
            # object centroid
            D = dist.cdist(np.array(objectCentroids), inputCentroids)

            # in order to perform this matching we must (1) find the
            # smallest value in each row and then (2) sort the row
            # indexes based on their minimum values so that the row
            # with the smallest value as at the *front* of the index
            # list
            rows = D.min(axis=1).argsort()

            # next, we perform a similar process on the columns by
            # finding the smallest value in each column and then
            # sorting using the previously computed row index list
            cols = D.argmin(axis=1)[rows]

            # in order to determine if we need to update, register,
            # or deregister an object we need to keep track of which
            # of the rows and column indexes we have already examined
            usedRows = set()
            usedCols = set()
```

```
            # loop over the combination of the (row, column) index
            # tuples
            for (row, col) in zip(rows, cols):
                # if we have already examined either the row or
                # column value before, ignore it
                if row in usedRows or col in usedCols:
                    continue

                # if the distance between centroids is greater than
                # the maximum distance, do not associate the two
                # centroids to the same object
                if D[row, col] > self.maxDistance:
                    continue

                # otherwise, grab the object ID for the current row,
                # set its new centroid, and reset the disappeared
                # counter
                objectID = objectIDs[row]
                self.objects[objectID] = inputCentroids[col]
                self.disappeared[objectID] = 0

                # indicate that we have examined each of the row and
                # column indexes, respectively
                usedRows.add(row)
                usedCols.add(col)
```

```
            # compute both the row and column index we have NOT yet
            # examined
            unusedRows = set(range(0, D.shape[0])).difference(usedRows)
            unusedCols = set(range(0, D.shape[1])).difference(usedCols)

            # in the event that the number of object centroids is
            # equal or greater than the number of input centroids
            # we need to check and see if some of these objects have
            # potentially disappeared
            if D.shape[0] >= D.shape[1]:
                # loop over the unused row indexes
                for row in unusedRows:
                    # grab the object ID for the corresponding row
                    # index and increment the disappeared counter
                    objectID = objectIDs[row]
                    self.disappeared[objectID] += 1

                    # check to see if the number of consecutive
                    # frames the object has been marked "disappeared"
                    # for warrants deregistering the object
                    if self.disappeared[objectID] > self.maxDisappeared:
                        self.deregister(objectID)

            # otherwise, if the number of input centroids is greater
            # than the number of existing object centroids we need to
            # register each new input centroid as a trackable object
            else:
                for col in unusedCols:
                    self.register(inputCentroids[col])

        # return the set of trackable objects
        return self.objects
```

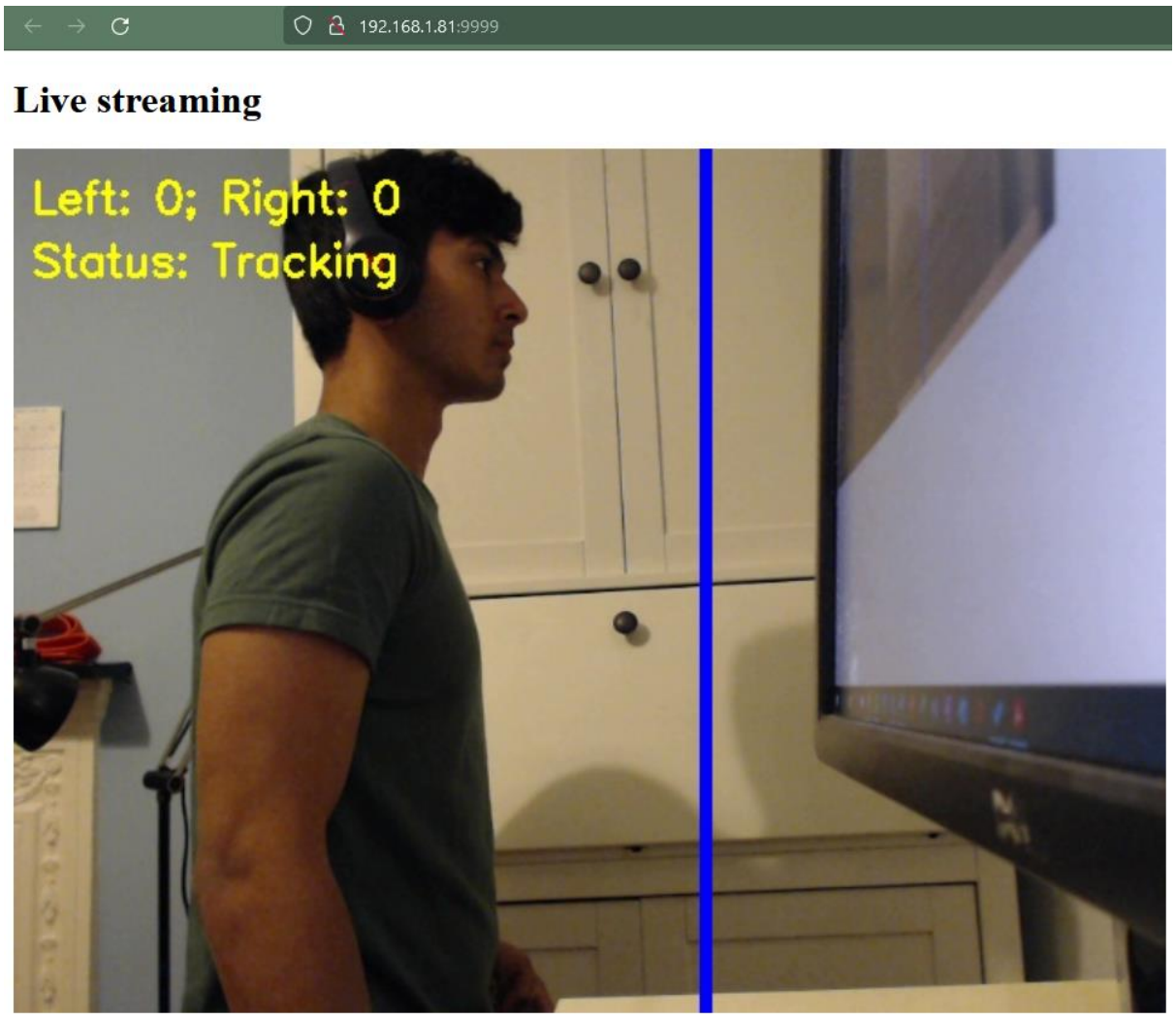
The last file is the trackable object file which again creates a class from which I can create objects that represent a person in the image which is to be tracked across the screen:

```
class TrackableObject:
    def __init__(self, objectID, centroid):
        # store the object ID, then initialize a list of centroids
        # using the current centroid
        self.objectID = objectID
        self.centroids = [centroid]

        # initialize a boolean used to indicate if the object has
        # already been counted or not
        self.counted = False
```

Testing

When I was running the code initially, the webpage ran fine, the window was displaying properly with the output however, the detections were not be shown on the output window. This suggested to me that the model wasn't working correctly.

The following output showed that detection was occurring as it should, and this prevented the correct count being displayed.



To identify the error, I began outputting different variables and checking the values that they were giving to make sure that they were what was expected.

I began by adding a print statement after the model has been loaded to make sure it is being loaded correctly:

```
229     # Loading the model
230     detection_model = load_model(cutom_model_path)
231
232     print("Model successfully loaded")
```

This gave output as expected:

```
Model successfully loaded
```

Next, I moved onto testing the generate_frames function as this is where the detections themselves were being made.

I suspected that the error was to do with the output_dict and how it was being used. To test this, I printed the output_dict:

```
38          output_dict = run_inference_for_single_image(model,image_np)
39          print(output_dict)
```

However, the output it gave wasn't useful as there was too much information in output_dict:

```
 [-0.01597979,  0.29373702,  0.05250077,  0.43754098],
 [-0.02989947,  0.27998227,  0.06599656,  0.45122015],
 [-0.04398878,  0.32906035,  0.08736855,  0.39965972],
 [-0.0662495 ,  0.3216697 ,  0.10682211,  0.40644163],
 [-0.03208226,  0.3411932 ,  0.07447114,  0.4312746 ],
 [-0.03935447,  0.3265659 ,  0.08588912,  0.4496732 ],
 [-0.01546887,  0.3207835 ,  0.05395597,  0.46296668],
 [-0.03021548,  0.30879575,  0.06766789,  0.4763242 ],
 [-0.04209997,  0.35481468,  0.09074892,  0.4241958 ],
 [-0.06443459,  0.3477519 ,  0.11088623,  0.43133208],
 [-0.03175384,  0.36607212,  0.07435793,  0.4557172 ],
 [-0.03952294,  0.35129496,  0.08564445,  0.47397003],
 [-0.01584743,  0.3456837 ,  0.05433583,  0.4883057 ],
 [-0.0304361 ,  0.33299053,  0.06785971,  0.50220066]],
dtype=float32), 'detection_scores': array([0.05436487, 0.04770392, 0.04677331, 0.04153563, 0.0365514 ,
 0.03556569, 0.02905375, 0.02805502, 0.02794542, 0.02672602,
 0.02522665, 0.02424818, 0.02395738, 0.0234915 , 0.02328634,
 0.02299121, 0.02264811, 0.02214008, 0.02144326, 0.02142025,
 0.02133444, 0.02125887, 0.02099541, 0.02099395, 0.0203767 ,
 0.02032666, 0.02015057, 0.02010343, 0.02004553, 0.02000069,
 0.01951102, 0.01944756, 0.01893333, 0.01860135, 0.01852247,
 0.0180961 , 0.01791094, 0.01780709, 0.01766558, 0.01746451,
 0.01744849, 0.01736665, 0.01730367, 0.0172581 , 0.01708918,
 0.01704963, 0.01703395, 0.01702371, 0.01664678, 0.01658665,
 0.01645765, 0.01644177, 0.01624613, 0.01622173, 0.01610613,
 0.01600535, 0.01599908, 0.01567   , 0.01556959, 0.01547031,
 0.01542782, 0.01540357, 0.01537945, 0.01517806, 0.01508667,
 0.01500202, 0.01492689, 0.01490445, 0.01468447, 0.01459808,
 0.01448983, 0.01441709, 0.01435609, 0.01429467, 0.0142849 ,
 0.01411067, 0.0140983 , 0.01408212, 0.01407939, 0.01401985,
 0.01397816, 0.01387284, 0.01383482, 0.01377812, 0.01368239,
 0.01368017, 0.01362728, 0.01351381, 0.01351096, 0.01345459,
 0.01344934, 0.01343003, 0.0133595 , 0.01335715, 0.01327451,
 0.01324322, 0.0132299 , 0.01312568, 0.01311067, 0.01307725],
dtype=float32), 'detection_classes': array([2, 1, 2, 2, 2, 1, 1, 2, 4, 4, 3, 3, 3, 4, 1, 4, 1, 1, 3, 3, 1, 4,
 4, 2, 4, 4, 4, 1, 1, 4, 4, 1, 4, 4, 4, 1, 4, 1, 1, 2, 4, 4, 1, 4,
 1, 1, 4, 4, 2, 1, 1, 4, 4, 4, 1, 4, 1, 1, 1, 1, 1, 4, 1, 1, 2, 4,
 1, 4, 1, 1, 1, 1, 1, 2, 3, 1, 1, 1, 2, 1, 1, 1, 4, 1, 1, 1, 1, 1,
 1, 4, 1, 1, 1, 4, 4, 3, 1, 3, 4, 4, 1], dtype=int64), 'num_detections': 100}
```

To give more concise output that would be helpful, instead of printing the whole detections dictionary, I decided to look into the for loop and use an if condition to print only the detection scores if they were greater than a small threshold (0.1). This would tell me if

detections were being made correctly as if an object (a person) was in front of the camera then the mores scores would be printed above the threshold.

```
output_dict = run_inference_for_single_image(model, image_np)
for i in output_dict['detection_scores']:
    if i > 0.1:
        print(i)
```

This is the output it gave me:

```
0.22681704
0.17062652
0.13984971
0.11666239
0.4264259
0.21973747
0.31835815
0.21723285
0.114170514
```

This indicates that detections are in fact being made correctly as there are some numbers which are fairly high.

Looking further into the generate_frames function I found an if condition which caused the code within in to not be executed. This was the if condition:

```
for i, (y_min, x_min, y_max, x_max) in enumerate(output_dict['detection_boxes']):
    if output_dict['detection_scores'][i] > threshold and (labels == None or category_index[output_dict['detection_classes'][i]]['name'] in labels):
```

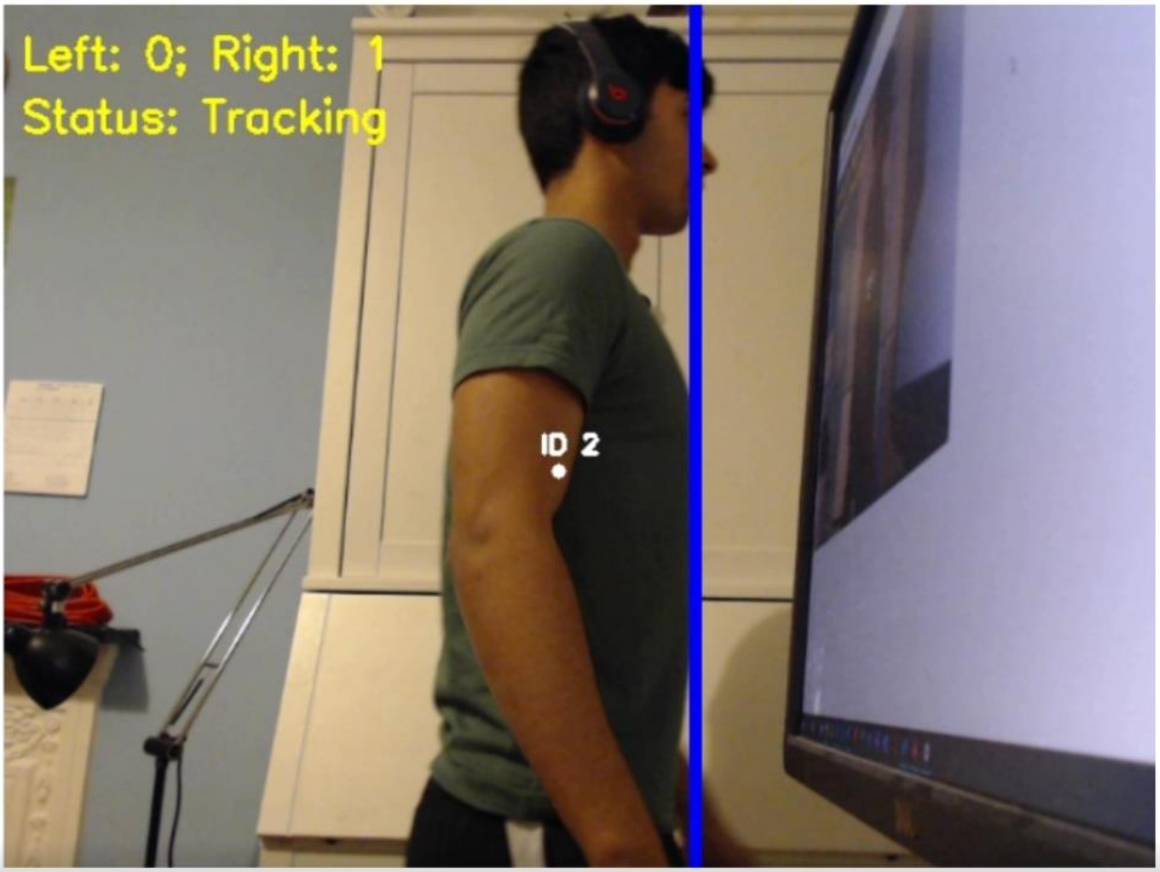
This was under the detection section of the generate frame's function where the error seemed to be coming from.

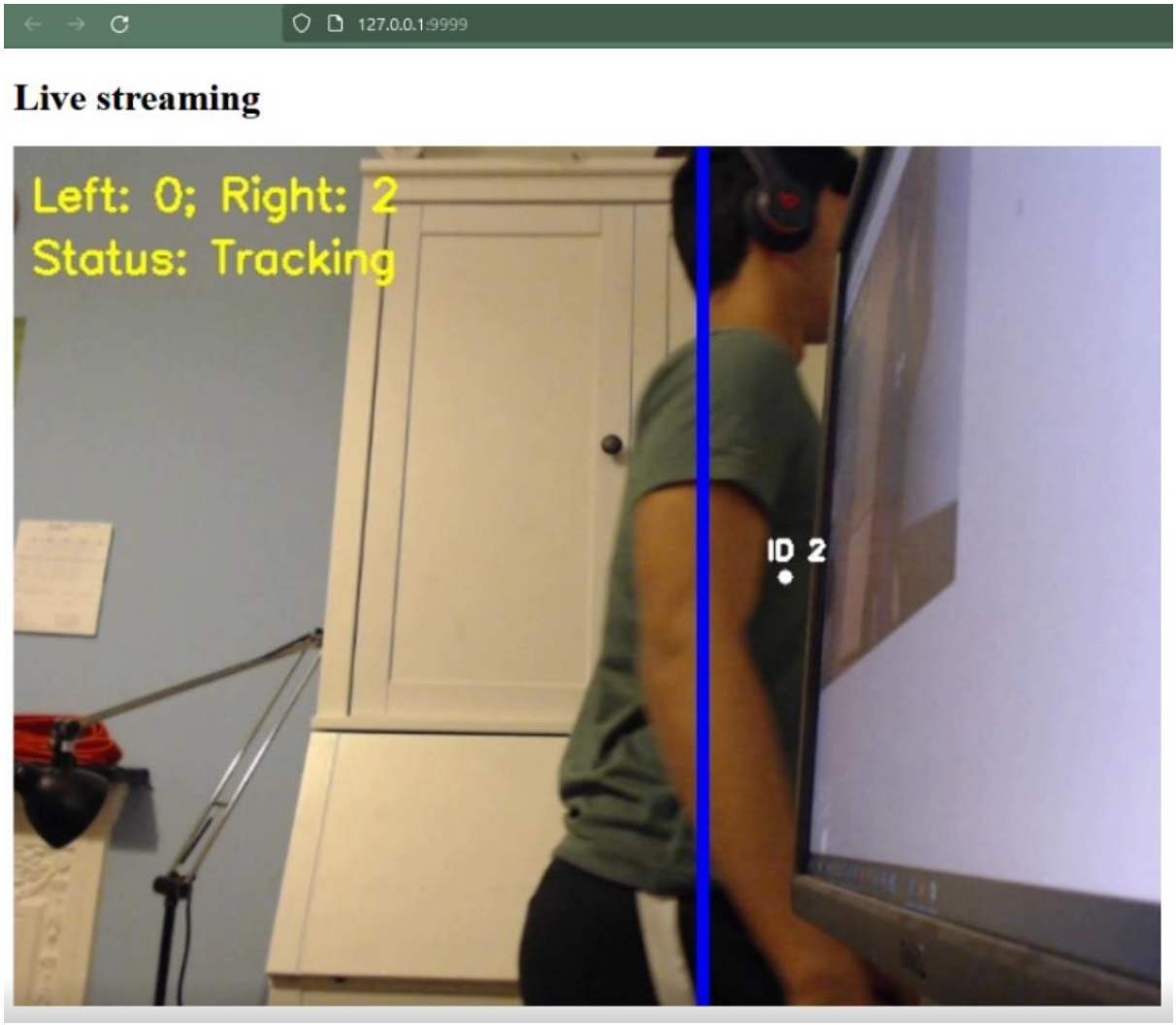
Since I had already tested the first condition, I looked at the second condition. I was unsure about whether it was needed since the threshold was the main condition and I only had one class (person) so the second part of the condition may be unnecessary. Removing it was my next step.

This was the output:

127.0.0.1:9999

Live streaming





The output shows that not only were detections working but the count was also working correctly.

Review

In this stage I have:

- Integrated the object counting framework with the basic flask app
- Made use of the model

This stage takes the flask app and builds on it to allow for the model to be used and implements the counting framework so that cumulative counting can be carried out.

Addressing the Success Criteria:

1. This is achieved in this stage, using the cumulative counting to achieve this along with the model
2. This is not addressed in this stage
3. The website remains lightweight
4. This is done in this stage
5. Dots with IDs are used instead of bounding boxes which achieve the same outcome but are more minimalistic and still easy to identify. They don't clutter the feed.
6. This is not addressed in this stage
7. This is not addressed in this stage

Overall, in this stage, the counting aspect is achieved and can therefore be used properly, however there are aspects which can be refined such as the UI on the webpage which will be addressed in later stages.

Stage 4 – Adding Start, Stop, Save Buttons

Code, Explanation and Justification + Testing

The main thing I am adding is a show feed button so that the detection can start when the user wants it to rather than opening up straight away, along with this I will be adding a hide feed button and a save detections button.

The aim of this stage is to introduce buttons that will allow for detection to be started and stopped and save the counts. For this I am going to create both a start detection and stop detection button first, then create the save count button

This was this original html:

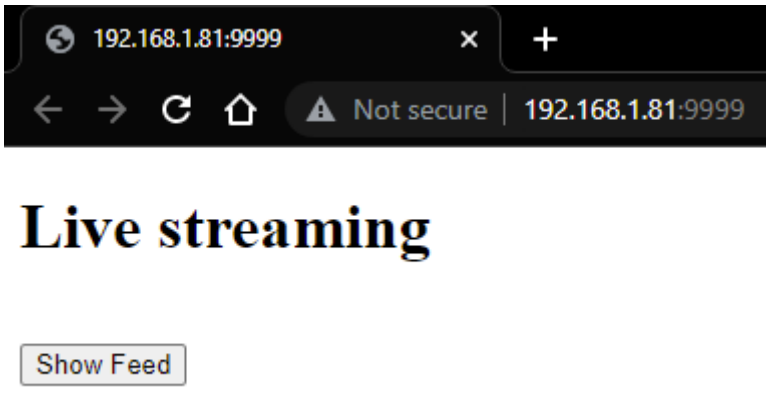
```
<!DOCTYPE html>
<html>
  <body>
    <h1>Live streaming</h1>
    <div>
        <img src="{{ url_for('video') }}" width="50%"/>
    </div>
    </body>
</html>
```

This is how I have changed it to add the start detection button:

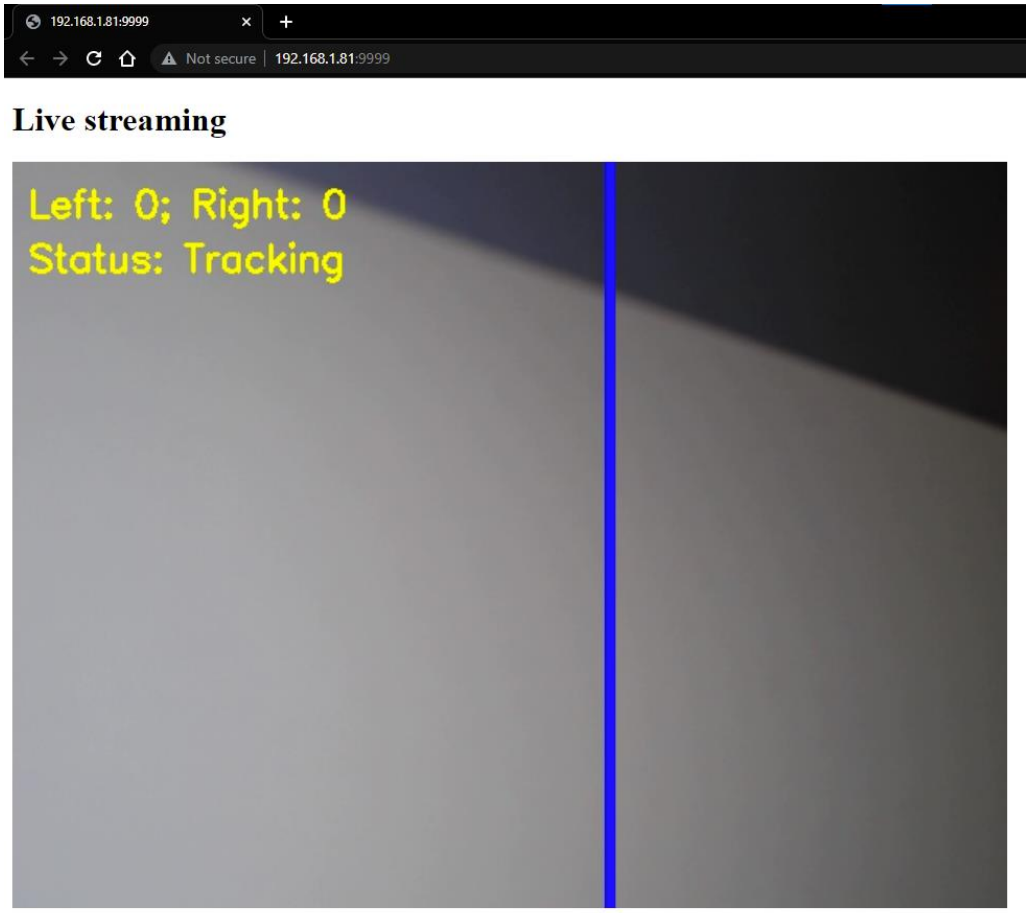
```html
<!DOCTYPE html>
<html>
  <body>
    <h1>Live streaming</h1>
    <div>
        <img id="image" src="" width="50%"/>
    </div>
    <button type="button"
        onclick="show()" id="btnID">
        Show Feed
    </button>
    <script>
        function show() {

            /* Get image and change value
            of src attribute */
            let image = document.getElementById("image");

            image.src = "{{ url_for('video') }}"

            document.getElementById("btnID")
                    .style.display = "none";
        }
    </script>

    </body>
</html>
```

In the code I am implementing a button and added ids. I have also added JavaScript that allows me add functionality to the button. On clicking the button, the image is displayed and since the image is where the detections take place, this will start the detections.

This is the webpage before the button is clicked:



This is the webpage after the button is clicked:



This shows that the button functions correctly. However, before adding the stop detections button, I realized that if the user wanted to start the detections again, the button to start would have been already hidden, therefore I removed the code to remove the start detection button after it had been clicked so the index file became this:

index_testing.html > ...

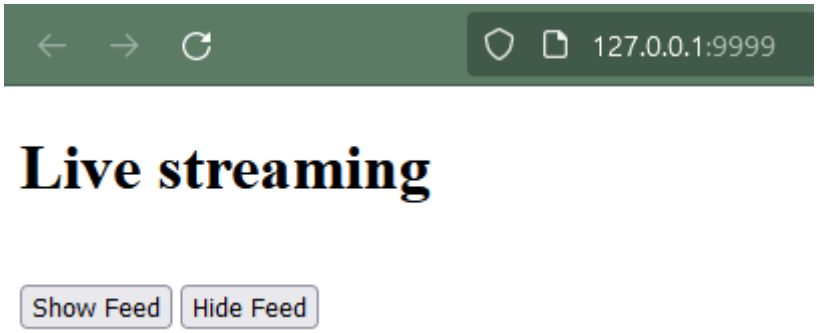
```
<!DOCTYPE html>
<html>
  <body>
    <h1>Live streaming</h1>
    <div>
        <img id="image" src="" width="50%"/>
    </div>
    <button type="button"
        onclick="show()" id="btnID">
        Show Feed
    </button>
    <script>
        function show() {

            /* Get image and change value
            of src attribute */
            let image = document.getElementById("image");

            image.src = "{{ url_for('video') }}"

            /* Code for hiding show image button (removed):
            document.getElementById("btnID")
                    .style.display = "none";*/
        }
    </script>

    </body>
</html>
```

Next, I am going to move onto adding the hide feed button. Doing this is similar to adding the show feed button. I am creating another button with a different ID and name. I am also creating another function in the script section in JavaScript called hide, this changes the image source to an empty string instead of the URL that would get the video and carry out detections.

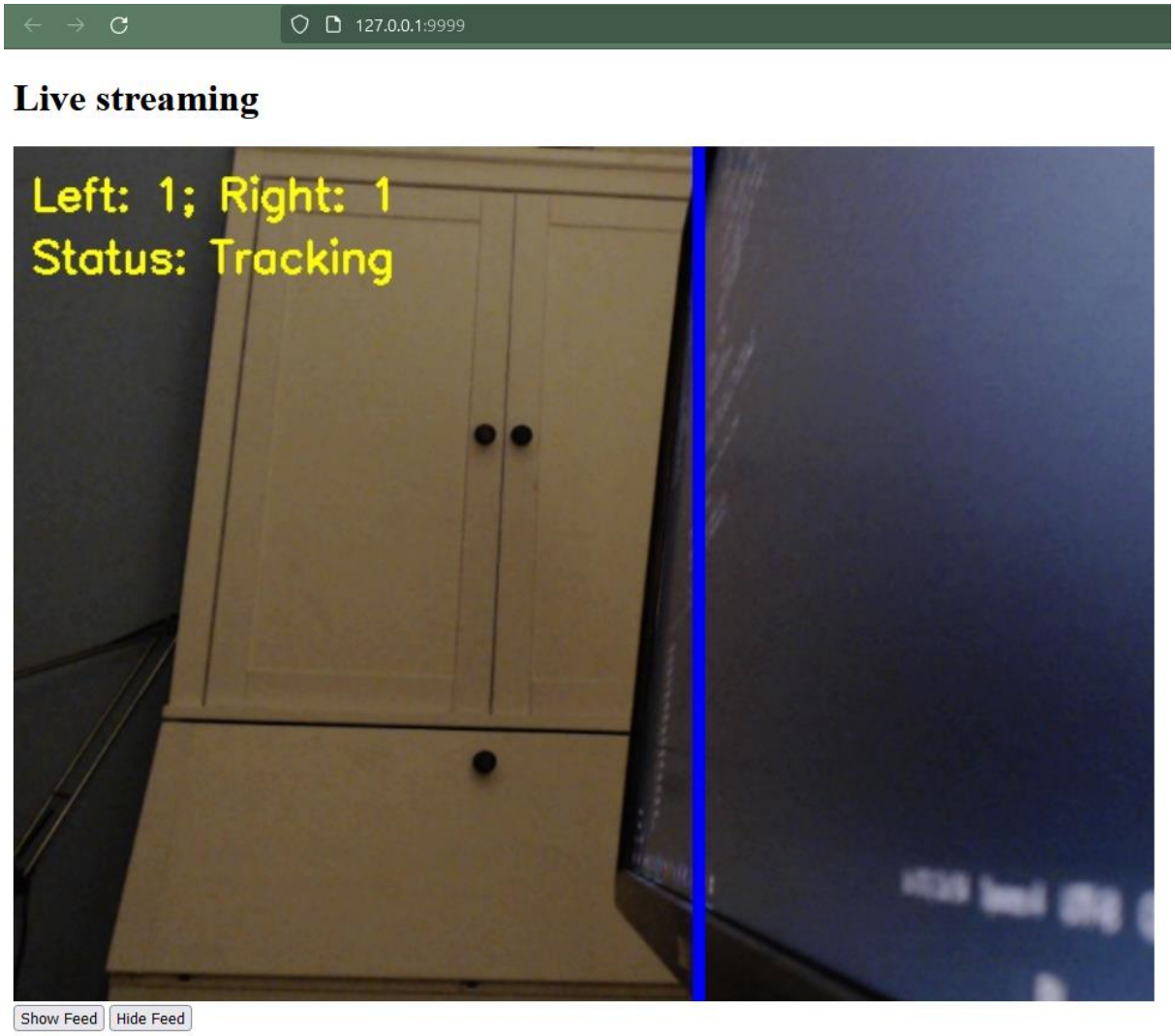
templates > index.html > ...

```html
<!DOCTYPE html>
<html>
  <body>
    <h1>Live streaming</h1>
    <div>
        <img id="image" src="" width="50%"/>
    </div>
    <button type="button"
        onclick="show()" id="btnID">
        Show Feed
    </button>
    <button type="button"
        onclick="hide()" id="btn2ID">
        Hide Feed
    </button>
    <script>
        function show() {

            /* Get image and change value
            of src attribute */
            let image = document.getElementById("image");

            image.src = "{{ url_for('video') }}"

            /* Code for hiding show image button:
            document.getElementById("btnID")
                    .style.display = "none";*/
        }
        function hide() {

            /* Get image and change value
            of src attribute */
            let image = document.getElementById("image");

            image.src = ""

        }
    </script>

    </body>
</html>
```

The webpage initially looked like this after the implementation of the buttons:



Once the show feed button was clicked, this was what the webpage looked like:



With the implementation of these buttons, the user is now able to start and stop detection whenever they want, i.e., when the hide feed button is pressed, detection is paused, then when the user wants to start detection again, the show feed button can be pressed, and it continues detections and displaying the feed.

The last button I am implementing is the save detections button. First, I am adding another button to the html as before:

```
16      <button action="/save_result" id="btn3ID">
17          Save Count
18      </button>
```

It has action equal to a function.

I am not implementing this function in the main python file:

```
301  # Creating a function to save the net counts
302  @app.route('/save_result')
303  def save_result(net_left_count, net_right_count):
304      # Opens file and writes required text
305      with open("Saved_counts.txt", 'w') as f:
306          f.write("Net left count: ")
307          f.write(net_left_count)
308          f.write("\n")
309          f.write("Net right count: ")
310          f.write(net_right_count)
```

The net_left_count and net_right_count variables have been created in the generate frames function so that a net count is given rather than a left and right count, as this is more helpful as it would give one of these as a positive value for the number of people in a room.

```
net_left_count = counter[0] - counter[1]
net_right_count = counter[1] - counter[0]
```

When run, a text file is created which the user can check, here is an example of the text file that is created:

```
1   Net left count: 3
2   Net right count: -3
```

Now I have added all the buttons planned in this stage.

User feedback

After giving it to the different users, I asked the following questions:

1) What do you think of the app at its current stage?
2) Does it meet your requirements?
3) Are there any elements you would like to see additionally?

Responses:

Teacher:

1) The app has developed greatly since the first iteration, and I am glad to see the detections working well.
2) Yes, I am able to see the count in and out as well as the feed, it is simple to use and there isn't any difficulty in using it. Also, I am able to save the count however, there is no further work on this in terms of analysing it directly.
3) There isn't however a total count in either direction which may be more useful than just the count left and right. Also, the page could be made more appealing.

Shop Owner:

1) It looks to be functioning correctly with the basic features implemented.
2) It does except for needing some further analysis on the data.
3) Some further analysis on the data may be useful.

Society organizer:

1) The app has come along well, and it works correctly.
2) It does, and it isn't difficult to use.
3) Possibly an improvement in the appearance of the webpage may be something to work on.

Overall, the user feedback seemed to indicate an improvement over the previous iterations however some things were pointed out to be addressed such as analysis and improving the appearance of the page. As a result of this, I will move onto refining the appearance of the webpage with CSS. In terms of analysis, this is the last thing I will consider in the given timeframe as the core aspects of the project are addressed, this would be something extra if I get the opportunity to implement.

Review

In this stage I have:

- Added the start detection button which I could use to see how to add other buttons
- Added the hide feed button
- Added the save count button

This stage allowed me to keep the app lightweight while providing useful features that the stakeholders would benefit from.

Addressing the Success Criteria:

- During this stage criteria 2 is addressed, giving the option to now activate the feed via a button press as well as hide the feed and save the count

In this stage, the html of the app was worked on, and JavaScript was implemented and shows what can be done with buttons and allows them to be seen in practise. Also, additional functions were added, one in python to allow for the saving of the count.

Stage 5 – Refining User Interface

Code, Explanation and Justification + Testing

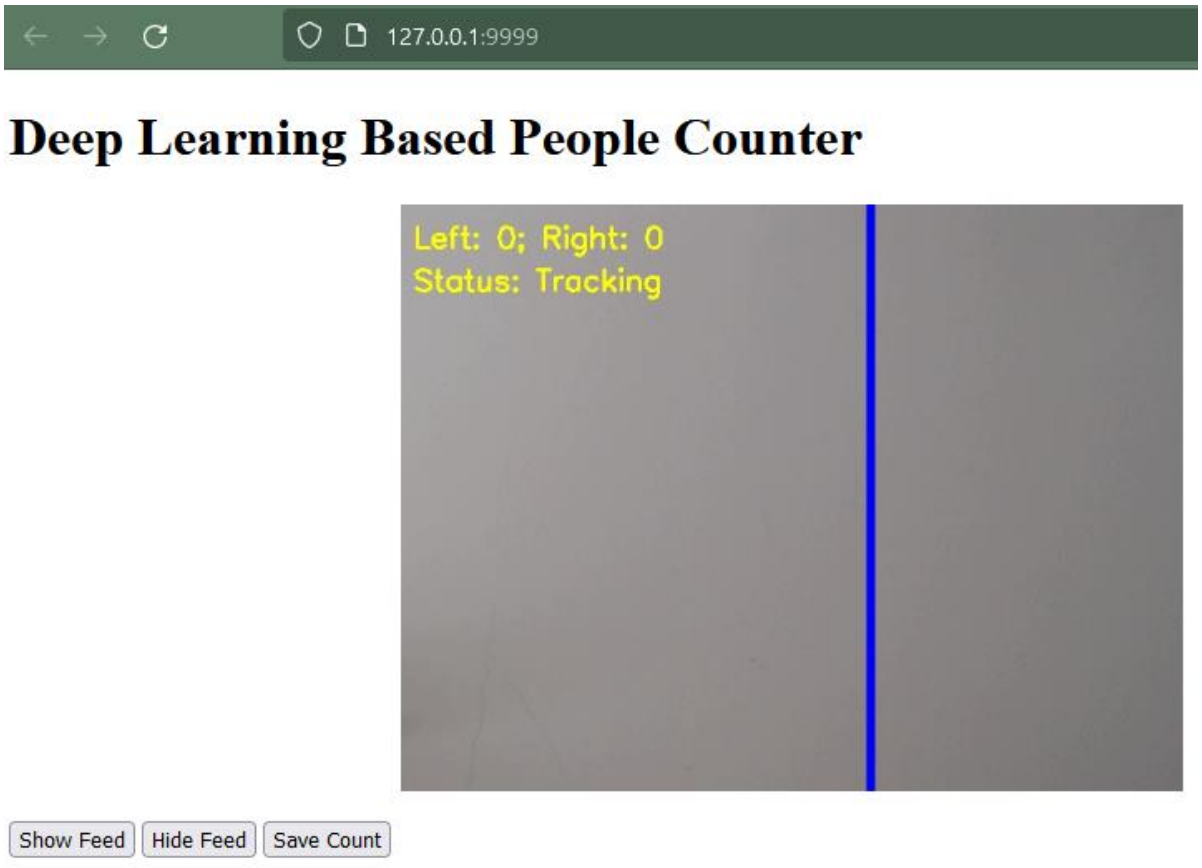
I am starting by linking an external CSS file to the html:

```
<head>
    <link rel="stylesheet" href="styling.css">
</head>
```

Next, I am adding some styling for the video feed, first addressing it's positioning on the page, making it centre, I also changed the title to 'Deep Learning Based People Counter. The reason I am putting the feed at the centre as this will make it the focus, which it needs to be as it is the core of the project.

```
#image {
    display: block;
    margin-left: auto;
    margin-right: auto;
    width: 50%; }
```

This made the webpage look like this:



Next, I am adding a border with a colour to make the feed stand out more on the page.

```
    border: 5px solid;
    border-color: rgb(48, 59, 59);
}
```

In addition to this, I am going to style the buttons, so they are in the middle with the image as otherwise it would look unappealing with different elements located in different places:

```
input[type="button"]{
    background-color: aliceblue;
    display: block;
    margin-left: auto;
    margin-right: auto;
    width: 10%;
}
```

However, when I run the output, it is the same as before, with the feed border but the buttons haven't changed:

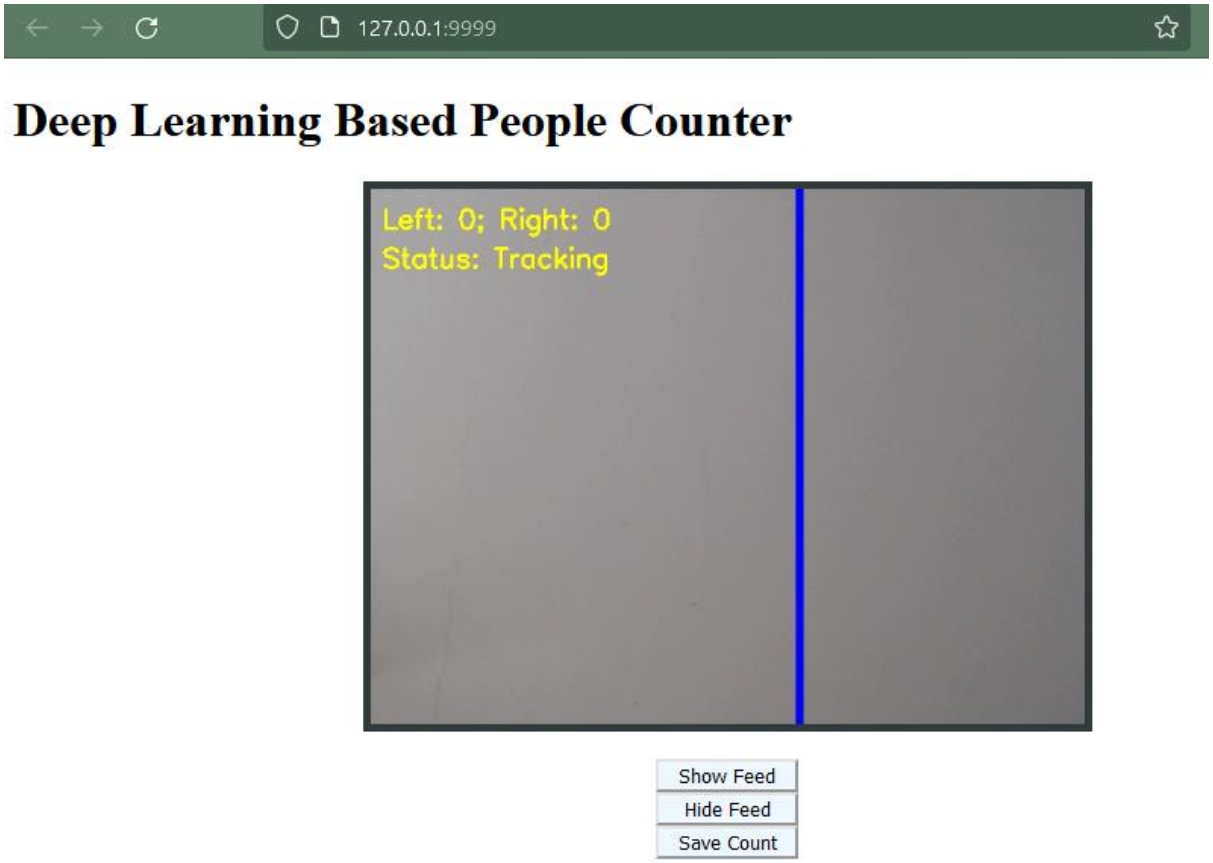
Deep Learning Based People Counter

Show Feed | Hide Feed | Save Count

This means that the styling isn't correctly applied to the buttons. After researching the issue, I found that all the different button types had to be included in the CSS so that the styling was applied to all button types.

```
button, input[type="button"], input[type="submit"]{
```

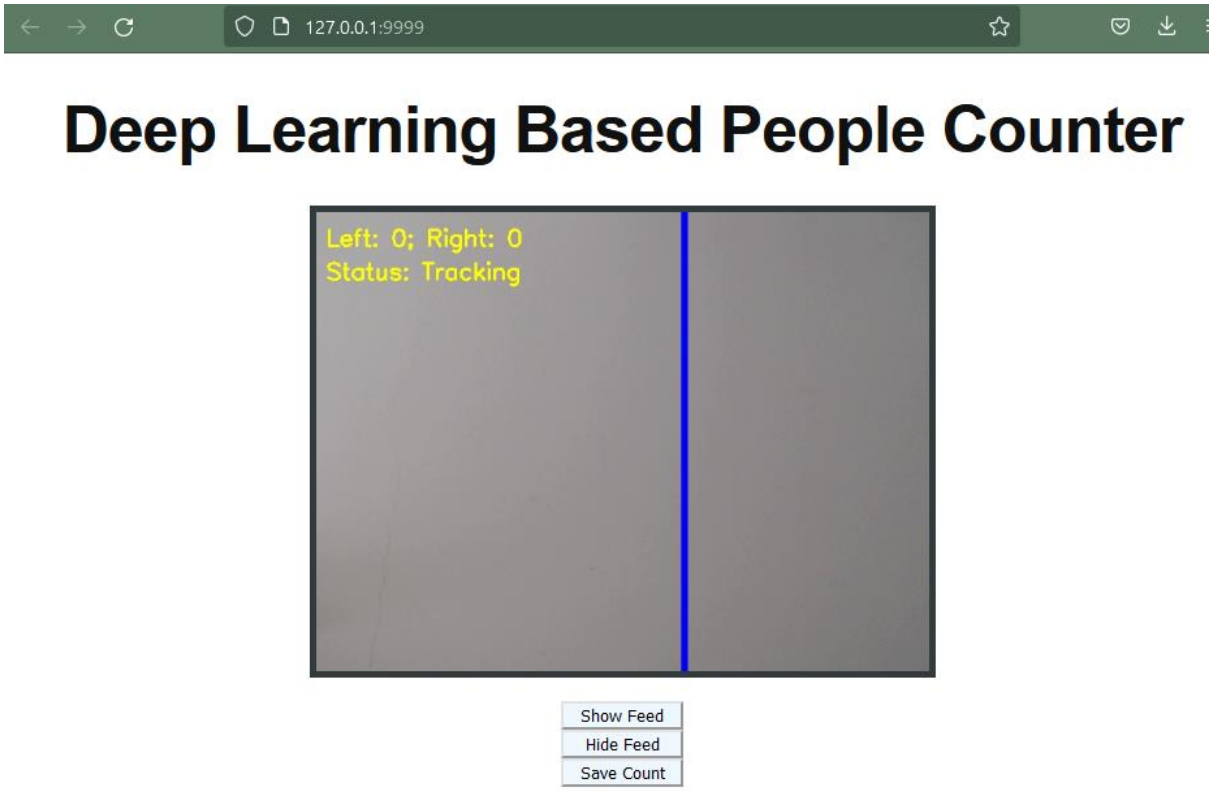
This fixed the issue:



Now I am moving on to style the H1 tag so that the title stands out more and the purpose of the page is clear:

```
h1 { color: #111;
    font-family: 'Helvetica Neue', sans-serif;
    font-size: 50px;
    font-weight: bold;
    letter-spacing: -1px;
    line-height: 1;
    text-align: center;
}
```

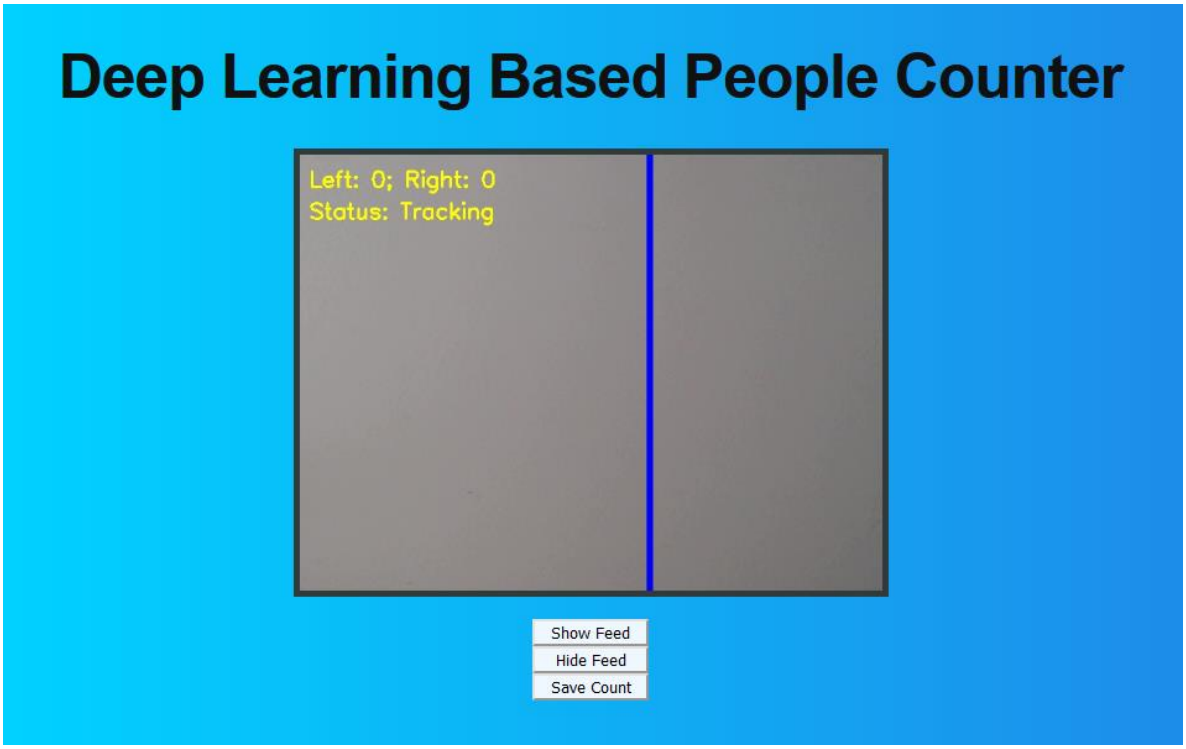
This is the output:



Finally, I am adding a background image to finish up the styling. The reason I am adding this as it will make the page stand out more rather than having a plain white background and this will make it more appealing to look at for users:

```
body {
    background-image: url('img/aqua.png');
}
```

Now with all the styling, this is the final webpage:



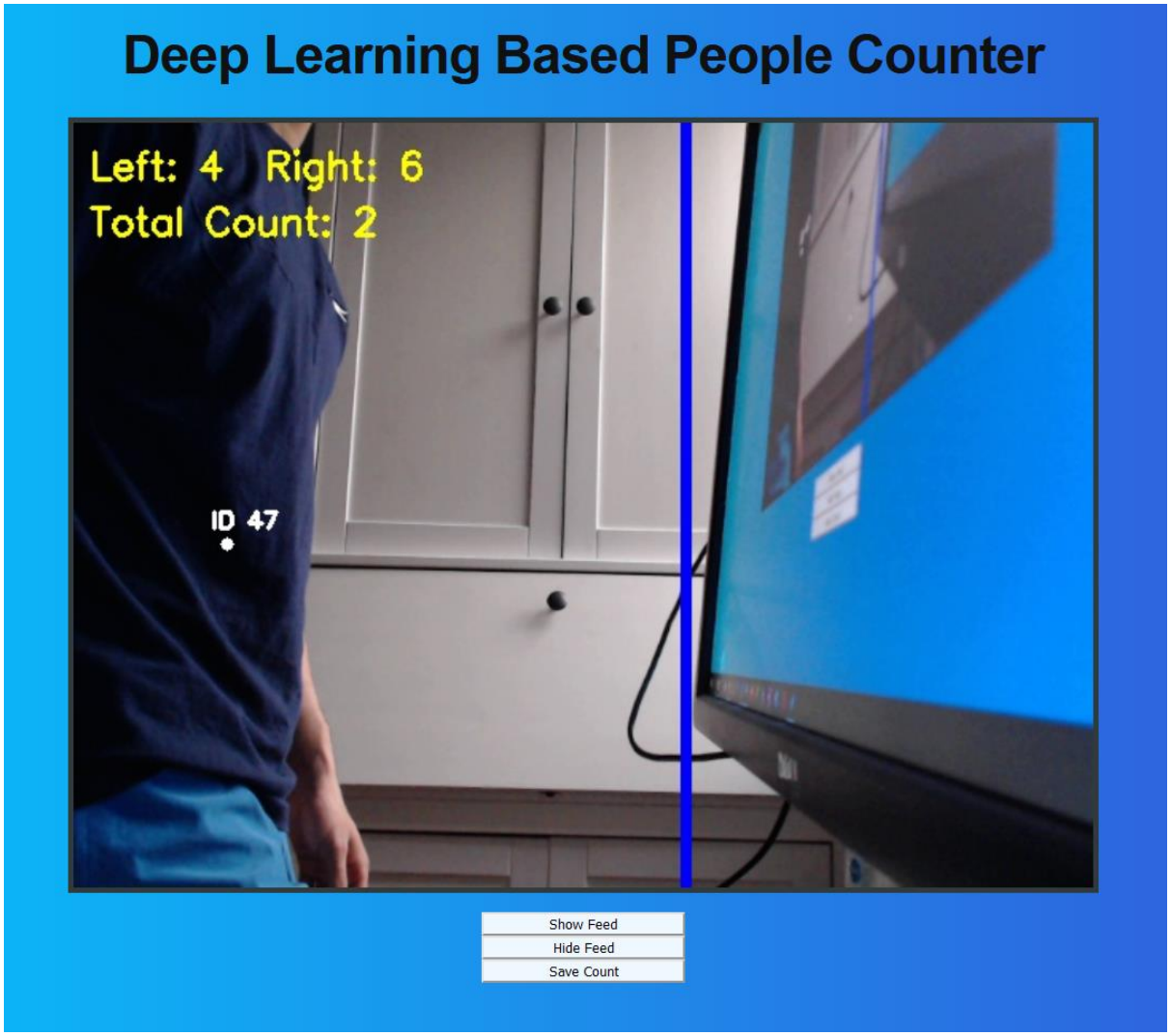
Now I am shifting my attention to the feed itself. I want to remove the status text as I feel it is not contributing to the display and is unnecessary. Therefore, I am going to remove it as well as replacing it with a total count so that the number of people in a room at a given time Is given which is clear to the user. This will contribute to the usability as it will be easier to understand what is going on.

This following code replaces the code to display the status on the image:

```
if net_left_count >= 0:
    cv2.putText(image_np, 'Total Count: ' + str(net_left_count), (10, 70), font, 0.8, (0, 0xFF, 0xFF), 2, cv2.FONT_HERSHEY_SIMPLEX)
else:
    cv2.putText(image_np, 'Total Count: ' + str(net_right_count), (10, 70), font, 0.8, (0, 0xFF, 0xFF), 2, cv2.FONT_HERSHEY_SIMPLEX)
```

I tested this, walking to the right 6 times, each time moving the camera and going back to the left then repeating the process walking left 4 times. This simulates 6 people walking into a room and 4 people walking out.

This was the output:



This shows that the program can now successfully display the total count of people in a room.

User Feedback

Now that I had refined the user interface, I gave it to the users and asked the following questions:

1) How does the webpage feel in terms of how easy it is to use and understand what is going on?
2) Is the webpage appealing?
3) Are there any particular difficulties that you encountered when using it?

Responses:

Teacher:

1) The interface is simple and easy to use, I have a good grasp of what is going on in the page and this contributes to it being easy to use as well.
2) It looks much better than it did in the previous versions and the simple nature is retained so I think the webpage is definitely appealing and I don't think there are any issues with how it looks.
3) No, it was straightforward to use the page and I didn't run into any difficulties using it.

Shop Owner:

1) The page is extremely easy to use and there are no complex controls, and it does what I need it to do.
2) I believe the webpage has found a good balance of displaying information as well as keeping the page minimalistic, so it is definitely appealing, and I like the fact that it does this while still being very useful.
3) There weren't any issues I encountered.

Society organizer:

1) The app did what I wanted and using it didn't require any effort, all I had to do was click the button to show the feed and it worked so I am happy with it. Showing the left and right count was useful and it gave me a sense of how the total count was calculated as well as this, the line also gave me an insight as to how the count left and right was made, increasing the counts depending on if a person walked left or right.
2) I liked the colour scheme, the buttons stood out on the page and were simple to use
3) No difficulties on my end, it worked as expected.

For the feedback for this section, I was pleased, and it indicated that the refinement stage for the UI was successful and aided in the usability of the program and was more appealing to the stakeholders. Also, it didn't take away from any functionality and importantly, the app was kept lightweight.

Review

In this stage I have:

- Added an external CSS stylesheet to improve the appearance of the webpage
- Removed the status of the detections
- Edited the display in the feed to show total count instead of the status

This stage allowed me to make the app visually appealing on the front end so that all the work that was done on the back end could be showcased in a more appealing way which allowed the users to easily understand the function of the app and use it without any difficulties.

Addressing the Success Criteria:

- Criteria 1 – This is addressed completely, with testing of this shown through the screenshots
- Criteria 2 - This criterion is not completely met as the buttons included are start/stop detection and save feed
- Criteria 3 - This Is met as shown through the user responses on the final UI as well as screenshots that show that the page is kept simple and lightweight and the fact that it runs smoothly due to implementations of efficiencies, such as the skip frames in the generator function, also contributes to the app being lightweight.
- Criteria 4 – This was met in a previous stage and show through screenshots, so it is met also at this current stage.
- Criteria 5 – This stage is also met, however instead of bounding boxes, dots with ID numbers are used which keep the app lightweight.
- Criteria 6 – There are buttons on the page which allow features to be toggled however there isn't a dedicated information menu due to the limited number of buttons, this wasn't added. So, this criterion was partially met.
- Criteria 7 – The analysis is in the form of the save count, so this is partially met, further analysis is not completed at this stage.

Overall, I found this stage to be highly important as it tidied up the front-end of the app and allowed the functionality of the backend to be shown clearly and effectively, and stakeholders responded well to this.

Evaluation

Success Criteria Met/Not Met

Criteria	Met/Partially Met/Not Met
1 - Is the correct number of people in a room at any given time given as the minimum output?	Met
2 – Does the website or app give the user the option to toggle different features?	Partially Met
3 - Is the website/app simple and lightweight so the user can easily use it?	Met
4 - Does the program correctly identify if people are coming into and out of a room?	Met
5 – If the user toggles the live camera feed output with bounding boxes around detected people, is this displayed clearly and correctly?	Met
6 - Is there an information menu to direct the user to be able to toggle the different features and allow them to easily use the software with a user-friendly graphical interface?	Partially Met
7 – Is the analysis effective and does it cover all the user's needs?	Not met

Testing for Function & Robustness

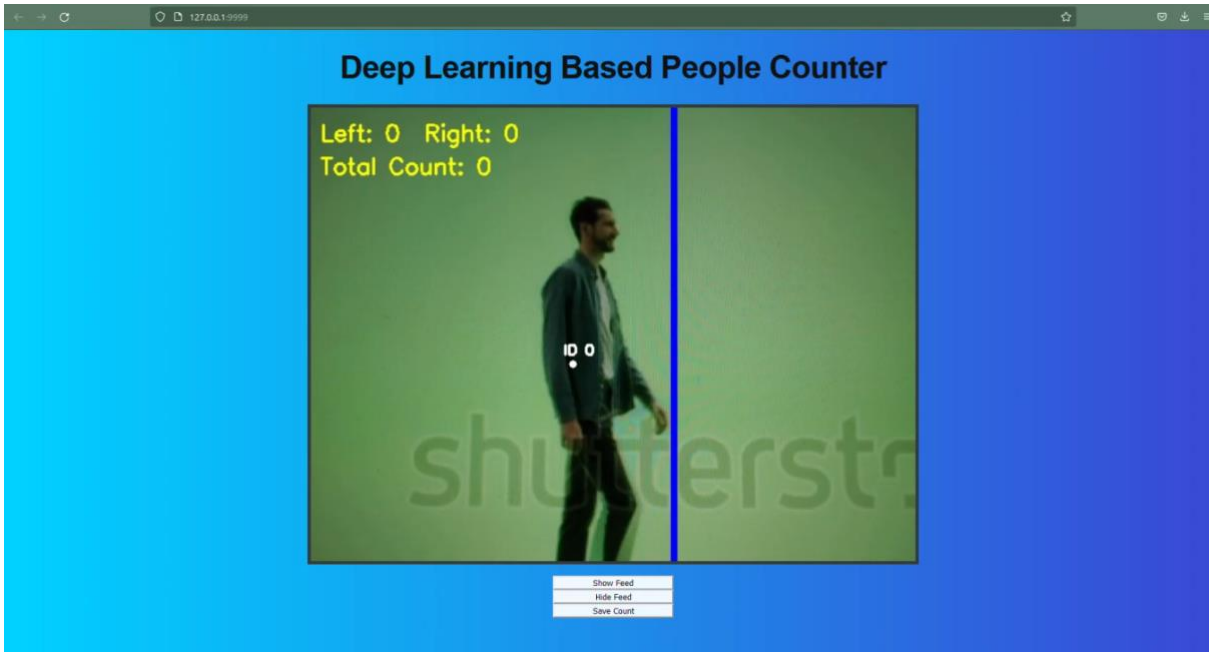
Criteria 1 (Met):

This criterion is one of the core functionalities of the solution and I have made sure to create a model that can detect people with an acceptable accuracy as well as displaying this through a webpage using the flask framework, showing the webcam feed with the total count overlayed which gives the correct number of people in a room at any given time in a clear and concise way to the user.

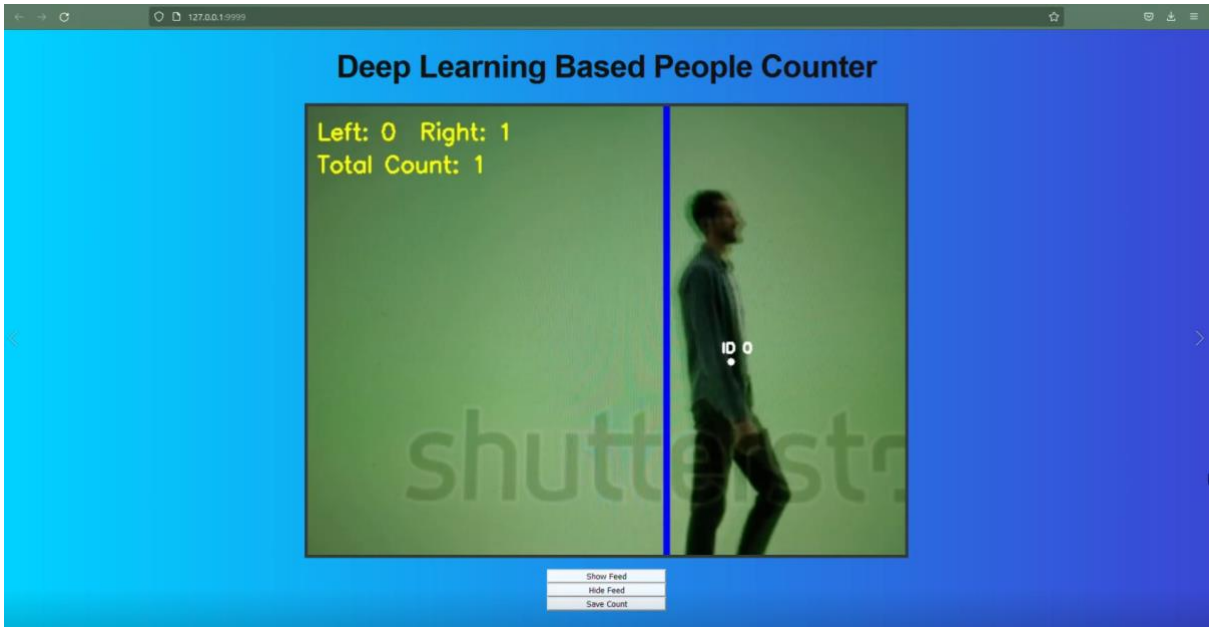
To test this, I have used stock footage from the Shutterstock website of first one person walking past the screen to show that the detections functions correctly and everything is working as expected.

This is a link to the URL of the video used: https://www.shutterstock.com/video/clip-1052006683-green-screen-chroma-key-studio-strong-handsome (accessed 14/03/2022)

This is the screenshot of the person walking before they cross the line:



This is a screenshot after the person has passed the line:



This test shows that the app can successfully detect a person, track them, and give the count when they walk past the line.

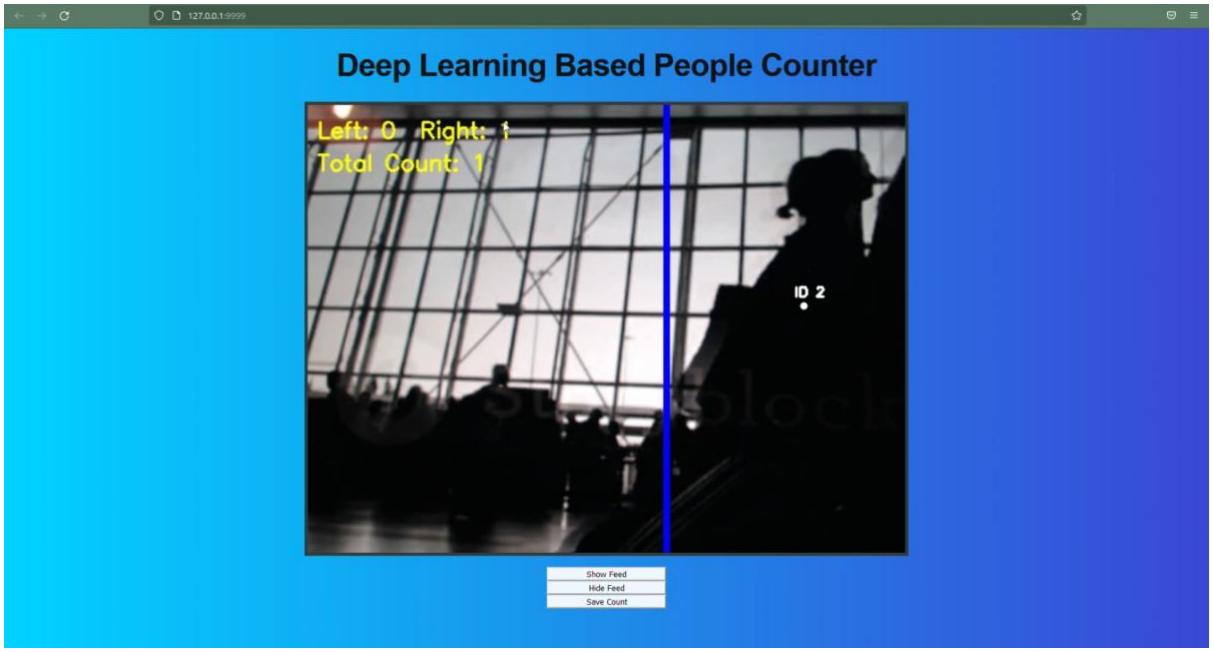
Next, I am going to use footage which will show multiple people walking past the line to test that the app can handle more than one person walking past I will be including screenshots that show the page as the video progresses.

This is the link to the video I used for this test: https://www.shutterstock.com/video/clip-1052006683-green-screen-chroma-key-studio-strong-handsome (accessed 15/03/2022)

This video in particular involved different lighting conditions to the images that the model was trained on so this will test how robust the app is.

This is the screenshot before the first walks past:

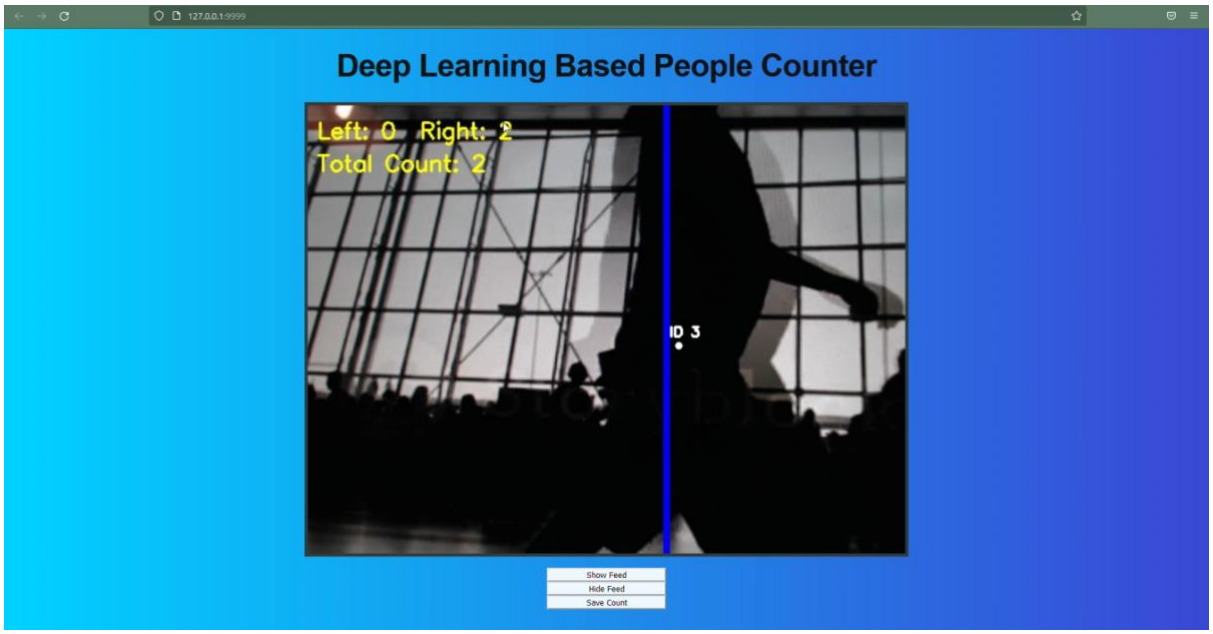
This is the screenshot after the first person walks past:



The count has incremented by one. One thing to note is that the camera detects only the people in the foreground, this is desirable as the app is designed to be used in situations such as at a door where people are walking directly in front of the camera.

This is the screenshot before the second person walks past:

This is the screenshot after the second person walks past:



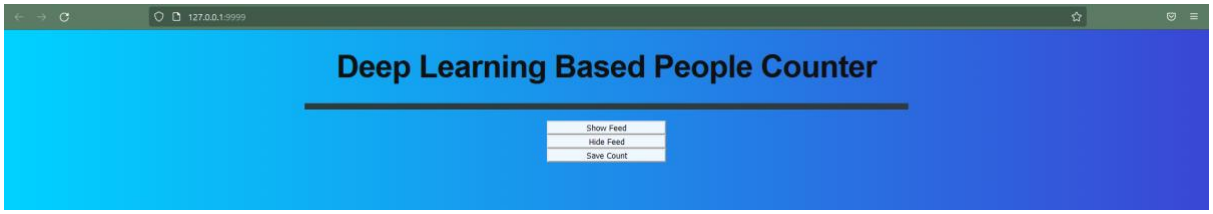
This shows that the app can handle multiple people walking past and is robust enough to handle difficult lighting conditions.

Criteria 2 (Partially Met):

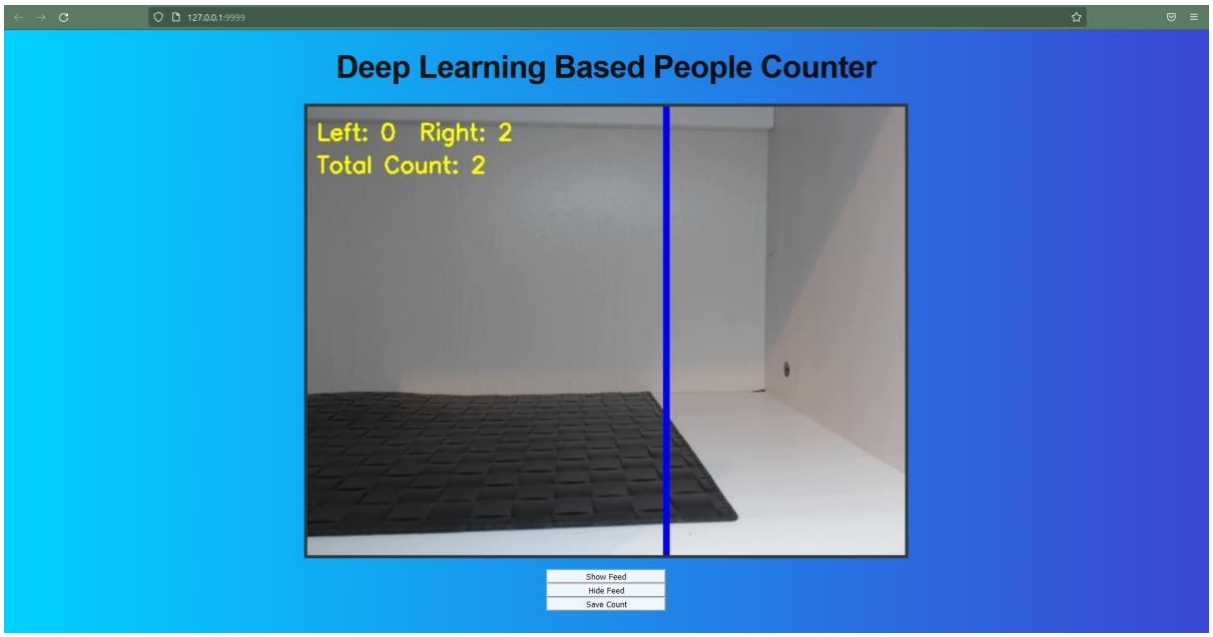
This criterion was partially met due to fewer buttons being used so I will be testing the buttons that I have created.

The first Button I will test is the show feed button.

This is a screenshot of the webpage before it is pressed:

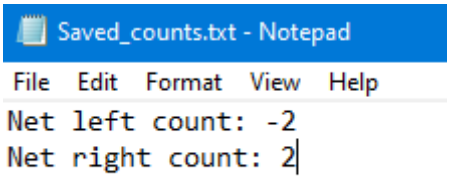


This is a screenshot of the webpage after it is pressed:



Next, I will be testing the hide feed button, the screenshot before it is pressed is shown above, and the following is the screenshot of after it is pressed:

Next, the testing of the save count button. For the purpose of testing this, I will use the previous screenshot where the total count was 2 so this should be saved in some form after I click the save count button. After I click it, the following file is created:



The total right count shows the total count of people. During development, I saved both the net left and right counts as doing so shows the direction in which the people have walked so in this case, since the left count is negative, and the right count is positive I know that overall, there are 2 people on the right.

In terms of testing these buttons for robustness, there isn't a lot of scope for this since the only input is clicking the button or not clicking it. The only way that robustness could be tested for is for different counts that are being saved however, the count has already been tested to be robust for criterion 1 so the save button will automatically save the correct count as a consequence.

Criterion 4 (Met):

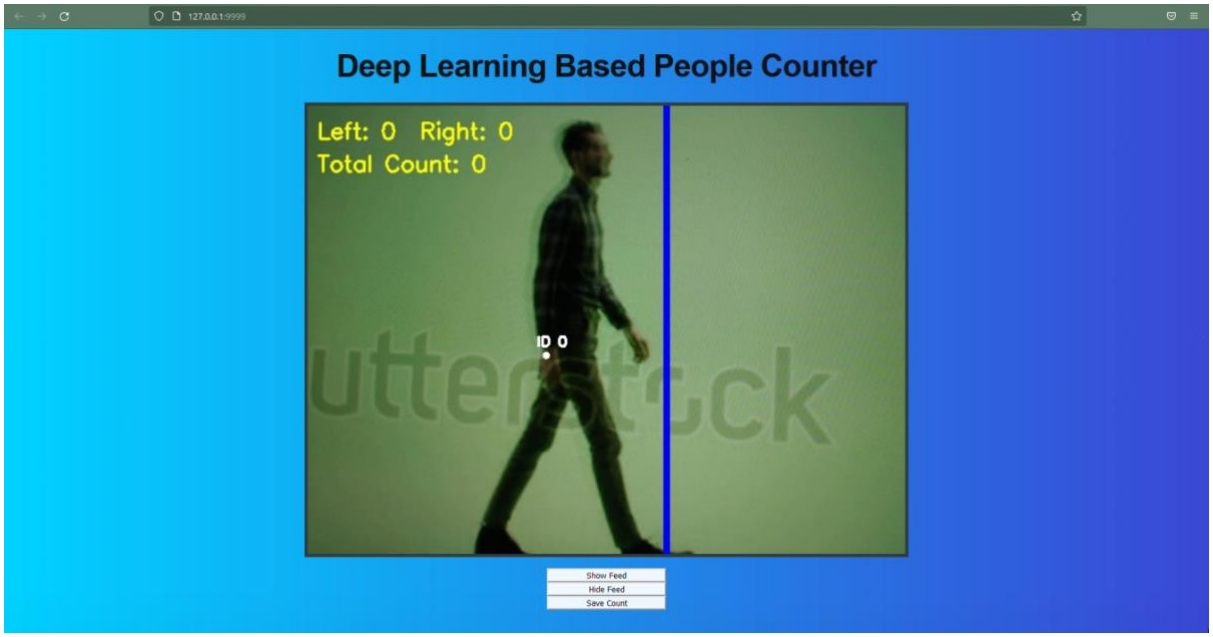
Evidence for this criterion is shown in evidence for criterion 1 which was also met however, something more specific to this criterion which I want to test is people moving in both directions and checking if the correct total count is given then.

To do this I used 2 different videos, one with a person moving left and one with a person moving right so that I could test that the cumulative count was working:

link to first video: https://www.shutterstock.com/video/clip-1052730983-green-screen-chroma-key-studio-strong-handsome (accessed 16/03/2022)

Link to second video: https://www.shutterstock.com/video/clip-1052730992-green-screen-chroma-key-studio-beautiful-young (accessed 16/03/2022)

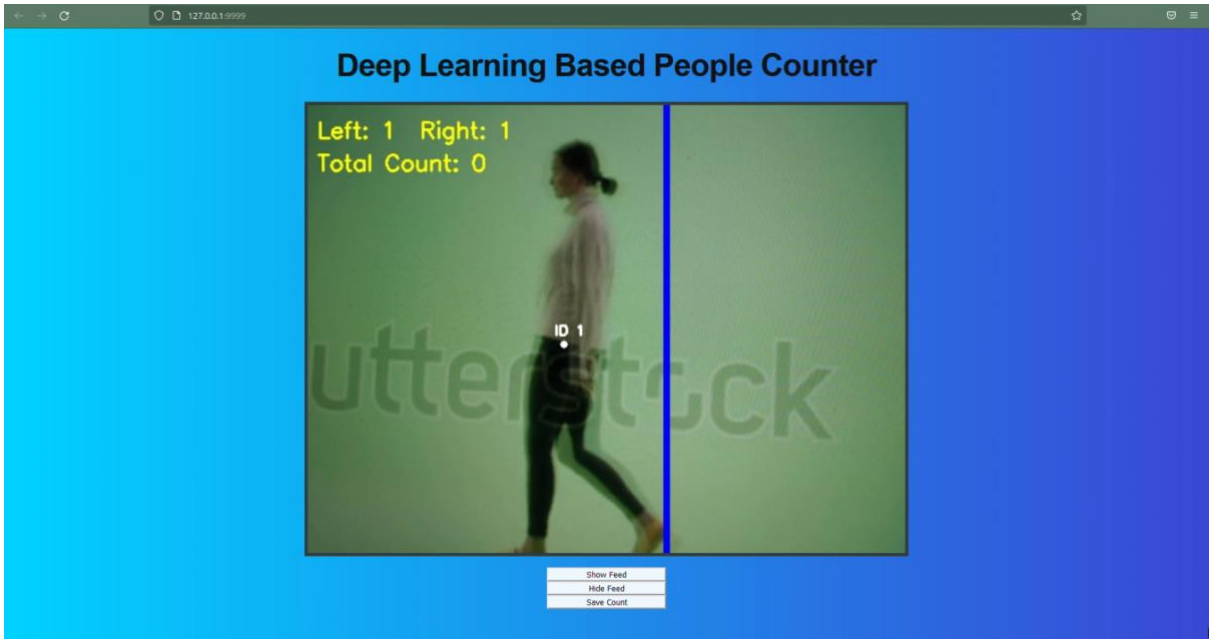
This is a screenshot before the first person walks to the right:



This is a screenshot after the first person walks to the right:

This is a screenshot before the second person moves to the left:

This is a screenshot after the second person moves to the left:



This output is evidence for the app working for people moving left and right so it does correctly identify people coming into and out of somewhere, therefore showing that the success criterion is met.

Usability Testing

The following criteria related to usability, and I used user feedback to evaluate these.

Criterion 3 (Met):

This criterion was something that I kept in mind throughout the whole of the development stage as it was one of the core aspects of the solutions that I wanted to ensure as this made it accessible for the stakeholders in terms of not needed highly expensive hardware such as examples of solutions described in analysis as well as being simple and easy to use.

The stakeholder feedback in the last stage of development directly addresses this, especially the first 2 questions:

1) How does the webpage feel in terms of how easy it is to use and understand what is going on?
2) Is the webpage appealing?

Responses:

Teacher:

1) The interface is simple and easy to use, I have a good grasp of what is going on in the page and this contributes to it being easy to use as well.

2) It looks much better than it did in the previous versions and the simple nature is retained so I think the webpage is definitely appealing and I don't think there are any issues with how it looks.

Shop Owner:

1) The page is extremely easy to use and there are no complex controls, and it does what I need it to do.
2) I believe the webpage has found a good balance of displaying information as well as keeping the page minimalistic, so it is definitely appealing, and I like the fact that it does this while still being very useful.

Society organizer:

1) The app did what I wanted and using it didn't require any effort, all I had to do was click the button to show the feed and it worked so I am happy with it. Showing the left and right count was useful and it gave me a sense of how the total count was calculated as well as this, the line also gave me an insight as to how the count left and right was made, increasing the counts depending on if a person walked left or right.
2) I liked the colour scheme, the buttons stood out on the page and were simple to use

The feedback for this stage indicated that the app was definitely simple to use, and the users were pleased with the interface as well.

To address the lightweight component further, I asked the following questions after giving the stakeholders the software to test via a Jetson Nano:

1) Is the FPS (frames per second) of the feed acceptable?
2) Are there any issues with performance when running the app?

Responses:

Teacher:

1) Yes, the video shown runs fast enough for the feed to be useful and I can see the detections easily.
2) No issues.

Shop Owner:

1) Definitely, I am able to use external hardware for this as the store already has computers which I can make use of for sufficient processing power, so this wasn't an issue for me
2) There were some initial compatibility issues with the libraries which I had to sort out but after doing this, the performance was good.

Society organizer:

1) Using the Jetson Nano, I was able to get a strong FPS for the feed and it was smooth and easy to read off.
2) I didn't encounter any performance issues.

These responses back up the lightweight aspect of the app as the users didn't have problems using the app. However, the shop owner mentioned some initial compatibility issues in terms of the libraries that had to be installed. In further development, I would add a requirements.txt file along with the software to avoid this issue. It could be directly used in a python command and running it would install all the libraries without the user having to do this for themselves manually.

Criterion 5 (Met):

This Criterion stipulated bounding boxes being correctly displayed on the feed if toggled, In the actual implementation, I used dots that represented the centroids of the bounding boxes as this made the app more lightweight. In all the previous screenshots, a dot on each detection is shown clearly along with an ID for the detection number (the nth person detected). This aided in usability and made detections clear. The user feedback described for criterion 3 above addresses this as they mention that it is easy to see the detections and the interface is appealing so clearly the stakeholders are happy with this aspect of the solution and therefore this criterion can be labelled as met.

Criterion 6 (Partially Met):

This criterion involved the information menu and allowing for different features to be toggled in a user-friendly graphical interface. The user-friendly graphical interface has been evidenced in the stakeholder feedback. The information menu is also already evidenced as part of the buttons however, there aren't enough buttons that have been created which make a suitable information menu which is the reason why this criterion is labelled as partially met, however in terms of usability, not meeting this criterion has not taken from the ease of use. If I had included more buttons, then this may have taken away from usability as a separate area for the buttons would be more suitable for the user however, since I have only used three, the usability isn't hindered, which is also backed up by the stakeholder feedback, so partially met fits this criterion the best.

Criterion 7 (Not Met):

To gather further evidence relating to this criterion, I asked the following question to the stakeholders:

- Do you feel that the save button constitutes analysis and does manually using it help in analysing data?

Responses:

Teacher:

- I think that using the button on the bell would help me track how many people are late over certain days however this is something I would have to look at myself and the data isn't given to me in an easy-to-understand format, something like a graph may be more helpful.

Shop Owner:

- Due to the volume of people using the shop and responsibilities of employees, manually using this button to analyse trends is not a feasible thing so I don't think that the button really constitutes as sufficient analysis for me and therefore I doesn't really help in analysing the data for me.

Society Organizer:

- After each meeting, I can record the count and look at it over many meetings which is helpful in showing me how many people come to the society each week however a more useful way to display the data such as a time series graph would very helpful so I think that it does contribute to analysis but not as effectively as use of graphs would be.

These responses showed that the button was limited in analysis and, especially in the case of the shop owner, was not sufficient for analysis which is why indicates that the criteria should be labelled as not met.

Addressing Partially Met/ Unmet Criteria (Including usability features)

Success Criterion 2:

This criterion involved allowing the user to toggle different features. The things I have implemented that address this criterion involve adding the show feed and hide feed buttons as they toggle the feed on and off. However, this is limited in terms of the number of features that the user is able to toggle which is why I have labelled this criterion as partially met.

I wasn't able to add the rest of the buttons due to time constraints and prioritization of the core functions of the solution.

In further development, given more time, I would add buttons such as logging the total count every 10 minutes which would address the business stakeholder more as it would allow them to use this for analysis.

To do this, I would use the python time module and after 10 minutes passes, I would automatically save the count to a file in a readable format. To implement the button with this, firstly I would create a save count every 10 minutes button in the html file, then I would add a decorator function to the flask main app.py file which would take the input from the button and save a Boolean variable to indicate whether or not the button has been pressed. This will then be used in an if condition checking whether the Boolean variable is true or false then it would execute the file handling.

Success Criterion 6:

This criterion is similar to criterion 2, however it stipulates a separate information menu that is easy for the user to use with a graphical user interface.

It can be argued that I have met this with the use of the buttons and adding a user-friendly graphical interface however, I haven't implemented a range of buttons that would be enough to constitute a menu hence the reason I have labelled this criterion as partially met.

This was also due to time constraints and the fact that the core parts of the solution as well as the few key buttons were prioritized over adding the extra features that would be part of the information menu. Also, in relation to the buttons I included in my design phase that I planned to implement including the display total count in and out buttons, I felt that these would be redundant as the point of the solution is to display a count so this should be a permanent part of the solution rather than something that can be toggled so I didn't add this.

If I had more time, during further development I would Adjust the formatting of the page through the CSS and HTML to allow for an information menu on one side as shown in the design phase and it would contain the buttons I didn't include.

A further button That I would have been beneficial to add was the save count every 10 minutes button described above for criterion 2, however this could be improved with the user being allowed to give an input for a time period in which the detections should be saved i.e., 5 minutes or 30 minutes. This flexibility would be more helpful so I think it would be something that would be highly beneficial in further development, and it would cater to individual users more.

To implement this, I would create a set time interval button that would then cause an input box in the html. This part would be done using a JavaScript function similar to the show and hide functions for the show feed and hide feed buttons I created. Then the input from the user would be validated both with JavaScript and at the backend in the app.py file so that the load on the server (the computer running the inference) is reduced and so that the JavaScript can't be bypassed by the user to give erroneous outputs. Next the input would be passed to the app.py file using another decorator function and then a variable would store

the value that the user gave after the validation and this interval would be used instead of 10 minutes with the button described in addressing criterion 2.

Also, adding a button to display use of the app and explaining the different features may be helpful. This would be implemented by a button in html and a JavaScript function that would display a string when clicked. A button to hide this text would also have to be added.

Success Criterion 7:

This criterion involved having effective analysis that would address the needs of the stakeholders, including the business, teacher, and society organizer.

In my developed solution I have created a save count button which can be manually used by the user to save the count when they want, i.e., they could press the button every 10 minutes and use this to pick up trends in the data however, my solution didn't do this automatically and therefore I labelled the criterion as unmet.

The reason I wasn't able to do this was also due to time constraints, again the core aspects of the solution were prioritized, and the analysis was an extra feature above the base functionality of the solution.

If I had more time, this would be one of the main things I would add in further development.

To do this, implementing the save count during a user given time period button described above would be very useful as I could use it automatically when the program is running i.e., every 1 minute without the user clicking a button to activate it so that the count is automatically saved and after this, I can create a graph, plotting count against time which would be highly useful for the stakeholders, especially the business as it would show the times where the most people were in their shops. I can do this using matplotlib library, using in-bult methods to plot a graph, then output an image of the graph to the html page in a similar way to the video decorator function I created to display the feed, but instead a single image would be passed to the web page. Also, buttons show and hide buttons would be created for this which would be simply creating 2 new buttons and using the previously created show and hide JavaScript functions.

Limitations and Maintenance

The biggest limitation in the current solution is the lack of effective analysis and as described earlier would be highly helpful for stakeholders especially the shopkeeper and I have described implementing further buttons to address this, which would be added in further development.

In addition to this, the model may not be fully effective in different environments, despite achieving acceptable performance, it is likely to run into issues since it isn't guaranteed to be 100% accurate. This is something that can be continually improved with performance tuning and if I had more development time, I would definitely revisit this and conduct

further testing in different environments and use test images to improve the model from the different environments so that it can be even more robust.

Also, despite being significantly cheaper than other solutions such as the Irisys solution described in analysis, the solution will have to be implemented in sheltered environments, otherwise the setup could be damaged. This is something that I have not explored therefore it is a limitation of my solution compared to solutions already in the market which would have already had their hardware refined and given sufficient protection against harsher environments. This is something that would be done after the software components is further developed so it would be a lower priority in further development however it is still a limitation and would have to be addressed when making sure that the solution will fit the stakeholder's needs.

Improving the model shouldn't be difficult since the code for model creation is split into image collection and model creation so the image collection notebook can be easily used to gather more images which the model needs to improve on detecting people in. This section is well commented so this will be simple to do. Moreover, the training notebook can be stepped through easily, with comments indicating the functions of the cells so that only the necessary cells need to be executed.

Adding analysis won't be difficult either as it will be building on the app.py file where decorator functions would need to be added similarly to the existing ones so these can be referred and the general comments in this section will help in making sure that the flask framework is correctly used. The HTML CSS and JavaScript will be largely similar for adding features like buttons and there will be repetition so previous code can be re-used such as the CSS styling as well as the show/hide functions in the JavaScript.

Overall, maintaining the solution would involve responded to lower performance of the model in certain environments by adding more test images that would suit the environment, Improving the robustness of the hardware as well as addressing stakeholders needs, namely adding effecting analysis.

Final Code

App.py

```
# Importing the necessary libraries including flask and openCV
from flask import Flask, render_template, Response
import cv2
import cv2
import numpy as np
import argparse
import tensorflow as tf
import dlib

from object_detection.utils import label_map_util
from object_detection.utils import ops as utils_ops

from trackable_object import TrackableObject
from centroidtracker import CentroidTracker

# patch tf1 into `utils.ops`
utils_ops.tf = tf.compat.v1
# Patch the location of gfile
tf.gfile = tf.io.gfile

# Creating the flask app
app = Flask(__name__)

# Creating variable a camera which will store input from the webcam
camera = cv2.VideoCapture(0)

# Using a decorator and a function to render the html page located in the template folder
@app.route('/')
def index():
    return render_template('index.html')

def load_model(model_path):
    tf.keras.backend.clear_session()
    model = tf.saved_model.load(model_path)
    return model
```

```
def run_inference_for_single_image(model, image):
    image = np.asarray(image)
    # The input needs to be a tensor, convert it using `tf.convert_to_tensor`.
    input_tensor = tf.convert_to_tensor(image)
    # The model expects a batch of images, so add an axis with `tf.newaxis`.
    input_tensor = input_tensor[tf.newaxis, ...]

    # Run inference
    output_dict = model(input_tensor)

    # All outputs are batches tensors.
    # Convert to numpy arrays, and take index [0] to remove the batch dimension.
    # We're only interested in the first num_detections.
    num_detections = int(output_dict.pop('num_detections'))
    output_dict = {key: value[0, :num_detections].numpy()
                   for key, value in output_dict.items()}
    output_dict['num_detections'] = num_detections

    # detection_classes should be ints.
    output_dict['detection_classes'] = output_dict['detection_classes'].astype(
        np.int64)

    # Handle models with masks:
    if 'detection_masks' in output_dict:
        # Reframe the the bbox mask to the image size.
        detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
            output_dict['detection_masks'], output_dict['detection_boxes'],
            image.shape[0], image.shape[1])
        detection_masks_reframed = tf.cast(
            detection_masks_reframed > 0.5, tf.uint8)
        output_dict['detection_masks_reframed'] = detection_masks_reframed.numpy()

    return output_dict

# A function which takes the global camera variable and uses it to return a modified frame with a bounding box
around faces face
def generate_frames(model, category_index, cap, labels, roi_position=0.6, threshold=0.5, x_axis=True,
skip_frames=20,total_frames=0):
    while True:
```

```
    # read a frame from the camera
    success,image_np=camera.read()

    # Check is made whether the camera input was successfully read or not
    if not success:
        break
    else:

        ################################################################################

        height, width, _ = image_np.shape
        rgb = cv2.cvtColor(image_np, cv2.COLOR_BGR2RGB)

        status = "Waiting"
        rects = []

        if total_frames % skip_frames == 0:
            status = "Detecting"
            trackers = []

            # Actual detection.
            output_dict = run_inference_for_single_image(model, image_np)
            for i, (y_min, x_min, y_max, x_max) in enumerate(output_dict['detection_boxes']):
                if output_dict['detection_scores'][i] > threshold: #and (labels == None or
category_index[output_dict['detection_classes'][i]]['name'] in labels):
                    # print("DETECTION MADE")
                    tracker = dlib.correlation_tracker()
                    rect = dlib.rectangle(
                        int(x_min * width), int(y_min * height), int(x_max * width), int(y_max * height))
                    tracker.start_track(rgb, rect)
                    trackers.append(tracker)
        else:
            status = "Tracking"
            for tracker in trackers:
                # update the tracker and grab the updated position
                tracker.update(rgb)
                pos = tracker.get_position()

                # unpack the position object
```

```
            x_min, y_min, x_max, y_max = int(pos.left()), int(
                pos.top()), int(pos.right()), int(pos.bottom())

            # add the bounding box coordinates to the rectangles list
            rects.append((x_min, y_min, x_max, y_max))

    objects = ct.update(rects)

    for (objectID, centroid) in objects.items():
        to = trackableObjects.get(objectID, None)

        if to is None:
            to = TrackableObject(objectID, centroid)
        else:
            if x_axis and not to.counted:
                x = [c[0] for c in to.centroids]
                direction = centroid[0] - np.mean(x)

                if centroid[0] > roi_position*width and direction > 0 and np.mean(x) < roi_position*width:
                    counter[1] += 1
                    to.counted = True
                elif centroid[0] < roi_position*width and direction < 0 and np.mean(x) > roi_position*width:
                    counter[0] += 1
                    to.counted = True

            elif not x_axis and not to.counted:
                y = [c[1] for c in to.centroids]
                direction = centroid[1] - np.mean(y)

                if centroid[1] > roi_position*height and direction > 0 and np.mean(y) < roi_position*height:
                    counter[3] += 1
                    to.counted = True
                elif centroid[1] < roi_position*height and direction < 0 and np.mean(y) > roi_position*height:
                    counter[2] += 1
                    to.counted = True

            to.centroids.append(centroid)

            trackableObjects[objectID] = to
```

```
            text = "ID {}".format(objectID)
            cv2.putText(image_np, text, (centroid[0] - 10, centroid[1] - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
            cv2.circle(
                image_np, (centroid[0], centroid[1]), 4, (255, 255, 255), -1)

        # Draw ROI line
        if x_axis:
            cv2.line(image_np, (int(roi_position*width), 0),
                     (int(roi_position*width), height), (0xFF, 0, 0), 5)
        else:
            cv2.line(image_np, (0, int(roi_position*height)),
                     (width, int(roi_position*height)), (0xFF, 0, 0), 5)

        # display count and status
        font = cv2.FONT_HERSHEY_SIMPLEX
        net_left_count = counter[0] - counter[1]
        net_right_count = counter[1] - counter[0]
        if x_axis:
            cv2.putText(image_np, f'Left: {counter[0]}  Right: {counter[1]}', (10, 35), font, 0.8, (0, 0xFF, 0xFF), 2,
cv2.FONT_HERSHEY_SIMPLEX)
        else:
            cv2.putText(image_np, f'Up: {counter[2]}  Down: {counter[3]}', (10, 35), font, 0.8, (0, 0xFF, 0xFF), 2,
cv2.FONT_HERSHEY_SIMPLEX)
        if net_left_count >= 0:
            cv2.putText(image_np, 'Total Count: ' + str(net_left_count), (10, 70), font, 0.8, (0, 0xFF, 0xFF), 2,
cv2.FONT_HERSHEY_SIMPLEX)
        else:
            cv2.putText(image_np, 'Total Count: ' + str(net_right_count), (10, 70), font, 0.8, (0, 0xFF, 0xFF), 2,
cv2.FONT_HERSHEY_SIMPLEX)
        #cv2.putText(image_np, 'Status: ' + status, (10, 70), font, 0.8, (0, 0xFF, 0xFF), 2,
cv2.FONT_HERSHEY_SIMPLEX)

        total_frames += 1
        #############################################################################

        # The resulting frame is encoded and converted to bytes
        ret,buffer=cv2.imencode('.jpg',image_np)
        frame=buffer.tobytes()
```

```
        # Yield is used so that the generator function can be continually called
        yield(b'--frame\r\n'b'Content-Type: frame/jpeg\r\n\r\n' + frame + b'\r\n')

# Here a decorator is used to start the generator function to allow for the video to be taken and processed and
then returned to the webpage
@app.route('/video')
def video():
    return Response(generate_frames(detection_model, category_index, camera,
labels="label_map.pbtxt"),mimetype='multipart/x-mixed-replace; boundary=frame')

# Creating a function to save the net counts
@app.route('/save_result')
def save_result(net_left_count, net_right_count):
    # Opens file and writes required text
    with open("Saved_counts.txt", 'w') as f:
        f.write("Net left count: ")
        f.write(net_left_count)
        f.write("\n")
        f.write("Net right count: ")
        f.write(net_right_count)

# The flask app itself is run in the following section with the host ip address and port specified
if __name__ == '__main__':
    detection_model = load_model("my_ssd_mobnet_tuned_2\export\saved_model")
    category_index = label_map_util.create_category_index_from_labelmap(
        "label_map.pbtxt", use_display_name=True)
    counter = [0, 0, 0, 0]  # left, right, up, down
    total_frames = 0

    ct = CentroidTracker(maxDisappeared=40, maxDistance=50)
    trackers = []
    trackableObjects = {}
    app.run(host='127.0.0.1', debug=False, port=9999)
```

Centroidtracker.py

```
# import the necessary packages
from scipy.spatial import distance as dist
from collections import OrderedDict
import numpy as np


class CentroidTracker:
    def __init__(self, maxDisappeared=50, maxDistance=50):
        # initialize the next unique object ID along with two ordered
        # dictionaries used to keep track of mapping a given object
        # ID to its centroid and number of consecutive frames it has
        # been marked as "disappeared", respectively
        self.nextObjectID = 0
        self.objects = OrderedDict()
        self.disappeared = OrderedDict()

        # store the number of maximum consecutive frames a given
        # object is allowed to be marked as "disappeared" until we
        # need to deregister the object from tracking
        self.maxDisappeared = maxDisappeared

        # store the maximum distance between centroids to associate
        # an object -- if the distance is larger than this maximum
        # distance we'll start to mark the object as "disappeared"
        self.maxDistance = maxDistance

    def register(self, centroid):
        # when registering an object we use the next available object
        # ID to store the centroid
        self.objects[self.nextObjectID] = centroid
        self.disappeared[self.nextObjectID] = 0
        self.nextObjectID += 1

    def deregister(self, objectID):
        # to deregister an object ID we delete the object ID from
        # both of our respective dictionaries
        del self.objects[objectID]
        del self.disappeared[objectID]

    def update(self, rects):
        # check to see if the list of input bounding box rectangles
        # is empty
        if len(rects) == 0:
            # loop over any existing tracked objects and mark them
            # as disappeared
            for objectID in list(self.disappeared.keys()):
```

```
                self.disappeared[objectID] += 1

                # if we have reached a maximum number of consecutive
                # frames where a given object has been marked as
                # missing, deregister it
                if self.disappeared[objectID] > self.maxDisappeared:
                    self.deregister(objectID)

            # return early as there are no centroids or tracking info
            # to update
            return self.objects

        # initialize an array of input centroids for the current frame
        inputCentroids = np.zeros((len(rects), 2), dtype="int")

        # loop over the bounding box rectangles
        for (i, (startX, startY, endX, endY)) in enumerate(rects):
            # use the bounding box coordinates to derive the centroid
            cX = int((startX + endX) / 2.0)
            cY = int((startY + endY) / 2.0)
            inputCentroids[i] = (cX, cY)

        # if we are currently not tracking any objects take the input
        # centroids and register each of them
        if len(self.objects) == 0:
            for i in range(0, len(inputCentroids)):
                self.register(inputCentroids[i])

        # otherwise, are are currently tracking objects so we need to
        # try to match the input centroids to existing object
        # centroids
        else:
            # grab the set of object IDs and corresponding centroids
            objectIDs = list(self.objects.keys())
            objectCentroids = list(self.objects.values())

            # compute the distance between each pair of object
            # centroids and input centroids, respectively -- our
            # goal will be to match an input centroid to an existing
            # object centroid
            D = dist.cdist(np.array(objectCentroids), inputCentroids)

            # in order to perform this matching we must (1) find the
            # smallest value in each row and then (2) sort the row
            # indexes based on their minimum values so that the row
            # with the smallest value as at the *front* of the index
            # list
            rows = D.min(axis=1).argsort()
```

```
        # next, we perform a similar process on the columns by
        # finding the smallest value in each column and then
        # sorting using the previously computed row index list
        cols = D.argmin(axis=1)[rows]

        # in order to determine if we need to update, register,
        # or deregister an object we need to keep track of which
        # of the rows and column indexes we have already examined
        usedRows = set()
        usedCols = set()

        # loop over the combination of the (row, column) index
        # tuples
        for (row, col) in zip(rows, cols):
            # if we have already examined either the row or
            # column value before, ignore it
            if row in usedRows or col in usedCols:
                continue

            # if the distance between centroids is greater than
            # the maximum distance, do not associate the two
            # centroids to the same object
            if D[row, col] > self.maxDistance:
                continue

            # otherwise, grab the object ID for the current row,
            # set its new centroid, and reset the disappeared
            # counter
            objectID = objectIDs[row]
            self.objects[objectID] = inputCentroids[col]
            self.disappeared[objectID] = 0

            # indicate that we have examined each of the row and
            # column indexes, respectively
            usedRows.add(row)
            usedCols.add(col)

        # compute both the row and column index we have NOT yet
        # examined
        unusedRows = set(range(0, D.shape[0])).difference(usedRows)
        unusedCols = set(range(0, D.shape[1])).difference(usedCols)

        # in the event that the number of object centroids is
        # equal or greater than the number of input centroids
        # we need to check and see if some of these objects have
        # potentially disappeared
        if D.shape[0] >= D.shape[1]:
```

```
            # loop over the unused row indexes
            for row in unusedRows:
                # grab the object ID for the corresponding row
                # index and increment the disappeared counter
                objectID = objectIDs[row]
                self.disappeared[objectID] += 1

                # check to see if the number of consecutive
                # frames the object has been marked "disappeared"
                # for warrants deregistering the object
                if self.disappeared[objectID] > self.maxDisappeared:
                    self.deregister(objectID)

        # otherwise, if the number of input centroids is greater
        # than the number of existing object centroids we need to
        # register each new input centroid as a trackable object
        else:
            for col in unusedCols:
                self.register(inputCentroids[col])

    # return the set of trackable objects
    return self.objects
```

TrackableObject.py

```python
class TrackableObject:
    def __init__(self, objectID, centroid):
        # store the object ID, then initialize a list of centroids
        # using the current centroid
        self.objectID = objectID
        self.centroids = [centroid]

        # initialize a boolean used to indicate if the object has
        # already been counted or not
        self.counted = False
```

Index.html

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="{{ url_for('static', filename='styling.css')}}">
  </head>
 <body>
  <h1>Deep Learning Based People Counter</h1>
  <div>
    <img id="image" src="" width="50%"/><br>
  </div>
  <button type="button"
    onclick="show()" id="btnID">
    Show Feed
  </button>
  <button type="button"
    onclick="hide()" id="btn2ID">
    Hide Feed
  </button>
  <button action="/save_result" id="btn3ID">
    Save Count
  </button>
  <script>
    function show() {

      /* Get image and change value
      of src attribute */
      let image = document.getElementById("image");

      image.src = "{{ url_for('video') }}"

      /* Code for hiding show image button:
      document.getElementById("btnID")
          .style.display = "none";*/
    }
    function hide() {

      /* Get image and change value
      of src attribute */
```

```
        let image = document.getElementById("image");

        image.src = ""

    }
  </script>

  </body>
</html>
```

Styling.css

```
#image {
    display: block;
    margin-left: auto;
    margin-right: auto;
    width: 50%;
    border: 5px solid;
    border-color:rgb(48, 59, 59);
}

button, input[type="button"], input[type="submit"]{
    background-color: aliceblue;
    display: block;
    margin-left: auto;
    margin-right: auto;
    width: 10%;
}

h1 { color: #111;
    font-family: 'Helvetica Neue', sans-serif;
    font-size: 50px;
    font-weight: bold;
    letter-spacing: -1px;
    line-height: 1;
    text-align: center;
}

body {
    background-image: url('img/aqua.png');
```

```
}
```

Image Collection Notebook Code

1. Import Dependencies

```
!pip install opencv-python
```

Python

```
# Import opencv
import cv2

# Import uuid
import uuid

# Import Operating System
import os

# Import time
import time
```

Python

2. Define Images to Collect

```
# Label list defined to hold the label which we will use to define the object we want to detect
labels = ['person']
# variable defined to determine how many images will be collected later on
number_imgs = 5
```

Python

3. Setup Folders

```
# Defines variable to store path where images will be stored
IMAGES_PATH = os.path.join('Tensorflow', 'workspace', 'images', 'collectedimages')
```

Python

```
# Command to create the images path
!mkdir {IMAGES_PATH}

# Using for loop to create a folder for each label (Only creates a path for the person label)
for label in labels:
    path = os.path.join(IMAGES_PATH, label)
```

4. Capture Images

```
for label in labels:
    cap = cv2.VideoCapture(0)# captures the video input
    print('Collecting images for {}'.format(label))# Iterates for all labels (only one for people)
    time.sleep(5)#Stops the program for 5 seconds so that there is time between images being taken
    for imgnum in range(number_imgs):# Iterates 5 times to get 5 images
        print('Collecting image {}'.format(imgnum))
        ret, frame = cap.read()#Captures a frame from the webcam input
        # Creates a unique image name, making use of uuid
        imgname = os.path.join(IMAGES_PATH,label,label+'.'+'{}.jpg'.format(str(uuid.uuid1())))
        #Uses OpenCV to write the image to the path just created for it
        cv2.imwrite(imgname, frame)
        cv2.imshow('frame', frame)#Displays the image that was taken
        time.sleep(2) #Stops the program for 2 second so that there is time between images being taken

        if cv2.waitKey(1) & 0xFF == ord('q'):#Checks if q is pressed at which point, the code would be halted
            break
cap.release()
cv2.destroyAllWindows()
```

5. Image Labelling

```
!pip install --upgrade pyqt5 lxml
```

Python

```
# Creating the path where the Labelimg application will be stored
LABELIMG_PATH = os.path.join('Tensorflow', 'labelimg')
```

Python

```
!mkdir {LABELIMG_PATH}
!git clone https://github.com/tzutalin/labelImg {LABELIMG_PATH}
```

Python

```
# Installing labelimg
!cd {LABELIMG_PATH} && pyrcc5 -o libs/resources.py resources.qrc
```

Python

```
# Opening LabelImg
!cd {LABELIMG_PATH} && python labelImg.py
```

Python

6. Move them into a Training and Testing Partition

Model Creation Notebook Code

0. Setup Paths

```python
# Importing OS library
import os
```

```python
# Name of my model
CUSTOM_MODEL_NAME = 'my_ssd_mobnet'
# Name of the pre-trained model I am using
PRETRAINED_MODEL_NAME = 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8'
# URL of the pre-trained model I am using
PRETRAINED_MODEL_URL = 'http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz'
# This is a pre-made script that will generate a specific file I need
TF_RECORD_SCRIPT_NAME = 'generate_tfrecord.py'
# This will be the name of the label map that is needed when detecting with thte model
LABEL_MAP_NAME = 'label_map.pbtxt'
```

```python
paths = {
    'WORKSPACE_PATH': os.path.join('Tensorflow', 'workspace'),
    'SCRIPTS_PATH': os.path.join('Tensorflow','scripts'),
    'APIMODEL_PATH': os.path.join('Tensorflow','models'),
    'ANNOTATION_PATH': os.path.join('Tensorflow', 'workspace','annotations'),
    'IMAGE_PATH': os.path.join('Tensorflow', 'workspace','images'),
    'MODEL_PATH': os.path.join('Tensorflow', 'workspace','models'),
    'PRETRAINED_MODEL_PATH': os.path.join('Tensorflow', 'workspace','pre-trained-models'),
    'CHECKPOINT_PATH': os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME),
    'OUTPUT_PATH': os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'export'),
    'TFJS_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'tfjsexport'),
    'TFLITE_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'tfliteexport'),
    'PROTOC_PATH':os.path.join('Tensorflow','protoc')
 }
```

```python
files = {
    'PIPELINE_CONFIG':os.path.join('Tensorflow', 'workspace','models', CUSTOM_MODEL_NAME, 'pipeline.config'),
    'TF_RECORD_SCRIPT': os.path.join(paths['SCRIPTS_PATH'], TF_RECORD_SCRIPT_NAME),
    'LABELMAP': os.path.join(paths['ANNOTATION_PATH'], LABEL_MAP_NAME)
}
```

```python
# Looping over each path value in the paths dictionary
for path in paths.values():
    # Checking if the path already exists
    if not os.path.exists(path):
        #Creating the path
        !mkdir {path}
```

1. Download TF Models Pretrained Models from Tensorflow Model Zoo and Install TFOD

```python
# https://www.tensorflow.org/install/source_windows
```

```python
# !pip install wget
import wget
```

```python
# Checking if the path for the API to be downloaded in exists
if not os.path.exists(os.path.join(paths["APIMODEL_PATH"], 'research', 'object_detection')):
    # Downloads the API files using git clone
    !git clone https://github.com/tensorflow/models {paths['APIMODEL_PATH']}
```

```python
# Install Tensorflow Object Detection
url="https://github.com/protocolbuffers/protobuf/releases/download/v3.15.6/protoc-3.15.6-win64.zip"
wget.download(url)
!move protoc-3.15.6-win64.zip {paths['PROTOC_PATH']}
!cd {paths['PROTOC_PATH']} && tar -xf protoc-3.15.6-win64.zip
os.environ['PATH'] += os.pathsep + os.path.abspath(os.path.join(paths['PROTOC_PATH'], 'bin'))
!cd Tensorflow/models/research && protoc object_detection/protos/*.proto --python_out=. && copy object_detection\\packages\\tf2\\setup.py setup.py && python setup.py build && python setup.py install
!cd Tensorflow/models/research/slim && pip install -e .
```

```python
VERIFICATION_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'builders', 'model_builder_tf2_test.py')
# Verify Installation
!python {VERIFICATION_SCRIPT}
```

```
!pip install tensorflow --upgrade
```
Python

```
!pip install pyyaml
```
Python

```
!pip uninstall protobuf matplotlib -y
!pip install protobuf matplotlib==3.2
```
Python

```
import object_detection
```
Python

```
!pip list
```
Python

```
wget.download(PRETRAINED_MODEL_URL)
!move {PRETRAINED_MODEL_NAME+'.tar.gz'} {paths['PRETRAINED_MODEL_PATH']}
!cd {paths['PRETRAINED_MODEL_PATH']} && tar -zxvf {PRETRAINED_MODEL_NAME+'.tar.gz'}
```
Python

2. Create Label Map

```
# Creating a list containing a dictionary fo each label (only person in my case)
labels = [{'name':'person', 'id':1}]

# Creating the labelmap file in the already created labelmap folder
with open(files['LABELMAP'], 'w') as f:
    # Looping through each label (only goes through person)
    for label in labels:
        # Writes the required content for the label with the name and id
        # anything written is in the correct format
        f.write('item { \n')
        f.write('\tname:\'{}\'\n'.format(label['name']))
        f.write('\tid:{}\n'.format(label['id']))
        f.write('}\n')
```
Python

3. Create TF records

```
# OPTIONAL IF RUNNING ON COLAB
ARCHIVE_FILES = os.path.join(paths['IMAGE_PATH'], 'archive.tar.gz')
if os.path.exists(ARCHIVE_FILES):
  !tar -zxvf {ARCHIVE_FILES}
```
Python

```
if not os.path.exists(files['TF_RECORD_SCRIPT']):
    !git clone https://github.com/tf/GenerateTFRecord {paths['SCRIPTS_PATH']}
```
Python

```
!python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'train')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'], 'train.record')}
!python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'test')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'], 'test.record')}
```
Python

```
!pip install pandas
```
Python

4. Copy Model Config to Training Folder

```
!copy {os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'pipeline.config')} {os.path.join(paths['CHECKPOINT_PATH'])}
```
Python

5. Update Config For Transfer Learning

```
import tensorflow as tf
from object_detection.utils import config_util
from object_detection.protos import pipeline_pb2
from google.protobuf import text_format
```
Python

```
# Assigning config of pipleline.config to config variable
config = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
```
Python

```
config
```
Python

```
# Reading the config file and assigning text to pipeline_config
pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "r") as f:
    proto_str = f.read()
    text_format.Merge(proto_str, pipeline_config)
```
Python

```
# Assigning the number of classes (1 for person) to the section in the pipeline config
pipeline_config.model.ssd.num_classes = len(labels)
# Assigning the batch size which is a parameter used when training
pipeline_config.train_config.batch_size = 4
# Assigning the pretrained model path
pipeline_config.train_config.fine_tune_checkpoint = os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'checkpoint', 'ckpt-0')
# Changing a checkpoint type to detection
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
# Assigningn the label map path
pipeline_config.train_input_reader.label_map_path= files['LABELMAP']
# Assigning the train TFRecord path
pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'], 'train.record')]
# Assigning the label map to be used when evaluating
pipeline_config.eval_input_reader[0].label_map_path = files['LABELMAP']
# Assigning the test TFRecord path
pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'], 'test.record')]
```
Python

```
# Turning the pipeline_config into text that can be written
config_text = text_format.MessageToString(pipeline_config)
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "wb") as f:
    # Writing the additional paths to the config
    f.write(config_text)
```
Python

6. Train the model

```python
# Creating path to training script
TRAINING_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'model_main_tf2.py')
```

```python
command = "python {} --model_dir={} --pipeline_config_path={} --num_train_steps=2000".format(TRAINING_SCRIPT, paths['CHECKPOINT_PATH'],files['PIPELINE_CONFIG'])
```

```python
print(command)
```

```python
!{command}
```

7. Evaluate the Model

```python
command = "python {} --model_dir={} --pipeline_config_path={} --checkpoint_dir={}".format(TRAINING_SCRIPT, paths['CHECKPOINT_PATH'],files['PIPELINE_CONFIG'], paths['CHECKPOINT_PATH'])
```

```python
print(command)
```

```python
!{command}
```

8. Load Train Model From Checkpoint

```python
# Importing Modules for testing
import os
import tensorflow as tf
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
from object_detection.utils import config_util
```

```python
# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
detection_model = model_builder.build(model_config=configs['model'], is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(paths['CHECKPOINT_PATH'], 'ckpt-4')).expect_partial()

@tf.function
def detect_fn(image):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections
```

9. Detect from an Image

```python
# Importing modules for image detection
import cv2
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
```

```python
# Creating a category_index, needed for detections, created from label map
category_index = label_map_util.create_category_index_from_labelmap(files['LABELMAP'])
```

```python
# This is the path of the image which the model is going to be tested on
IMAGE_PATH = os.path.join(paths['IMAGE_PATH'], 'test', '10.png')
```

```python
# First reading in the image and converting it to a numpy array so it can be worked with
img = cv2.imread(IMAGE_PATH)
image_np = np.array(img)

input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
# detection carried out on image and retrurned as a dictionary
detections = detect_fn(input_tensor)

# Finding the detections as well as the number of detections from the dictionary
num_detections = int(detections.pop('num_detections'))
detections = {key: value[0, :num_detections].numpy()
              for key, value in detections.items()}
detections['num_detections'] = num_detections

# detection_classes should be ints.
detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

label_id_offset = 1
image_np_with_detections = image_np.copy()

# Plotting the detection with pyplot
viz_utils.visualize_boxes_and_labels_on_image_array(
            image_np_with_detections,
            detections['detection_boxes'],
            detections['detection_classes']+label_id_offset,
            detections['detection_scores'],
            category_index,
            use_normalized_coordinates=True,
            max_boxes_to_draw=5,
            min_score_thresh=.8,
            agnostic_mode=False)

# Displaying the image with the detection
plt.imshow(cv2.cvtColor(image_np_with_detections, cv2.COLOR_BGR2RGB))
plt.show()
```

10. Real Time Detections from your Webcam

```python
# Capturing the video from the camera
cap = cv2.VideoCapture(0)
# Getting the dimensions of the video
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# Starting a loop so that the inference will run till it is exited
while cap.isOpened():
    # Reading the frame from the video input
    ret, frame = cap.read()
    # Converting the frame to a numpy array
    image_np = np.array(frame)

    # Converting the image to a tensor so that the detect_fn function can be used on it
    input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
    detections = detect_fn(input_tensor)

    # Getting the detections and number of detectyions from the dictionary detections
    num_detections = int(detections.pop('num_detections'))
    detections = {key: value[0, :num_detections].numpy()
                  for key, value in detections.items()}
    detections['num_detections'] = num_detections

    # detection_classes should be ints.
    detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

    label_id_offset = 1
    image_np_with_detections = image_np.copy()

    # Creating an image to display
    viz_utils.visualize_boxes_and_labels_on_image_array(
                image_np_with_detections,
                detections['detection_boxes'],
                detections['detection_classes']+label_id_offset,
                detections['detection_scores'],
                category_index,
                use_normalized_coordinates=True,
                max_boxes_to_draw=5,
                min_score_thresh=.8,
                agnostic_mode=False)

    # Displaying the image with the detections and resized
    cv2.imshow('object detection',  cv2.resize(image_np_with_detections, (800, 600)))

    # This is the flag to exit the loop, clicking q exits the loop
    if cv2.waitKey(10) & 0xFF == ord('q'):
        cap.release()
        cv2.destroyAllWindows()
        break
```

Python

10. Freezing the Graph

```python
FREEZE_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'exporter_main_v2.py ')
```

Python

```python
command = "python {} --input_type=image_tensor --pipeline_config_path={} --trained_checkpoint_dir={} --output_directory={}".format(FREEZE_SCRIPT ,files['PIPELINE_CONFIG'], paths['CHECKPOINT_PATH'], paths['OUTPUT_PATH'])
```

Python

```python
print(command)
```

Python

```python
!{command}
```

Python