Ministerul Educaţiei
Universitatea ”OVIDIUS” Constanţa
Facultatea de Matematică şi Informatică
Specializarea Informatică în Limba Engleză

Real-time detection and marking of ”targets” in a video stream

Lucrare de licenţă

Coordonator ştiinţific:
Prof.univ.dr. Popovici Dorin-Mircea

Absolvent:
Roșca Mihai-Bogdan

Constanţa
2022

Abstract

Like many other methodological innovations, Object Detection has been applied in Computer Vision, allowing us to identify and locate objects in an image or video.

This paper derives from in-depth research, a hybrid project split into two systems, Multi-Object Tracker and Single-Object Tracker, that presents the usage of these technologies, the pros and cons of each system in a video stream and the neural network algorithm You Only Look Once (YOLOv3).

The two Object Detection systems' main scopes are the bounding boxes drawn around an entity, tracking an object by drawing a rectangle around detected objects, the Frames Per Second, the accuracy of both systems and finally, which method is better.

In this paper, you will find distinctively performing algorithms that can compete with the YOLO algorithm and the difference between the modified versions for this algorithm and the version we are currently using.

Exploring this project's use cases and requirements led to reviewing each algorithm's significance separately, as they cannot be compared directly due to specific types of research performed.

Rezumat

La fel ca multe alte inovații metodologice, Object Detection a fost aplicată în Computer Vision, permițându-ne să identificăm și să localizăm obiecte într-o imagine sau videoclip.

Această lucrare derivă dintr-o cercetare aprofundată, un proiect hibrid împărțit în două sisteme, Multi-Object Tracker și Single-Object Tracker, care prezintă utilizarea acestor tehnologii, avantajele și dezavantajele fiecărui sistem într-un flux video și rețeaua neuronală algoritmului You Only Look Once (YOLOv3).

Domeniile principale ale celor două sisteme de detectare a obiectelor sunt căsuțele de delimitare desenate în jurul unei entități, urmărirea unui obiect prin desenarea unui dreptunghi în jurul obiectelor detectate, cadrele pe secundă, precizia ambelor sisteme și, în sfârșit, care metodă este mai bună.

În această lucrare, veți găsi algoritmi cu performanțe distinctive care pot concura cu algoritmul YOLO și diferența dintre versiunile modificate pentru acest algoritm și versiunea pe care o folosim în prezent.

Explorarea cazurilor de utilizare și a cerințelor acestui proiect a condus la revizuirea semnificației fiecărui algoritm separat, deoarece acestea nu pot fi comparate direct din cauza unor tipuri specifice de cercetare efectuate.

Contents

Abstract **2**

Rezumat **3**

Contents **i**

List of Figures **2**

1 Motivation **3**

1.1 Introduction 3

1.2 Safety 4

1.2.1 Autonomous Driving Efficiency 4

1.2.2 Video Surveillance 5

1.3 Object Detection 5

1.4 Finishing Touch 6

2 State of the art **7**

2.1 Traffic speed radar 7

2.2 Facial Recognition 8

2.2.1 Usage 8

2.2.2 Face ID Detection 9
2.3 Medicine 10
2.3.1 Surgery Organ Detection 10

3 Proposed Solution and Implementation 11
3.1 Used Technologies 11
3.2 OpenCV 12
3.3 YOLO Vision Algorithm 12
3.3.1 Algorithm Comparison 13
3.4 YOLO Operation 14
3.4.1 Residual Block 14
3.4.2 Bounding box regression 14
3.4.3 Intersection over Union 14
3.5 Training YOLO 15
3.6 YOLO Downsides 15
3.7 Weights - Configurations 16
3.8 Algorithm Comparisons 17
3.8.1 Fast R-CNN 17
3.8.2 Histogram of Oriented Gradients 17
3.8.3 Single Shot Detector 18

4 Application presentation 19
4.1 Multi-Tracker 20
4.1.1 Comparisons 21
4.2 Code Multi Track 22
4.3 Single-Tracker 27
4.4 Code Single Track 29
4.5 Multi-Usage Code 30

4.6 UML Diagram 31
4.7 Difference between Systems 32

5 Conclusion **34**

Bibliography **36**

List of Figures

1.1 Image Recognition 5

2.1 Speed Traffic 7

2.2 Face ID Detection 9

2.3 Kidney Organ Detector 10

3.1 Visual Algorithm 12

3.2 Algorithm Comparisons 13

3.3 14

3.4 Canny Edge Detector 15

3.5 16

4.1 Multi Tracker 20

4.2 Virtual People 20

4.3 Recordings 22

4.4 FPS Meter 23

4.5 Bounding Box 25

4.6 Object Collision 29

4.7 UML Diagram 32

Chapter 1

Motivation

1.1 Introduction

In 2016 California, Uber blamed the drivers for all self-driving car traffic transgressions. Regulators in the state have demanded that Uber is to remove self-driving from the roads after an incident at multiple red stops. However, after the allegations against the company, immediately the blame jumped on ”human error”.

The problem was with the self-automated Uber system that ran over six red lights in the city during a test drive that was also not approved by the state of California itself. However, after a video was posted to prove the driver was never manual driving, the company was hit hard in return.

It could not detect the right light and almost hit pedestrians due to this massive error. Bugs like this in the system make technology try to get approved before being put in the real world, but sometimes it gets away with it, and the technology is released in its testing phase when it is the most dangerous.

Another report was found by the company Tesla in recent years in San Francisco. Specialists made a sheet of data stating that within the 392 crashes in 2021, 70 per cent of those belonged to Tesla’s autopilot.

There have always existed significant errors like this after this technology was introduced in the real world in specific job fields, and even though a significant percentage of them were successful, there are still, even in the present day, big blunders like the ones stated above.

Though this was only an example of traffic, these errors are not only noticed in this field but also in video surveillance cameras that use this system in secured places. A central concept of current machine learning is that accuracy guarantees are largely domain-specific: good performance on a specific image data does not necessarily translate to another type of image data.

The project was made to explore such an idea, using Object Detection and the algorithm that we will mainly talk about, You Only Look Once, also known as YOLO, and to implement two-tracker systems that we can switch between at our will.

Furthermore, many other reports are heard from around the globe if we pay enough attention. However, companies continue to undervalue the importance of safety and traffic errors that could lead to dangerous situations.

1.2 Safety

Safety should always be society's main priority when dealing with new technologies.

Object Detection has a vital role in advanced driving assistance systems, also known as ADAS, which can detect pedestrians and driving lanes to improve road safety.

With this mobility and transporting piece of technology, we can detect objects from far or close distances, enabling the driver to both passively and actively use the vehicle at hand.

Object detection then comes into play. Automatic vehicle behaviour and driving signal presence have improved substantially. A computer-controlled transmission can now react faster than a human driver, thus pushing Object Detection to be a key element in driving systems.

Pros:

- ▷ Stress levels are lower than in manual driving
- ▷ Popularly available
- ▷ Great in stop-and-go traffic
- ▷ Safety increase towards pedestrians

Cons:

- ▷ Higher purchase price than manual
- ▷ Higher repair costs

1.2.1 Autonomous Driving Efficiency

Because of this technology, using a self-autonomous car has presented to handle better hitting the potholes it cannot avoid. Before the right front wheel hits a pothole that the system sees coming sooner than the manual driving counterpart, the shocks on that wheel will prep for it. The car will make ride quality and efficiency changes in response to actual road conditions it can see coming.

So far, with our current technology, Object Detection is growing every day to become the new standard in everyday traffic. However, this idea will be hard to implement due to costs and time. It would take quite a long time in Romania for experimentation.

1.2.2 Video Surveillance

Video Surveillance inside shops, malls and other places has increased security over the past years. Though this was only an example of traffic, these errors are not only noticed in this field but also in video surveillance cameras that use this system for security measures, such as crime prevention, identification of vandalism, etcetera. A real-time response in the case of an incident, however, requires manual observation of the video stream, which is in most cases economically not feasible. Now object detection's goal is to lower the crime rate as it can track everything far better and even pinpoint illegally used or stolen objects inside the store.

1.3 Object Detection

Object detection is a technology used in Computer Vision and Image Processing that connects with video analysis and image understanding and ties with computer vision techniques. It aids us in deciphering objects and understanding them in videos or images.

Thus the objective is to find all object instances of one or more given object classes, regardless of size, scale, location, position and perspective, trying to produce even with limited information.

Image recognition only outputs a class label for an identified object, and image segmentation creates a pixel-level understanding of a scene's elements. What separates object detection from these other tasks is its unique ability to locate objects within an image or video that can allow us to count and track those objects.

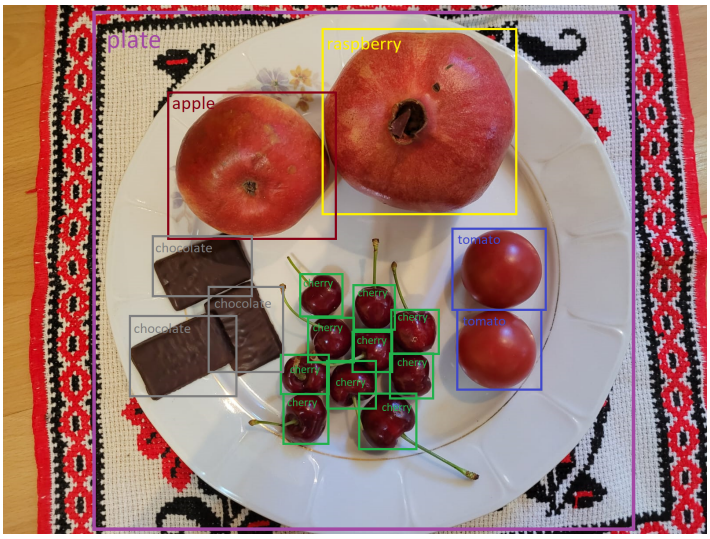

Figure 1.1: Image Recognition

1.4 Finishing Touch

So far, we have touched on two important examples that object detection has improved over the past years; there are many other examples in which our technology has helped in most job fields, but instead talked about the essential ones.

Autonomous driving, also known as a self-autonomous system, and video surveillance remain a trendy research topic with endless practical applications even though the errors are still happening in the real world.

Nevertheless, we have taken this technology beyond its peak point because of deep learning. Algorithms, over time, have managed almost to grasp any object presented in detection, segmenting and recognition.

Chapter 2

State of the art

The level of development, as of a device, procedure, process, technique, or science, reached at any particular time that uses the latest technological pieces.

2.1 Traffic speed radar

On most highways around the globe, we use a speed radar, either by police officers, cameras placed at high points to capture it or systems pre-installed to show each driver's speed.

When tracking the speed of vehicles on a segment of a road, the most vital steps for it are Vehicle Detection, Speed Estimation but also Capturing the vehicle as an image.

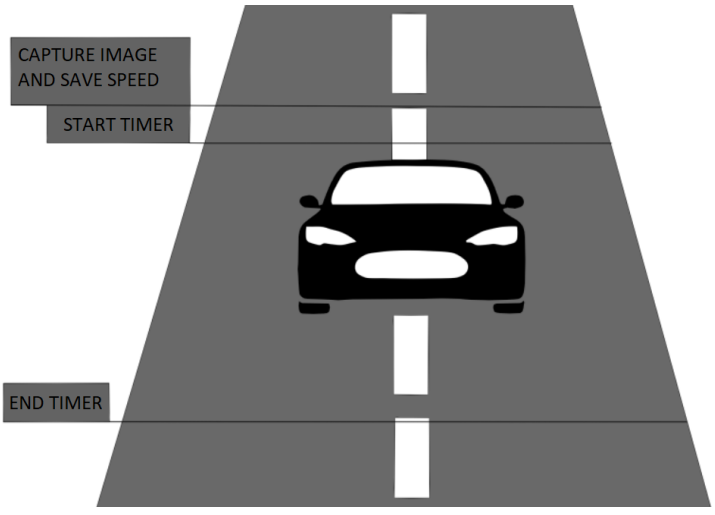

Figure 2.1: Speed Traffic

The vehicle would then be saved as an image and saved as Data.

2.2 Facial Recognition

Facial recognition is a biometric security category but also a biometric tool used to identify or confirm an individual's identity using their face. Facial recognition systems are used to identify people in photos, videos or even in real-time.

Examples of this biometric Software:

- ▷ Voice recognition
- ▷ Fingerprint recognition
- ▷ Eye retina/iris recognition

The technology is mainly used for security and law enforcement, as its interest has increased in many other working areas.

It does not rely on a massive database of photos to determine an individual's identity. It simply identifies and recognizes one person as the sole owner of the device while limiting access to others.

2.2.1 Usage

1) Face Detection:

We use a camera that can detect and locate the image of a face. It can detect within a crowded area with many other people.

2) Face Analysis:

The image of the face is captured and analyzed as most facial recognition Software relies on 2D rather than 3D images because it can match a 2D image with public photos more conveniently.

3) Converting the image to data:

The face capture process transforms analogue information into a set of digital data based on the person's facial features, thus turning the analysis into a mathematical formula. It then becomes a numerical code (faceprint).

4) Matching the faceprint:

The faceprint is compared against a database of other known faces. For example, in Facebook's photo system, any photo tagged with a person's name becomes a part of Facebook's database, which is used for facial recognition.

2.2.2 Face ID Detection

Using Face ID Detection, we can securely unlock the iPhone or iPad, authenticate the purchases, connect to apps, and more, all at a glance.

The phone itself would use an infrared camera (also known as a thermal imager) that can detect and measure the infrared energy of objects but also a dot projector that is much different from power-hungry laser beams, which traditionally use 3D Imagery.

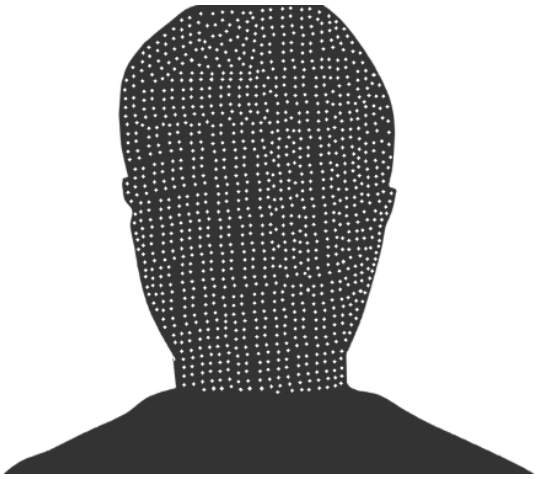

Figure 2.2: Face ID Detection

2.3 Medicine

2.3.1 Surgery Organ Detection

Organ detection is helpful for various medical applications, whether planning surgeries or finding pathologies. Real-time organ tracking can be profitable for adaptive radiotherapy or laparoscopic surgeries. Object detection models could help with these tasks.

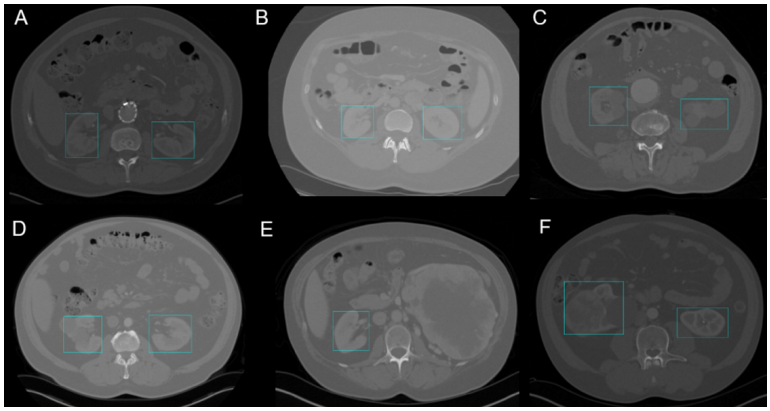

Figure 2.3: Kidney Organ Detector

In the picture above, we use the YOLO (”you look only once”) algorithm to detect a patient’s kidneys.

Using the YOLOv3 algorithm, the algorithm that the Real-time detection and marking of ”targets” in a video stream project employs, it is using it to detect 2D organs within a person’s body.

Using a computed tomography scanner, we have managed to get a good look at a patient’s kidneys:

- A - B: represents the scan of the kidneys with different mean intensities (unknown and potential hosts of bacteria/parasites)
- C - D: Cystic kidneys are round pouches of fluid forming on or in the kidneys
- E: Failed detection of the hypertrophied kidney
- F: Tumoral kidney

Chapter 3

Proposed Solution and Implementation

3.1 Used Technologies

The language used for this project is Python version 3.10.5 using the Visual Studio Code IDE (Integrated Development Environment), along with other essential packages:

- ▷ OpenCV (cv2)
- ▷ Imutils
- ▷ Numpy
- ▷ Time

Other packages included, though not essential overall. Its main driving force will be an algorithm called YOLO, an R-CNN family of algorithms that uses regions to localize the objects in images.

3.2 OpenCV

Before we can explain the purpose and functionality of the project, we need to talk about the main library of programming functions aimed at real-time computer vision, *OpenCV*.

OpenCV is an excellent tool for image processing and performing computer vision tasks. It is an open-source library that performs tasks such as Face Detection, Object Tracking, Landmark detection and much more. It can support multiple languages, including C++, Java, Python (the programming language I will be using for this project), etcetera.

The library is equipped with hundreds of valuable functions and algorithms that are freely available to us. These functions are standard and are used in almost every computer vision task.

3.3 YOLO Vision Algorithm

YOLO algorithm (You only look once) proposes using an end-to-end neural network that makes predictions of bounding boxes and class probabilities all at once. YOLO works by dividing the image into N grids, each having an equal dimensional region of S x S.

Each of these N grids is responsible for detecting and locating the object it contains.

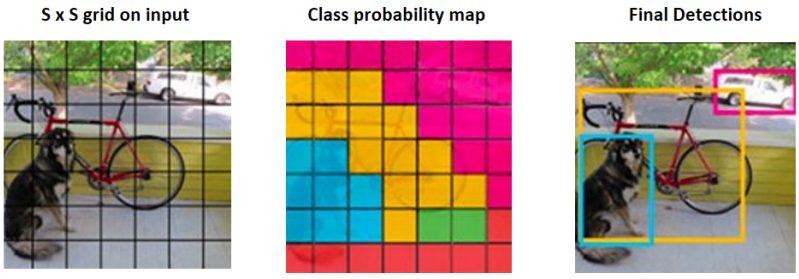

Figure 3.1: Visual Algorithm

Once the object instance has been detected, we can extract further information such as:

- Outline of features
- Objects within other objects
- Colour scheming
- Facial recognition

The YOLO algorithm employs convolutional neural networks (CNNs) to detect objects in real-time. As the name suggests, the algorithm requires only a single forward propagation (the way to move from the Input layer to the Output layer in the neural network) through a neural network that detects objects.

The prediction in the entire image is finalized in a single algorithm run. The CNN is used to predict various class probabilities and bounding boxes simultaneously.

The YOLO algorithm is vital due to many factors::

- ▷ YOLO improves detection speed because it can predict objects in real-time.
- ▷ The algorithm provides accurate results with minimal background errors.
- ▷ This algorithm has excellent learning capabilities that can enable it to learn the representations of objects and apply them in object detection.

3.3.1 Algorithm Comparison

YOLO v3 is the main component of this project; though not the fastest or strongest algorithm, it mainly uses more of the Central Processing Unit than the Graphic Card. The stronger the Graphic Card, the better the algorithm we can use compared to YOLO v4, which beats our algorithm in most fields.

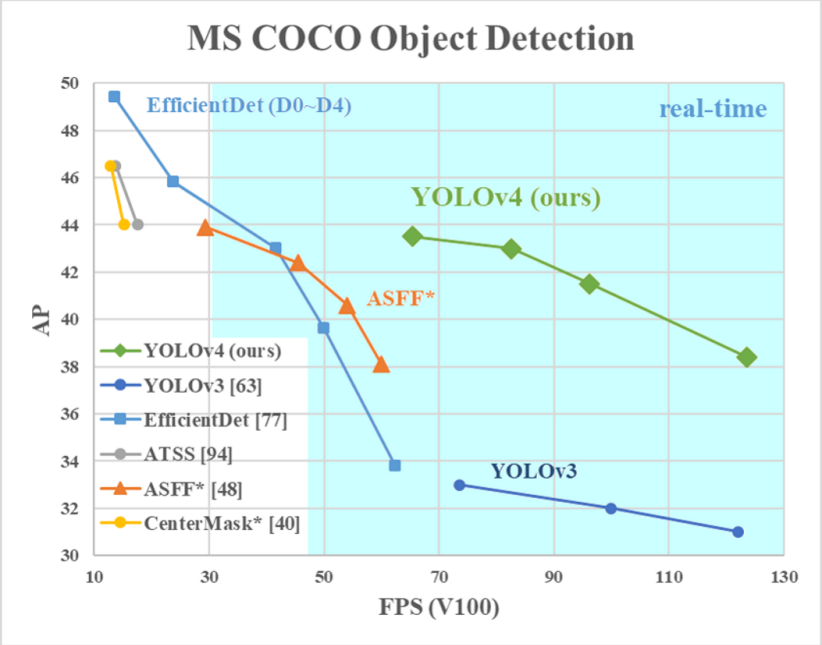

Figure 3.2: Algorithm Comparisons

The Frames Per Second(FPS) and the Average Precision (AP) should be balanced when trying to detect a giant video stream using the YOLO algorithm.

YOLOv3 also includes the Improved Anchor Box Algorithm with it's distance function between each prediction box and the reference standard box.

$$d(x) = \sum_i \sum_j 1 - IOU\left(box_i, truth_j\right)$$

Figure 3.3:

3.4 YOLO Operation

YOLO Algorithm works using three different techniques:

- ▷ Residual blocks
- ▷ Bounding box regression
- ▷ Intersection over Union (IOU)

3.4.1 Residual Block

Residual blocks are skip connection blocks that learn residual functions regarding the layer inputs instead of learning unreferenced functions.

To understand, inside a grid view, all bounding boxes would work together to detect the object that appears within them. If the object appears within a grid cell, the cell is now tasked with detecting the object.

3.4.2 Bounding box regression

The bounding boxes are rectangular boxes that contain the object and or its set points.

It is used to enclose a target and serve as a reference point for object detection, thus creating a collision box for that object.

3.4.3 Intersection over Union

Intersection over Union (IOU) is mainly used in applications related to object detection, where we train a model to output a box that fits perfectly around an object.

If the object is within the grid cell, it will be their responsibility to predict each bounding box and confidence scores. If the predicted bounding box is the same as the real box then the Intersection over Union will be equal to one, eliminating bounding boxes that are not equal to the real box.

3.5 Training YOLO

We need to talk about the poor results from collecting data straight toward model training. There may be problems with the data. Even if there are not, applying image augmentation expands the dataset and reduces overfitting.

Everything that we need to prepare images for Object Detection covers the following:

- Resize images if needed and update all interpretations to tie our new recent-sized image.
- Colour correction can mainly ruin the detection process, so to improve our model performance, we have to modify it.
- We examine if the annotations are correct.
- Probe if the exchangeable image file format, also known as EXIF, is accurate.
  Our detector needs to be fed with labelled training data to learn to detect objects in images. The images can either be existent ones that cannot be recognized or photos we want to add to the dataset.
  After that, we can train our model using the pre-trained weights file, and then we can start converting it into a preferable format, YOLO.

```
python Train_YOLO.py
```

We use the following above command to start training our detector, though it might take a few hours to a few minutes depending on our processing power.

3.6 YOLO Downsides

You Only Look Once has some quite severe limitations and drawbacks:

- Objects that collide with one another can cause a misinterpretation
- It cannot handle small objects
- Its limitation is the algorithm itself

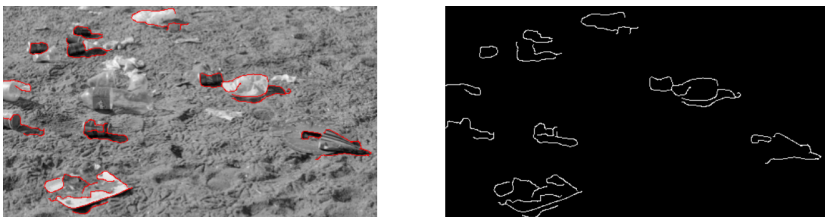

Figure 3.4: Canny Edge Detector

The YOLO algorithm object detector divides an image or a video stream into an SxS grid. Each cell in the grid is predicted with only one object. If there are multiple small objects in a single cell, YOLO will not be able to detect them, causing missed object detections.

$$\lambda_{\textbf{coord}} \sum_{i=0}^{S^4} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Figure 3.5:

The model is trained on ImageNet divided by one thousand datasets. However, this function gives equal weight to the classification and localization tasks.

The dataset should not use the YOLO algorithm because small objects are grouped.

This is also why the project itself is hybrid when it comes to tracking one object that can be small and multiple objects that are big.

In terms of smaller objects, the algorithm Faster R-CNN works perfectly in this regard, but it is also the slowest.

SSDs also used in this field, but they can struggle even more with smaller objects than YOLO. However, trading off speed and accuracy quite well.

3.7 Weights - Configurations

The project at hand using the YOLO Algorithm can go up to 80 words, objects that are registered in the project. Nevertheless, the algorithm can teach different words that can be later used.

Now we need to talk about weights and different types of configuration files. The project, and YOLO itself, can be adjusted in terms of Average Precision and FPS, and as such, the project uses the 320 weights and configuration file. Other files similar to it can also

be used through the output but would not give us a balanced mix of speed and accuracy. Tiny is another file that can be used though it is dismissed due to its low accuracy.

After analyzing the video streams with different model configurations, we notice a significant change in average precision and speed:

- YOLOv3 tiny: 12.50 FPS, Low AP
- YOLOv3 320: 3.5 FPS, High AP
- YOLOv3 416: 3.3 FPS, High AP
- YOLOv3 608: 3 FPS, High AP

3.8 Algorithm Comparisons

However, YOLO is not the only algorithm out there for Object Detection. There are several, but we will only talk about three important ones in this project.

3.8.1 Fast R-CNN

This training algorithm for object detection is written in Python but also C++, also known as the Fast Region-based Convolutional Network method, or Fast R-CNN for short. It mainly focuses on the disadvantages of algorithms that it is trying to overcome, improving the speed and accuracy.

- It has higher detection quality than the average R-CNN algorithm
- Uses a multi-task loss to train in a single-stage
- Disk Storage is not needed for stashing data
- All networks are updatable after the training

3.8.2 Histogram of Oriented Gradients

A histogram of oriented gradients, also known as HOG, is a feature descriptor utilized to detect objects in image processing and other computer vision techniques. The Histogram of oriented gradients descriptor technique includes occurrences of gradient orientation in localized portions of an image, such as the detection window and the region of interest (ROI). Its main advantage is its simplicity, making the information they carry easier to understand.

In the HOG feature descriptor, the distribution (histograms) of directions of gradients (oriented gradients) are used as features. Gradients (x and y derivatives) of an image are helpful because the magnitude of gradients is large around edges and corners (regions of abrupt intensity changes), and we know that edges and corners pack in a lot more information about object shape than flat regions.

3.8.3 Single Shot Detector

Finally, the Single Shot Detector, also known as SSD, is used for detecting objects in images using one deep neural network. The SSD approach discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios.

It is significantly faster in speed and high-accuracy object detection algorithm. It is a quick comparison between the speed and accuracy of different object detection models.

Chapter 4

Application presentation

We can now begin to talk about the Real-time detection and marking of ”targets” in multiple video streams. All videos have been placed in a folder, and our project can switch between them at leisure.

The project can also switch between ”states”, a Multi-Target or Single-Target object detection system.

Multi-Target will detect as many objects on the screen as possible, trying to find the best accuracy in each frame. However, the program needs a solid graphics card and a better version of YOLO v4, though this process is quite complicated.

Single-Target draws a box around an object and tries to track it to the end of the bounding box, forcing it to stop moving after that. It tracks down not only the object but also its coordinates in real-time.

Finally, we can talk about the buttons used in our project. At the start of the application, as said before, our project can swap between the two systems, switch between videos, and finally restart them:

- ’r’ - Restart project
- ’t’ - Swap between systems
- ’x’ - Scroll videos through the right side of the folder
- ’z’ - Scroll videos through the left side of the folder

As a final touch in the project, an FPS meter is added on the top left of the screen for the Multi Tracking system to showcase the issues that slow down the video target stream. CPU and YOLO v3 can sustain decent average FPS. However, upgrading the system would take a lot of costs and time, including adding additional packages that could substantially slow down the process.

4.1 Multi-Tracker

The first system that shall be reviewed will be the Multi Tracker system. Its purpose is to track the separated list of objects from a file called coco.names, 80 objects in fact (car, motorcycle, bike, person, etcetera). It displays the object's name for each one it can analyze during the video target stream and gives a percentage accuracy to each visible object on the screen.

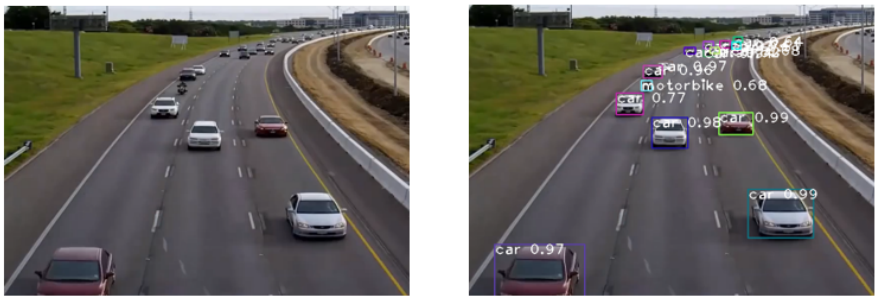

Figure 4.1: Multi Tracker

The above picture, we can notice that all vehicles have been detected. The Multi Tracker system uses an array and tracks every single object given on the screen. It removes as many redundant boxes as possible, thus displaying the virtual objects.

Our list, in this case, is the following::

Total objects on screen: 35

Array list: [7 32 17 4 24 0 26 18 14 29 15 28 20 13 19 16]

Redundant objects: 19

That was all about detecting objects in the real world through it can also detect objects inside a simulated virtual environment. Furthermore, this is where accuracy can be misinterpreted because virtual objects might not look exactly, for example, like a human. However, there are some similarities, and it pinpoints the objects.

Figure 4.2: Virtual People

4.1.1 Comparisons

In the picture previously presented, we finally talk about the displayed accuracy on the screen, the accuracy representing what percentage that object is presented as the real thing.

With what has been presented in the Artificial Intelligence section at the very start of this presentation, we want to tackle the subject of comparison between my YOLO Object Detection project and the Auto Pilot systems that Uber, Tesla, and many other companies use.

Accuracy is essential, else mistakes will happen. The project itself does its best to figure out if the vehicle is in fact the intended vehicle, but if the object is far away, it can misinterpret as something else.

Artificial Intelligence, at its core, is not perfect, and humans make it in the end. Even if self-taught, it was made to learn how humans taught it to do its job. Mistakes over the following years will happen unless we test more and more, do research over each test run and release the system only once the data presents itself as suitable for launch.

The recording can be either about traffic in real life or even a simulated one. The project can figure out that a 3D blended car or drawing is still somewhat a vehicle, highlighting it nevertheless. Even 2D pictures of cars can work effectively.

4.2 Code Multi Track

The code below represents a global call for a couple of variables that guide us towards our Multi Tracker folder, where all videos are stored.

```
MUFolder = 'C:\\Users\\Dan\\Documents\\Code\\Targets\\MUTracker'
os.chdir(MUFolder)
files = os.listdir(MUFolder)
MUarray = []
MUindex = 0
for f in files:
if f.endswith('.mp4'):
MUarray.append(f)
```

Our variable "files" takes the name of each video stream, and then we use it inside a for loop, adding it to an empty Multi Tracker array. However, there is also an empty index variable that we will use later to increment or decrement to find a specific video inside the director.

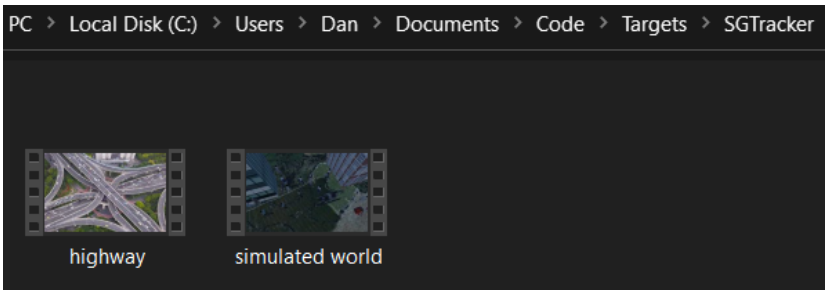

Figure 4.3: Recordings

```
cTime = time.time()
fps = 1 / (cTime - pTime)
formattedFPS = "{:.2f}".format(fps)
pTime = cTime
cv2.putText(img, f'FPS: {float(formattedFPS)}', (20, 70), cv2.
    FONT_HERSHEY_PLAIN, 3, (255, 0, 0), 2)
```

The FPS meter displays at the top left side of the screen as a constantly changing value that is a float number. The FPS variable becomes the subtraction between cTime and pTime, start time and end time (which is 0 at the beginning every time), and then we divide one with the output of the subtraction.

```
boxes = []
confidences = []
class_ids = []

for output in layersOutputs:
  for detection in output:
    scores = detection[5:]
```

Figure 4.4: FPS Meter

```
    class_id = np.argmax(scores)
    confidence = scores[class_id]
    if confidence > 0.5:
      center_x = int(detection[0]*width)
      center_y = int(detection[1]*height)

      w = int(detection[2]*width)
      h = int(detection[3]*height)

      x = int(center_x - w/2)
      y = int(center_y - h/2)

      boxes.append([x, y, w, h])
      confidences.append((float(confidence)))
      class_ids.append(class_id)
```

There are written variables that would inherit empty lists made to display the boxes drawn on each object and the predicted classes for each ID.

To extract the bounding boxes, confidences and predicted classes, we need to create two loops that help us loop over the output of the layer.

The first for loop is extracting all the information from layerOutputs, and the second one is used to extract the information from each output.

Each of the detection variable outputs should contain four bounding box offsets. We can consider that the first four elements are the locations of the bounding boxes, and the fifth element is the box confidence that reflects how likely the box contains objects and how accurate the boundary box is and contains the number of classes of predictions of the probability of that as well.

We use the scores variable to store all the active class predictions that start from these six elements to the end.

Then we want to identify the classes that have the maximum scores of the highest scores. The numpy argument, max, is used to get the maximum score location, so after that, we extract the higher scores and then assign them to the confidence variable.

```
center_x = int(detection[0]*width)
center_y = int(detection[1]*height)

w = int(detection[2]*width)
h = int(detection[3]*height)

x = int(center_x - w/2)
y = int(center_y - h/2)
```

All of this is made because we want to ensure the predictions have a high enough confidence for us to consider that object is being detected. Thus we make an if loop where we check if the confidence is more significant than the probability of 0.5, then we can use our predictive boxes that contain the first four values of the detection variables: x, y, w (width) and h (height).

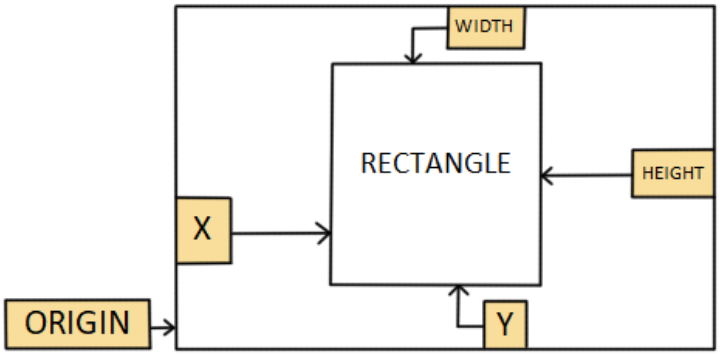

Figure 4.5: Bounding Box

We have normalized the image functions within the code, so in order for us to scale it back down, we need to use the height and width that we extracted from the original image. Thus we multiply each variable by width or the height depending on each variable.

We do this because the YOLO Algorithm predicts the result with the centres of the bounding boxes, so we need to extract the upper left corner positions to present them with OpenCV.

If we print the boxes in the console, we receive the number of objects that are not redundant.

Finally, we can simply apply all the pieces of information to the corresponding leads.

```
boxes.append([x, y, w, h])
confidences.append((float(confidence)))
class_ids.append(class_id)
```

When we perform Object Detection, it happens that very often, we will have more than one box in the same object, so we use the function called the non-maximum suppressions

to only keep the highest scores boxes. We need to pass four parameters into the functions containing all the boxes and their corresponding confidence, so we will filter boxes by scores and a first hole that will be used in the non-maximum suppressions.

4.3 Single-Tracker

In the images below, initially inside the project being a video stream recording, we can see many objects that are presented, mainly vehicles (in a different recording, some objects even interact with each other).

For following an object, we are going to need a tracker firstly. In OpenCV, there are multiple types of trackers that we can use that track and object differently depending on the specific need or preference, such as:

```
cv2.TrackerBoosting_create
cv2.TrackerCSRT_create
cv2.TrackerMedianFlow_create
cv2.TrackerMIL_create
cv2.TrackerMOSSE_create
cv2.TrackerKFC_create
cv2.TrackerTLD_create
```

Personally, and for this specific project about multiple object detection and single-tracking, the best tracker that felt perfect for this job was the CSRT Tracker, as it can track fast objects, objects that interact with other objects, and even if the object changes width or height.

Now the Single tracker can draw a box around any object displayed on the video stream that is currently paused. After drawing the box, the display finally shows the x y coordinates so we can view where precisely on the screen our object is in real-time.

If the object is out of bounds, the tracker with the bounding box itself is left behind; thus, we have to restart the program or stop if satisfied with the given results.

As we can see above, coordinates are displayed on the screen after drawing a bounding box over our vehicle, following the object through its route in real-time.

Start: X = 188 , Y = 629

Destination: X = 636 , Y = 436

We can also use this system again inside simulated worlds and virtual worlds such as video games.

Some flaws make the Single-Tracker less advantageous due to collisions or losing sight of the object on the screen. It is hard to figure out how to constantly track the objects due to changes in size and appearance. Not even a given ID to that object works because it changes itself every frame regardless. The tracker can get lost or jump from object to object if the bounding box is not good enough.

Figure 4.6: Object Collision

When trying to determine if a collision occurs between two objects, we generally do not use the vertex data of the objects themselves since these objects often have complicated shapes.

In turn, it makes collision detection complicated. For this reason, it is a common practice to use more simple shapes for collision detection that we overlay on top of the original object. We then check for collisions based on simple shapes.

Simple shapes give us more straightforward and more efficient collision detection algorithms. However, they share a common disadvantage:

- These shapes usually do not fully surround the object
- A collision may be detected even though a collision might never happen due to the background as well inside a video stream
- These shapes are just approximations of the fundamental shapes and thus does not cover the entire object perfectly

4.4 Code Single Track

The code below represents a global call for a couple of variables that guide us towards our Single Tracker folder, where all videos are stored.

```
SGFolder = 'C:\\Users\\Dan\\Documents\\Code\\Targets\\SGTracker'
os.chdir(SGFolder)
files = os.listdir(SGFolder)
SGarray = []
```

```
SGindex = 0
for f in files:
if f.endswith('.mp4'):
SGarray.append(f)
```

Similarly, as the Multi Tracker, our files variable takes the video file's name, and within the for loop, we append it inside the Single Tracker empty array. Then with an empty integral variable, we can, in the future, change the videos at will by pressing the 'x' or 'z' buttons.

```
def drawBox(img, bbox):
x,y,w,h = int(bbox[0]),int(bbox[1]),int(bbox[2]),int(bbox[3])
cv2.rectangle(img,(x,y),((x+w),(y+h)),(0,0,0),2,1)
# show x,y coordinate:
cv2.putText(img,"X =", (0,30),cv2.FONT_HERSHEY_COMPLEX,0.7,(255,0,0),2)
cv2.putText(img,str(int(x)), (40,30),cv2.FONT_HERSHEY_COMPLEX
    ,0.7,(255,0,0),2)
cv2.putText(img,"Y =", (100,30),cv2.FONT_HERSHEY_COMPLEX,0.7,(255,0,0),2)
cv2.putText(img,str(int(y)), (140,30),cv2.FONT_HERSHEY_COMPLEX
    ,0.7,(255,0,0),2)
```

We are drawing the box for the Single-Track and inputting all of the coordinates we need to track a given object at any time.

```
os.chdir(SGFolder)
cap = cv2.VideoCapture(array[index])
while True:
  key = cv2.waitKey(50)
  success, img = cap.read()
  img = imutils.resize(img,width=1350)
  success,bbox = tracker.update(img)
```

Now the tracker can work on any given object, but we still need a way to follow the vehicle object throughout the whole recording. We will use two variables to get our captured recorded video inside the folder.

We then update the two variables each time during the while loop; therefore, the tracker on the object that we wish to follow will no longer be static but follow it.

4.5 Multi-Usage Code

Many lines of code might be similar between the Multi Track and the Single Track systems. This section has been explicitly made, so lines of code do not repeat within the presentation.

Line of code that was made for all buttons that have been listed before to switch between Multi-Track and Single-Track, swap video files within two different folders, restart the specific system project and finally close the application down.

```
key = cv2.waitKey(1)

if key==27:
  break
if cv2.waitKey(33) == ord('r'):
  cap = cv2.VideoCapture('car.mp4')
elif key == ord('z'):
  cap.release()
  cv2.destroyAllWindows()
  index -= 1
  MultipleTracker(array,index)
elif key == ord('x'):
  cap.release()
  cv2.destroyAllWindows()
  index += 1
  MultipleTracker(array,index)
elif key == ord('t'):
  cap.release()
  cv2.destroyAllWindows()
  SingleTracker(SGarray,SGindex)
```

4.6 UML Diagram

Now let us talk about the code itself. The project is split into three functions and our global variables that are used for later. Our primary function that starts the SingleTracker, the SingleTracker function, which draws a bounding box on an object, finds its location within the x and y coordinates while the recording is still playing, and it can switch to the MultipleTracker. Finally, the MultipleTracker function allows us to track multiple objects on the screen, giving them a particular ID and counting the FPS, all of this also being on while the recording is still playing.

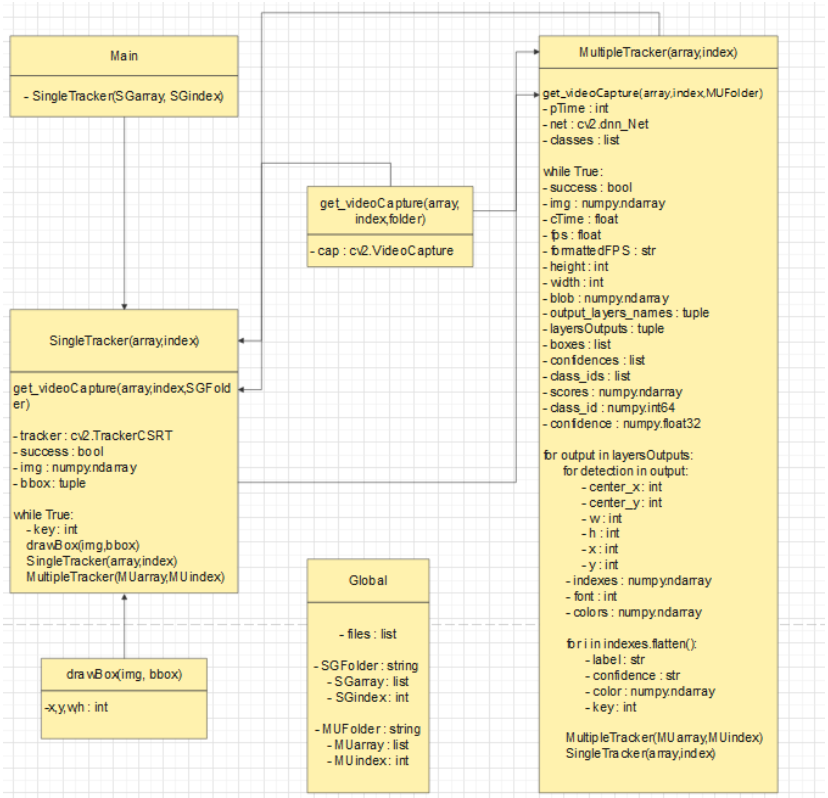

Figure 4.7: UML Diagram

4.7 Difference between Systems

The Multi-Object tracker and the Single-Object tracker are pretty different in approach but similar in speed since they are both one hybrid being.

Most advantages lie within the Multi-Object tracker automatically due to its multi-tasking job to pinpoint every object displayed on-screen while drawing a bounding box over every single one, but there is a big catch. We are sacrificing much speed to get the best accuracy out of it, while the Single Object tracker that tracks even the coordinates is a bit faster.

- ▷ Multiple objects tracked at once
- ▷ High accuracy with original ID given to each object
- ▷ Slower, due to high costs of components

This is not something spectacular, but for specific reasons, the Single-Object tracker can be helpful in other ways. While testing the algorithm onto a fast and size-changing object, the system managed to surprise. Due to its CSRT tracker, it manages to hold onto that speeding object throughout the recording, though some pixels or other objects interacting with our object lead to losing track of it. Worst case scenario, it just starts following something else

entirely.

- ▷ Faster even without the need for expensive components
- ▷ Accuracy is average, but most of the time, running a perfect lap of the video stream

Its tracker changes size for each frame it captures in the video frame, and each time it is more random because we cannot get the exact coordinates perfect each time unless we predetermine them before the start of the recording.

Chapter 5

Conclusion

With all that said, with what we have gotten thus far, Artificial Intelligence and Object Detection are still not perfect, and it is a growing branch in programming that will continue to rise over time.

Real-time detection can be seen worldwide, whether on highways, streets, malls, stores, medical checkups and recognition apps for the phone.

The thought process behind this project was primarily to grasp the algorithm YOLO's benefits, advantages and disadvantages, and results. Machine Learning is used in many jobs in the real world and even in simulated worlds, and also believe it will never stagnate.

It is, though, unwise to compare results side-by-side from different papers. Every field they are tested on is relatively specific, with the purpose of another respective algorithm in mind. Those experiments are also done in different settings which are not purposed for side-by-side comparisons.

YOLO is not the perfect algorithm that could have been used, but in a project like this using weights, configurations, and the coco names have proved to be easier to use, and training the algorithm to learn new data images or new ones is not complex either. It is tough to have a fair comparison among different object detectors. There is no straight answer on which model is the best. For real-life applications, we make choices to balance accuracy and speed.

Though in the real world, companies have made it quite hard due to constant demand for quick release of technology, thus becoming unsafe. In the self-driving department, there will be many, many errors to come. Even in the making of this project, there were hardships, but the research that went into doing the project was worth the information gathered.

The scope of this application is to detect objects in a video stream target. While tracking it, we put a bounding box over each one, either selected in the Single Tracker's case or in the Multi Tracker, where every objects that can be detected can be tracked. However, most importantly, the application would follow real-life objects, such as simulated people, the same as detecting and virtual world objects.

Speed was never the point of the project due to costs, complicated Microsoft packages, and high-demanding GPU, the CPU being the project's driving force. Accuracy felt essential, and tried to see if simulated models of humans, objects and many more would also be detectable by the Multiple Object Tracker even in the real world.

Safety should be more tightened while keeping people's privacy. Nevertheless, with time the topic of detecting objects will grow due to high demand in many fields, and as we research, test more, and write more about the subject, we might just fix many things.

Bibliography

[1] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[2] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. A survey of deep learning-based object detection. *IEEE Access*, PP:1–1, 09 2019.

[3] Zhong-Qiu Zhao, Peng Zheng, Shou-Tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, PP:1–21, 01 2019.

[4] Fatih Porikli and Alper Yilmaz. *Object Detection and Tracking*, volume 409. 01 2012.

[5] Baohua Qiang, Ruidong Chen, Mingliang Zhou, Yuanchao Pang, Yijie Zhai, and Minghao Yang. Convolutional neural networks-based object detection algorithm by jointing semantic segmentation for images. *Sensors (Basel, Switzerland)*, 20, 09 2020.

[6] Chandan .G, Ayush Jain, Harsh Jain, and Mohana Mohana. Real time object detection and tracking using deep learning and opencv. pages 1305–1308, 07 2018.

[7] Upulie Handalage and Lakshini Kuganandamurthy. Real-time object detection using yolo: A review. 05 2021.

[8] Raffaele Rodogno and Marco Nørskov. *The automation of ethics: The case of self-driving cars*. 08 2019.

[9] Piotr Dollar, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: An evaluation of the state of the art, 2012.

[10] Paul Pilkington and Sanjay Kinra. Effectiveness of speed cameras in preventing road traffic collisions and related casualties: Systematic review. *BMJ (Clinical research ed.)*, 330:331–4, 03 2005.

[11] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. pages 3354–3361, 2012.

[12] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Zitnick. Microsoft coco: Common objects in context. 05 2014.

[13] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. 11 2016.

[14] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. pages 779–788, 2016.

[15] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. 1:886–893 vol. 1, 2005.

[16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander Berg. Ssd: Single shot multibox detector. 9905:21–37, 10 2016.

[17] Sheng He and Genke Yang. *Image Denoising Networks with Residual Blocks and RReLUs*, pages 60–69. 12 2019.

[18] Degang Sun, Yang Yang, Min Li, Jian Yang, Bo Meng, Ruwen Bai, Linghan Li, and Junxing Ren. A scale balanced loss for bounding box regression. *IEEE Access*, PP:1–1, 06 2020.

[19] Jiabo He, Sarah Erfani, Xingjun Ma, James Bailey, Ying Chi, and Xian-Sheng Hua. Alpha-iou: A family of power intersection over union losses for bounding box regression, 10 2021.

[20] Hu's lost. World's most amazing elevated highways in Shanghai. `https://www.youtube.com/watch?v=fhpTP06VBxU/`, 2018.

[21] Andréanne Lemay. Kidney Recognition in CT Using YOLOv3. `https://www.arxiv-vanity.com/papers/1910.01268//`.

[22] Techzizou. YOLOv4 vs YOLOv4-tiny. `https://techzizou.com/yolov4-vs-yolov4-tiny-custom//`.

[23] YOLO : You Only Look Once – Real Time Object Detection. `https://www.geeksforgeeks.org/yolo-you-only-look-once-real-time-object-detection//`.

[24] Zhenrong Deng, Rui Yang, Rushi Lan, Zhenbing Liu, and Xiaonan Luo. Se-iyolov3: An accurate small scale face detector for outdoor security. *Mathematics*, 8(1), 2020.