Pedestrian Detection & Trajectory Prediction Using Computer Vision

Master of Engineering in Electronic and Computer Engineering

Master's Project
Final Report

Ronan Keaveney
19125356

Ciaran Eising

26 March 2023

Abstract

In recent years, we have seen rapid technological advances in the spaces of autonomous driving and advanced driver-assistance systems (ADAS). Using technologies such as radar, lidar, inertial measurement units and cameras, engineers are slowly but surely making roads a safer place, especially for the most vulnerable road users: pedestrians and cyclists. One of the biggest and most pressing issues facing the autonomous driving industry is the need for a system which can effectively detect pedestrians from the landscape in front of the vehicle, and accurately predict their intentions and trajectory. Of course, this ability comes naturally to human beings, who evolved to be excellent at object detection, and can use subtle indicators such as a pedestrian's velocity, pose, gaze and other nuanced cues to help inform their decision on what a pedestrian is about to do. This task is much more difficult to emulate using a machine, and for many years it was even thought to be impossible. However, as we will explore in this report, this dream is quickly becoming a reality, thanks to recent advances in artificial intelligence and computer vision.

In this project, I will investigate the potential for using computer vision and artificial intelligence techniques such as convolutional neural networks (CNNs) and long short-term memory (LSTM), to detect and predict the trajectory of pedestrians using 2D video imagery from a car's on-board camera. I will also discuss and compare some of the other approaches which have been employed in the past and could be used in the future to solve this problem. My project will be divided into two separate models: a pedestrian detection model which uses the YOLOv7 framework to perform single-stage object detection on the images to detect and localise pedestrians; and a pedestrian trajectory prediction model which uses a pretrained convolutional model and LSTM to predict a pedestrian's trajectory in the next one second based on a second of past data.

Declaration

This report is presented in part fulfilment of the requirements for the LM806 Master of Engineering in Electronic and Computer Engineering **Final Year Project**.

It is entirely my own work and has not been submitted to any other University or Higher Education Institution or for any other academic award within the University of Limerick.

Where there has been made use of work of other people it has been fully acknowledged and referenced.

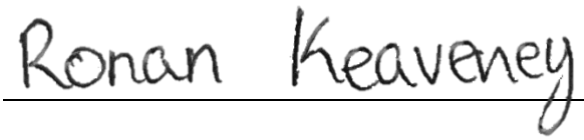
Name	Ronan Keaveney
Signature	
Date	02/03/2023

Table of Contents

List of Figures

List of Tables

Algorithm	mAP (%)	Inference Speed (fps)	Algorithm Type
YOLOv1	63.4	45	One-stage
Fast R-CNN	66.9	0.5	Two-stage
Faster R-CNN	73.2	7	Two-stage
SSD	74.3	59	One-stage

Table 1: Object detector performances on PASCAL VOC 2007 dataset (Titan X GPU) [1]

	Performance (ADE/FDE)					
Deterministic	ETH	HOTEL	ZARA1	ZARA2	UNIV	AVERAGE
LR	1.33/2.94	0.39/0.72	0.62/1.21	0.77/1.48	0.82/1.59	0.79/1.59
LSTM	1.13/2.39	0.69/1.47	0.64/1.43	0.54/1.21	0.73/1.60	0.75/1.62
S-LSTM[1]	0.77/1.60	0.38/0.80	0.51/1.19	0.39/0.89	0.58/1.28	0.53/1.15
CIDNN[49]	1.25/2.32	1.31/1.86	0.90/1.28	0.50/1.04	**0.51/1.07**	0.89/1.73
SocialAttention [45]	1.39/2.39	2.51/2.91	1.25/2.54	1.01/2.17	0.88/1.75	1.41/2.35
TrafficPredict [38]	5.46/9.73	2.55/3.57	4.32/8.00	3.76/7.20	3.31/6.37	3.88/6.97
SR-LSTM [53]	0.63/1.25	0.37/0.74	**0.41/0.90**	0.32/**0.70**	**0.51**/1.10	0.45/0.94
STAR-D	**0.56/1.11**	**0.26/0.50**	**0.41/0.90**	**0.31**/0.71	0.52/1.15	**0.41/0.87**
Stochastic	ETH	HOTEL	ZARA1	ZARA2	UNIV	AVERAGE
SGAN† [16]	0.81/1.52	0.72/1.61	0.34/0.69	0.42/0.84	0.60/1.26	0.58/1.18
SoPhie*† [40]	0.70/1.43	0.76/1.67	0.30/0.63	0.38/0.78	0.54/1.24	0.54/1.15
STGAT† [21]	0.65/1.12	0.35/0.66	0.34/0.69	0.29/0.60	0.52/1.10	0.43/0.83
STAR†	**0.36/0.65**	**0.17/0.36**	**0.26/0.55**	**0.22/0.46**	**0.31/0.62**	**0.26/0.53**

Table 2: Comparison of Yu's transformer-based pedestrian trajectory prediction model to state-of-the-art. STAR-D denotes deterministic version of STAR. [2]

	YOLOv7	YOLOv5
Precision	0.870	0.795
Recall	0.670	0.508
mAP@0.5	0.765	0.597
mAP@0.5:0.95	0.411	0.293
Inference Speed (ms)	16.3	29.3

Table 3: Test Statistics: YOLOv7 (my model) vs YOLOv5

Class	Instances	Precision	Recall	mAP@0.5	mAP@.5:.95
All	94677	0.885	0.676	0.758	0.449
Pedestrian	19644	0.869	0.671	0.765	0.410
Cyclist	694	0.873	0.591	0.676	0.379
Vehicle	74559	0.914	0.766	0.834	0.557

Table 4: Object detection testing stats for all classes

	ConvLSTM	Linear Regression
ADE	19.954	23.728
FDE	29.029	43.753

Table 5: ADE & FDE: ConvLSTM (my model) vs Linear Regression

Chapter 1 Introduction

1.1 Background & Motivation

41 pedestrians died on Irish roads in 2022, making up over 26% of all road traffic accident fatalities in Ireland last year. This alarming statistic underscores the urgency of developing effective solutions to minimize pedestrian fatalities and improve overall road safety. As urbanization continues to accelerate and cities around the world become increasingly congested, the need for advanced pedestrian detection and trajectory prediction models has never been more critical. Consequently, the primary motivation for choosing this project is the urgent need to save lives and reduce the number of pedestrian-related accidents on Irish roads, as well as to contribute to the global efforts aimed at enhancing road safety.

The rapid growth of urbanization and increasing traffic congestion create an urgent need for intelligent transportation systems (ITS) that can respond to these challenges. The development of advanced driver assistance systems (ADAS) and the eventual transition to autonomous vehicles require the ability to detect and predict pedestrian behaviour accurately in real-time. A robust real-time pedestrian detection model and pedestrian trajectory prediction model can play a critical role in enhancing the safety features of these systems.

The massive surge in development of autonomous vehicles, which we have seen in recent years, necessitates sophisticated pedestrian detection and trajectory prediction systems. As the automotive industry moves towards full automation, ensuring the safety of all road users becomes paramount. Autonomous vehicles must be able to identify pedestrians and anticipate their movements to avoid collisions and maintain safe interactions. By improving the performance of pedestrian detection and trajectory prediction models, this project aims to contribute to the development of more reliable and secure autonomous driving systems.

Furthermore, pedestrian detection and trajectory prediction have numerous practical applications beyond road safety. These models can be employed in smart cities to design more efficient and pedestrian-friendly urban environments, optimizing traffic signal timings, and promoting sustainable modes of transportation. Additionally, they can be used in

surveillance systems, robotics, and virtual reality environments to improve the overall user experience.

Recent advancements in computer vision, deep learning, and sensor technologies have opened new avenues for research in the field of pedestrian detection and trajectory prediction. Despite these advancements, several challenges remain to be addressed, such as occlusions, unpredictable pedestrian behaviour, and varying environmental conditions. By focusing on developing a real-time pedestrian detection model and pedestrian trajectory prediction model, this project aims to contribute to the ongoing research in this area by addressing these challenges and providing a comprehensive solution.

The choice of this project is further motivated by the potential societal impact of reducing pedestrian fatalities and injuries. The emotional, financial, and social costs associated with road traffic accidents are significant, affecting not only the victims and their families but also society as a whole. By developing more accurate and reliable pedestrian detection and trajectory prediction models, the project aims to contribute to reducing these costs and improving the overall quality of life.

In conclusion, this project is motivated by the urgent need to reduce pedestrian fatalities and improve road safety, as well as to contribute to the broader applications of pedestrian detection and trajectory prediction models in various industries. By addressing the challenges in this field and developing a real-time pedestrian detection model and pedestrian trajectory prediction model, the project seeks to make a meaningful contribution to both academia and society, ultimately enhancing the safety and well-being of pedestrians on Irish roads and beyond.

1.2 Project Overview

Perhaps the biggest and most pressing challenge facing the automotive industry in the coming years is the necessity for a system which can effectively detect pedestrians in the field of view of an autonomous vehicle and accurately predict their trajectory. While this ability comes very naturally to human drivers, who evolved to be excellent real-time object detectors and can intuitively use cues such as pedestrian velocity, body language and gaze to effectively predict their future behaviour, it is a much more difficult task to train a machine to perform the same

task. However, with the monumental jumps we have seen in artificial intelligence capabilities in the past decade or so, this challenge is looking increasingly achievable each year. In this project, I propose a computer vision-based approach to pedestrian detection and trajectory prediction, using only 2D imagery from a car's on-board front facing camera.

My proposed solution is composed of two distinct models. The first is a pedestrian detection model, which takes in a constant feed of video frames from the car's front facing camera and outputs bounding box co-ordinates for all pedestrians in the field of view, in real-time. The second model, for pedestrian trajectory prediction, takes a sequence of these images and bounding box co-ordinates as inputs and attempts to predict the pedestrians' position for a sequence of a predefined number of frames into the future.

1.3 Report Structure

This report is organised into six chapters, each focusing on a different aspect of my research into creating a pedestrian detection and trajectory prediction model using computer vision. The chapters which follow this introduction are as follows:

Chapter 2: Literary Review – This chapter presents a comprehensive review of the existing literature relevant to pedestrian detection and trajectory using computer vision. I have divided this chapter into two sub-chapters for my two distinct models. In each of these sub-chapters, I have begun by examining some of the classical methods that have been used in the past to tackle these challenges. For the most part, these are techniques which were popular before the advent of deep learning networks. Then, I move on to discussion of some of the more state-of-the-art approaches to these tasks, discussing recent studies in the field and comparing pros and cons of the various approaches.

Chapter 3: Background Theory – In this chapter, I outline some of the theoretical fundamentals of various machine learning techniques which are referenced in this paper and are used in my models. I have attempted to cater to those who do not have a background in machine learning by explaining some of the foundational concepts before moving on to a high-level outline of the computer vision tools I have used in my model.

Chapter 4: Methodology – This chapter details the methodology employed in the development of my pedestrian detection and trajectory prediction models. It outlines the

language and coding environment I used, the data collection and pre-processing procedures, as well as the architecture and implementation of the proposed models. I will also outline the baseline models which I will be using to compare my models' performance against.

Chapter 5: Results – This chapter covers the results of the experiments conducted to evaluate the performance of the proposed pedestrian detection and trajectory prediction models. A thorough analysis of the results is provided, including a comparison with a baseline model and existing state-of-the-art methods, an assessment of the model's strengths and weaknesses, and insights into the factors contributing to the model's performance.

Chapter 6: Conclusion – The final chapter summarizes the main findings of the research and discusses their implications for pedestrian safety and autonomous vehicle development. It also highlights the limitations of the study and suggests potential avenues for future research in the field of pedestrian detection and trajectory prediction using computer vision techniques.

Chapter 2 Literature Review

In this chapter, I will discuss some of the methodology which has been used in the past to tackle the problems of pedestrian detection and trajectory prediction. I will then move on to some of the more cutting-edge techniques, before weighing up the options and deciding on which approach I will use.

2.1 Pedestrian Detection

Object detection is one of the most fundamental and challenging problems in computer vision and machine learning more broadly. In the past two decades, we have seen massive technological advances revolutionise the space. Interest in the realm of object detection has also grown rapidly, with the number of publications on the topic growing steadily year over year, as shown in the figure below [3].

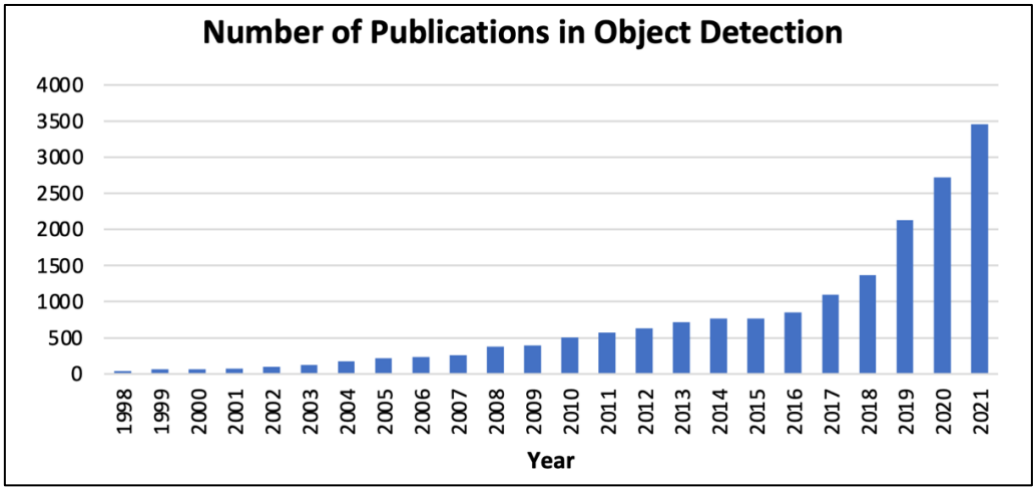


Figure 1: The increasing number of publications in object detection from 1998 to 2021 [3]

The history of object detection can broadly be divided into two distinct eras: the traditional object detection era (before ~2014) and the deep learning-based object detection era (after ~2014). In this section I will briefly discuss some of the traditional methods that have been used in the past to tackle the challenges associated with object detection. After we have a solid understanding of the history and fundamentals of the space, we will move on to some of the cutting-edge techniques.

2.1.1 Classic Methods

One of the first major milestones in the development of object detection algorithms was the introduction of the Viola-Jones detectors, by Paul Viola and Michael Jones, in 2001 [4, 5]. This model was primarily designed to detect human faces and depended on the use of a sliding window technique to go through all possible locations and scales in an image. The detector introduced three novel contributions: “integral image”, “feature selection” and “detection cascades” to help to drastically improve on the accuracy and speeds seen in previous object detection models. The VJ detector managed to achieve speeds 2 to 3 orders of magnitude better than other algorithms of its time under comparable detection accuracy. The Viola-Jones detector used Harr-like features [4], which are simple rectangular features that measure the difference between the sum of the pixels in adjacent rectangular regions.

Viola and Jones released a research paper in 2001, focusing specifically on the efficacy of their detection algorithm for pedestrian detection [6]. The algorithm combined motion and appearance information to build a robust model of walking humans, with a low false positive rate. The biggest strength of the Viola Jones model was its extremely low computation time. According to their paper, the model took about 0.25 seconds to detect all pedestrians in a 360 x 240 pixel image on a 2.8 GHz P4 processor. These numbers were a massive improvement over the existing approaches at the time. The algorithm was even effective in poor weather conditions, as you can see in the lower right image of the below figure.

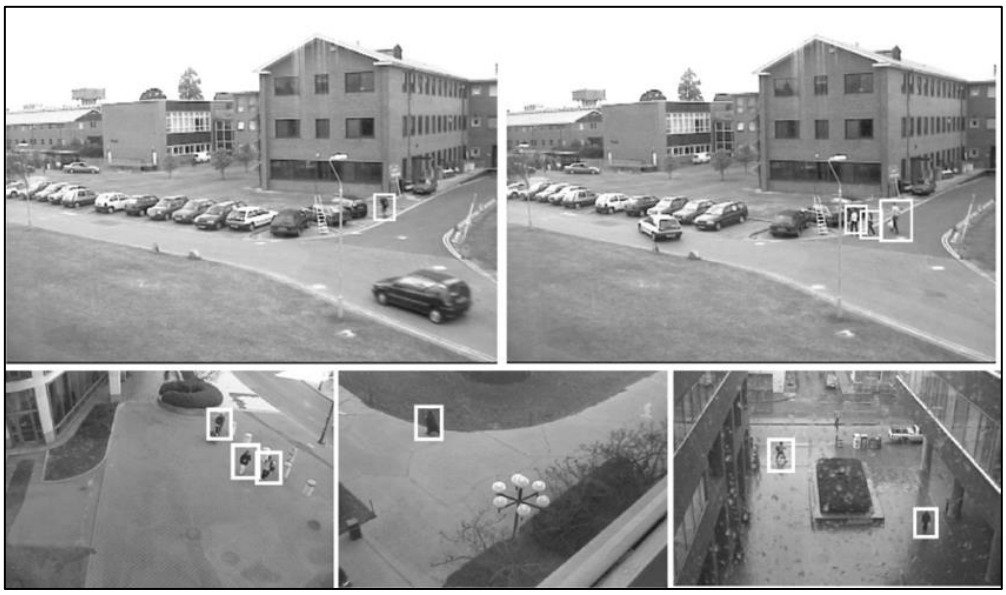


Figure 2: Example detections from Viola Jones dynamic pedestrian detector [6]

The next breakthrough in the field of object detection was the introduction of Histogram of Oriented Gradients (HOG) feature descriptors, by N Dalal and B Triggs in 2005 [7]. Serving as a replacement for the Haar-like features described above, HOG improved the performance of previous object detection algorithms such as the VJ detector by using a more robust and discriminative feature representation. Haar-like features are not very effective in capturing the structural information of an object and are sensitive to variations in lighting and pose. In contrast, HOG features represent the distribution of edge orientations and magnitudes in an image. They are more robust to variations in lighting and pose and capture more structural information about an object. Specifically, HOG features extract local gradient information from an image by computing gradient orientation histograms over small image regions called cells. These histograms are then concatenated into larger blocks to capture more global structure. The resulting feature vectors are then used to train an object detection model, such as a support vector machine (SVM).

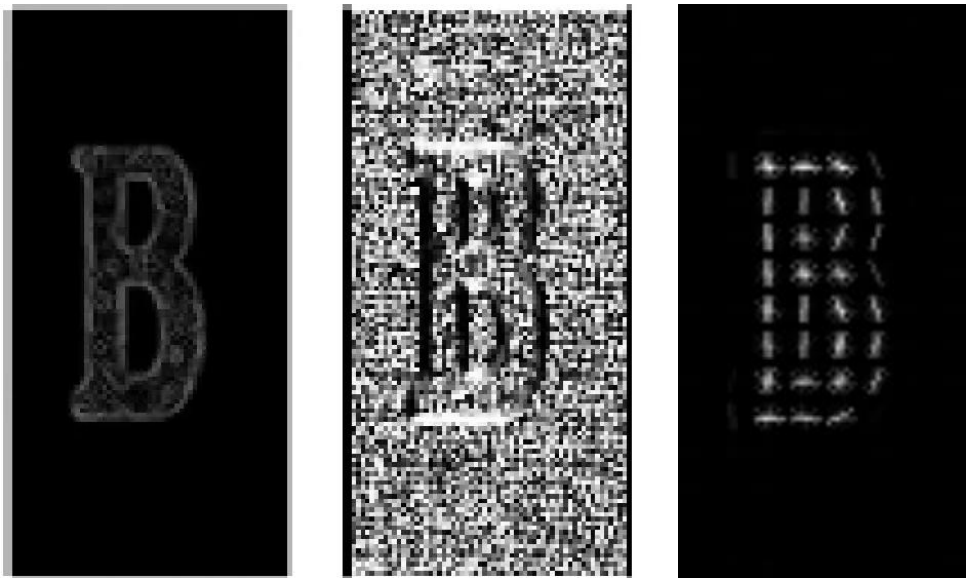


Figure 3: Visualisations of gradient magnitude (left), gradient angle (middle) and HOG features (right) of same image [8]

HOG was originally designed with the challenge of pedestrian detection in mind and the work of Dalal and Triggs was further refined to improve the performance of HOG-based SVM pedestrian detection models [9], up until the advent of deep neural networks and CNNs, which I will cover in the next section.

2.1.2 Two-Stage Methods

While these SVM models boasted impressive accuracy numbers, especially for smaller datasets, a major downside of them is that the features need to be hard coded into the model and could not be learned in the same way that we see with modern neural networks. As the performance of these hand-crafted models became saturated, the research of object detection reached a plateau after 2010. However, this changed in 2012 when the world saw the rebirth of convolutional neural networks, when A Krizhevsky et al. used deep CNNs to develop an image classifier for the ImageNet dataset, which massively outperformed the previous state-of-the-art [10]. While this model, known as AlexNet, was a massive jump in the object detection space, it did not perform the other main task involved in object detection: localisation. That is the ability to localise an object of interest in an image and output bounding box co-ordinates around it.

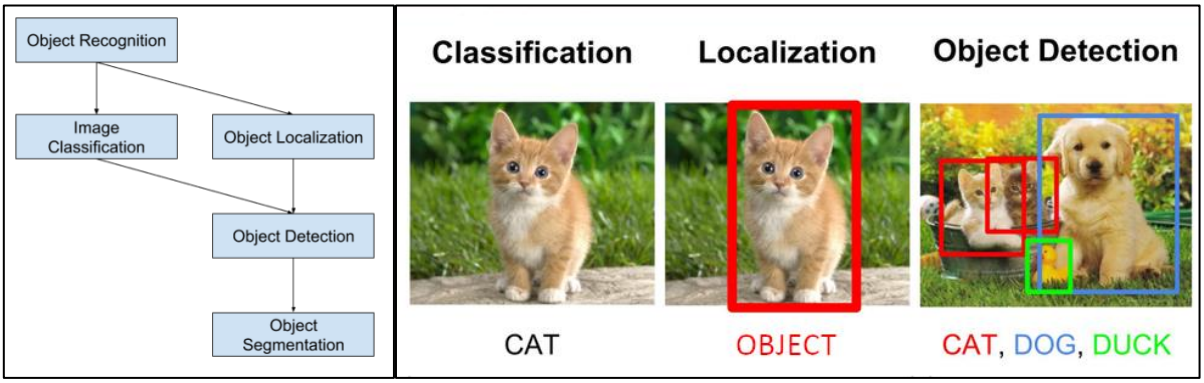


Figure 4: Object detection stage breakdown [11, 12]

It was two years after Krizhevsky's AlexNet breakthrough, in 2014, when Girshick et al. proposed a new approach which would start a full-on revolution in the realm of object detection: Regions with CNN features (R-CNN) [13]. The idea behind R-CNN is simple: it starts by extracting a set of object proposals (object candidate bounding boxes), by selective search. Each of these object proposals are then cropped and resized to a fixed resolution, before being fed into a pretrained CNN-based image classification model, such as AlexNet, to extract features from the image. Finally, linear SVM classifiers are used to predict the presence of an object within each region and to assign an image class, along with a confidence level, to the region.

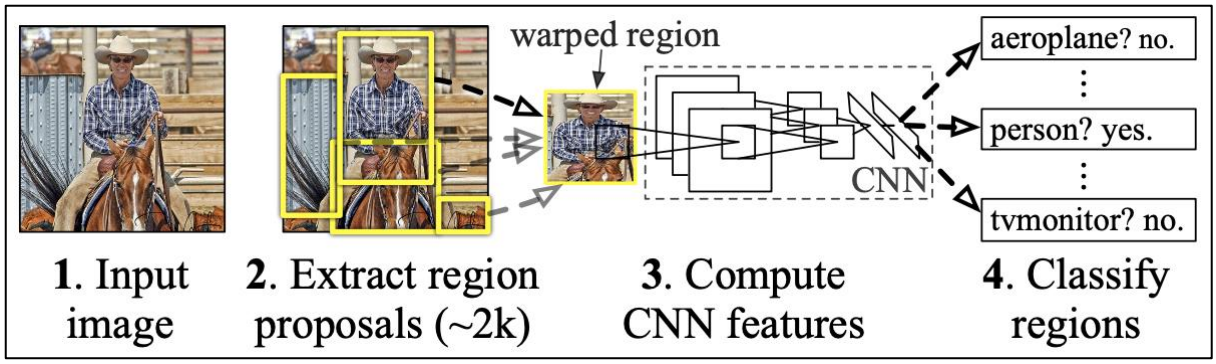


Figure 5: R-CNN object detection pipeline [13]

Krizhevsky's 2014 R-CNN proposal was undeniably a massive jump in object detection performance. Using the VOC07 dataset as a benchmark, R-CNN achieved a massive improvement over the HOG-based DPM-v5, which was state-of-the-art at the time [14]. While DPM-v5 had a mean Average Precision (mAP) of 33.7%, R-CNN achieved a staggering 58.5% [15].

However, R-CNN was not without its downsides. R-CNN performs lots of redundant feature computations on a large number of overlapped proposal object bounding boxes (over 2000 object candidates per image). This led to extremely slow detection speeds (14s per image with a GPU). This made it completely incompatible with real-time video object detection, such as for pedestrian detection for a car's on-board camera. For comparison, the HOG-based DPM-v5 object detector is capable of detection speeds as low as 33ms per image, enabling real-time object detection from a video feed at up to 30 frames per second. Additionally, because the training pipeline is several stages long (feature extraction and detection networks trained separately) , training is expensive in space and time. When features are extracted from each of the object proposal images, they must be written to a disk before they can be classified by the SVM object detectors. With very deep feature extraction networks, such as VGG16, this process takes 2.5 GPU-days for the 5000 images of the VOC07 training and validation sets. These features then require hundreds of gigabytes of storage.

Later in 2014, K He at al. proposed Spatial Pyramid Pooling Networks (SPPNet) [16]. Before this, all CNNs required a fixed-size input, e.g., 224x224 image for AlexNet [10]. However, with the advent of He's Special Pyramid Pooling layer, CNNs could generate a fixed-length representation of an image or region, regardless of dimensions, and did not require rescaling. This enabled feature maps to be computed just once for each image, and not over 2000 times

as we saw in R-CNN, where a feature map needed to be computed for each individual object proposal region. Then, fixed-length representations can be extracted for each region from this single feature map. SPPNet drastically outperformed R-CNN in speed and even detection accuracy. On the VOC07 dataset, SPPNet achieved an mAP of 59.2%, just beating out R-CNN's 58.5%, while achieving speeds over 20 times faster. While SPPNet was still not capable of real-time video object detection, this was a massive step in the right direction. However, SPPNet still had its drawbacks: it still had a multi-stage training pipeline, which led to very slow training. Also, SPPNet only fine-tunes the fully connected object detection layers during training, and not the pervious convolutional layers. This limits the potential for improvement of the convolutional side of the network.

A year after his initial R-CNN paper was published, Ross Girshick helped to introduce both Fast R-CNN, and later Faster R-CNN, both in 2015 [17, 18]. Fast R-CNN integrated many of the best features of R-CNN and SPPNet. It enables simultaneous training of the detector and bounding box regressor under the same network configurations. Not only did this drastically improve training and inference speeds, because the convolutional network could now be fine-tuned during training, accuracy also saw significant improvements. On the VOC07 dataset, Fast R-CNN increased the mAP from 58.5% (R-CNN) to 70%, while achieving detection speeds over 200 times faster than R-CNN.

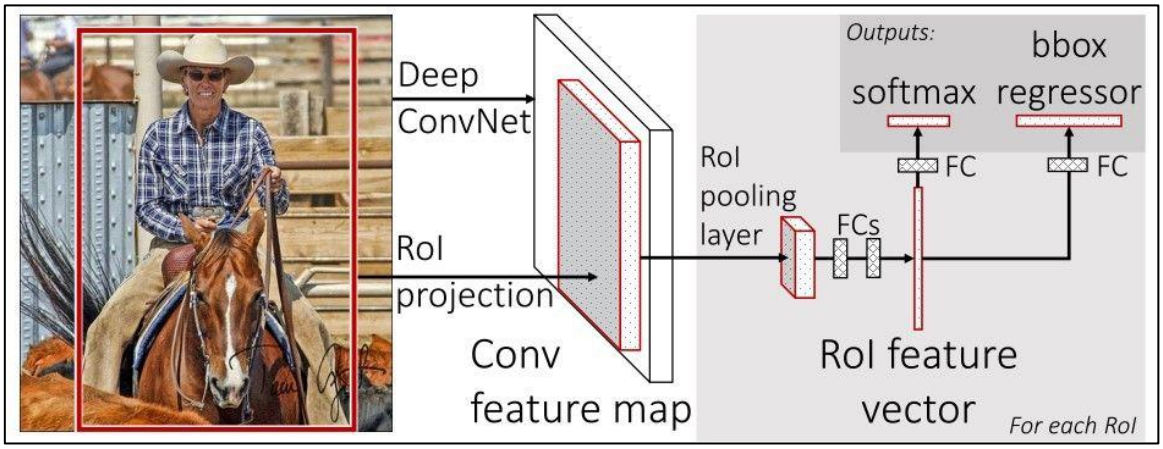


Figure 6: Fast R-CNN improved pipeline [19]

Faster R-CNN, which was led by S. Ren and received contribution from Girshick, [18]. The main contribution of R-CNN is the Region Proposal Network (RPN), which massively reduced the computational cost of region proposals. Again, Faster R-CNN saw significant improvements

over its predecessors in both accuracy and inference times. It was with the introduction of Faster R-CNN when the prospect of real-time video object detection, which is necessary for pedestrian detection for a car's on-board camera, became very realistic in the near future. Faster R-CNN could effectively detect objects in a 5 frame per second stream of video.

One problem that existed even with the introduction of Faster R-CNN in 2015, was that the existing models struggled to detect objects at different scales since the convolutional layer can only extract features of a fixed scale. For example, if an R-CNN model was trained on a dataset of objects which are around ~200 pixels tall, it might struggle to detect the same objects if they are 50 pixels tall. This problem extends to the application of pedestrian detection. For example, a model might be effective at detecting pedestrians which are 20 metres away, but struggle with pedestrians 40 metres away, unless these were also specifically included in the dataset. Because of this shortcoming of existing object detection models, T-Y Lin et al. proposed the Feature Pyramid Network (FPN) in 2017 [20], and J Hu et al. released FPN++ two years later [21]. These new networks have a top-down architecture with lateral connections for building high-level semantics at all scales. This helped to further increase the accuracy of existing networks and became a basic building block for many of the latest object detectors.

Several more incremental improvements were made on two-stage object detection models in the following years. Mask R-CNN, which was released in 2017 introduced object segmentation, which is a step further than object detection, where the individual pixels which contain an object are determined, as opposed to just a bounding box [22]. This greater precision of localisation is an important factor for the application of pedestrian detection, but training and inference speeds associated with Mask R-CNN are even slower than the already existing two stage detectors due to its higher complexity, so it is not suited to real-time pedestrian detection in its current form.

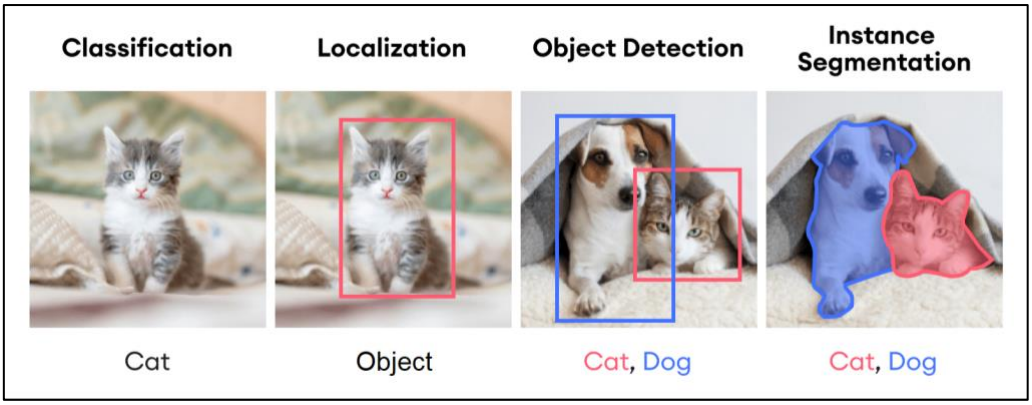


Figure 7: Object instance segmentation visualised [23]

Additionally, newer models such as Granulated R-CNN [24] and Cascade R-CNN [25], which introduced the multi-stage cascaded architecture, were developed to further build on the work of previous two-stage detection models. However, while these incremental improvements boosted detection accuracy, the inference speeds plateaued around 100-200 ms/frame, i.e., 5-10 frames per second for real-time video object detection. These inference speeds are simply not sufficient for the application of pedestrian detection for a car's on-board camera. Additionally, the computational complexity involved in two-sage detection is very significant, which means that cars would need very high-end GPUs, which add cost and power consumption. However, as we will see in the next section, there is an alternative to the traditional two-stage object detection architecture, which aims to improve on these shortcomings.

2.1.3 Single-Stage Methods

The history of modern single-stage object detection algorithms began in earnest in 2013 with Sermanet et al.'s introduction of OverFeat [26], a multi-scale sliding window approach, which detected objects in an image and used convolutional neural networks (CNNs) to predict class probability and bounding box co-ordinates. Contrary to the two-stage approaches which were being released at the time, OverFeat achieved all of localisation, classification and detection within a single integrated neural network, which allowed the various tasks to be learned simultaneously.

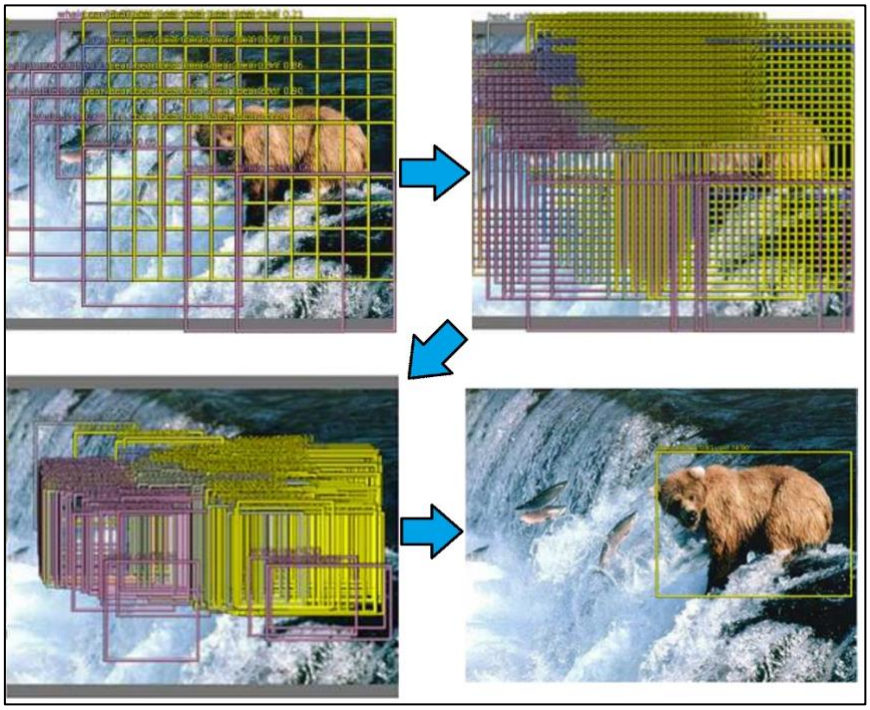


Figure 8: OverFeat's coarse-to-fine classification approach [26]

In the context of real-time pedestrian detection, OverFeat's coarse-to-fine approach allowed for reasonable performance which was competitive with its contemporary two-stage approaches, achieving a Mean Average Precision (mAP) of 24.3% on the PASCAL VOC 2012 dataset. However, the algorithm struggled with objects with a wide range of scales and aspect ratios. The reliance on predefined sliding windows also meant that the algorithm was less flexible and adaptive to different environments. While OverFeat had its downsides, it showcased the massive potential of one-stage object detectors and paved the way for more research into the future.

The You Only Look Once (YOLO) algorithm, proposed by Redmon et al. in 2015 [27], was a truly revolutionary development in the field of real-time object detection. YOLO framed object detection as a regression problem, predicting object bounding box co-ordinates and class probabilities in a single pass through the network. As is covered in-depth in section 3.1.2, YOLO's grid-based approach divided the input image into cells, which were individually responsible for detecting objects whose centre-point was in their cell. Where YOLO excelled over its contemporary competition was in its inference speed. On the PASCAL VOC 2007 dataset, YOLO achieved a respectable mAP of 63.4% at an astounding 45fps, which was a monumental improvement over the two-stage models released in the same year which achieved comparable accuracy scores at a fraction of YOLO's inference speed: Fast RCNN (66.9%; 0.5fps) [17] and Faster R-CNN (69.9%; 7fps) [1, 18].

Algorithm	mAP (%)	Inference Speed (fps)	Algorithm Type
YOLOv1	63.4	45	One-stage
Fast R-CNN	66.9	0.5	Two-stage
Faster R-CNN	73.2	7	Two-stage
SSD	74.3	59	One-stage

Table 1: Object detector performances on PASCAL VOC 2007 dataset (Titan X GPU) [1]

However, YOLO did struggle with low detection accuracy for smaller objects and high localisation errors due to its grid-based design. These drawbacks made it ill-suited for real-world application of pedestrian detection, though these problems would be addressed in the many iterations of the YOLO algorithm which would come in the following years. YOLO certainly spurred a paradigm shift in the field of object detection and drew great interest to single-stage algorithms, which have been the focus of intense research in the years since.

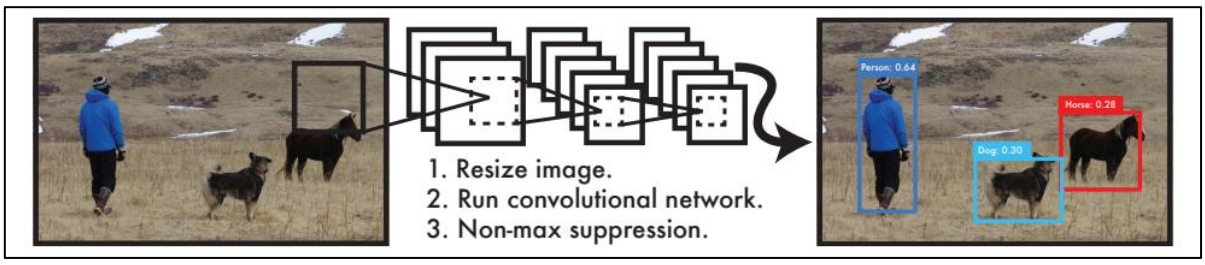


Figure 9: YOLO's refreshingly simple pipeline [27]

Later in 2015, Liu et al. introduced the Single Shot Detector (SSD) [28], which improved upon the limitations of YOLO by incorporating anchor boxes and multiple feature maps to improve detection of objects of varying sizes and aspect ratios. SSD used a series of convolutional layers with varying resolutions to predict bounding boxes and class probabilities simultaneously and detect objects of different scales on different layers of the network, where previous models had only run detection on their top layers. These changes further simplified end-to-end optimisation of the entire model, further improving the speed vs accuracy trade-off that existed in object detection. These improvements helped to propel SSD well ahead of the state-of-the-art in terms of inference speed and accuracy, as you can see from the table above. While SSD still did not handle small objects as well as two-stage approaches, it showed that vast improvements in this aspect were possible, which has been further proven in the years since.

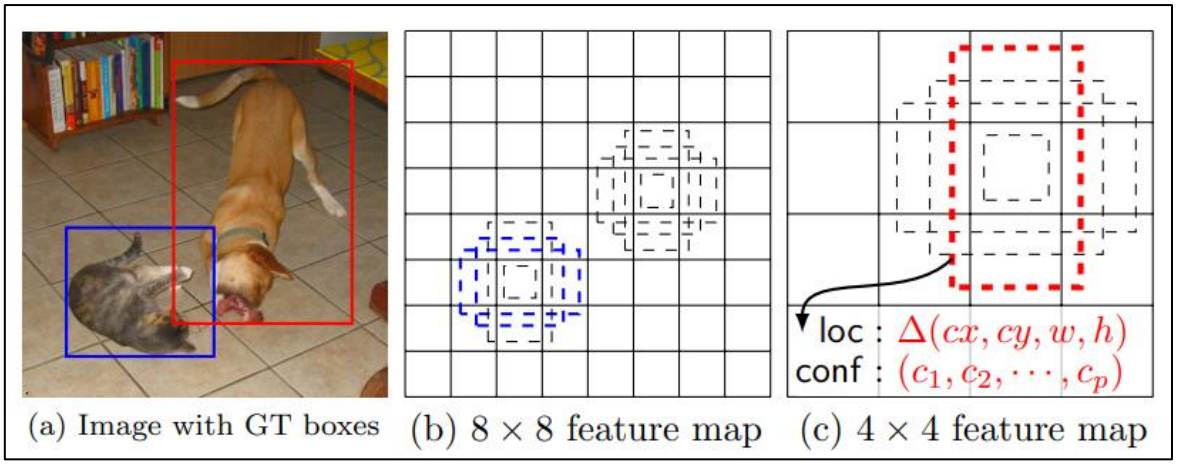


Figure 10: SSD framework: bounding box variation by location, scale and aspect ratio [28]

There have been several incremental improvements in the field of one-stage object detection in the years since the release of YOLO and SSD [29-31]. In 2017, RetinaNet's [32] Focal Loss function focused the training process on hard-to-classify examples, improving the model's ability to detect small and occluded objects, which is essential for real-world pedestrian detection, where a person might be partially hidden behind the bonnet of a car or a signpost, for example. These improvements helped RetinaNet achieve the highest accuracy/speed combination among its contemporaries on the MS COCO test-dev dataset, achieving an average precision of 39.1% at 5fps. This was a big improvement in accuracy, but RetinaNet's inference speeds were still not up to the standard required for real-time pedestrian detection.

In 2022, Wang et al. set a new state-of-the-art for real time object detectors with their proposal for YOLOv7 [33], the latest iteration of You Only Look Once. YOLOv7 surpassed all known object detectors in both speed and accuracy in the range from 5 FPS to 160FPS. Using novel techniques such as extended efficient layer aggregation and re-parameterization planning, YOLOv7 achieved best-in-class performance among all real-time detectors, which makes it an easy choice for my pedestrian detection model. As you can see from the figure below, where YOLOv7 is represented by the red dots, no other real-time object detection algorithms compete with YOLOv7 in terms of its accuracy and inference speed combination.

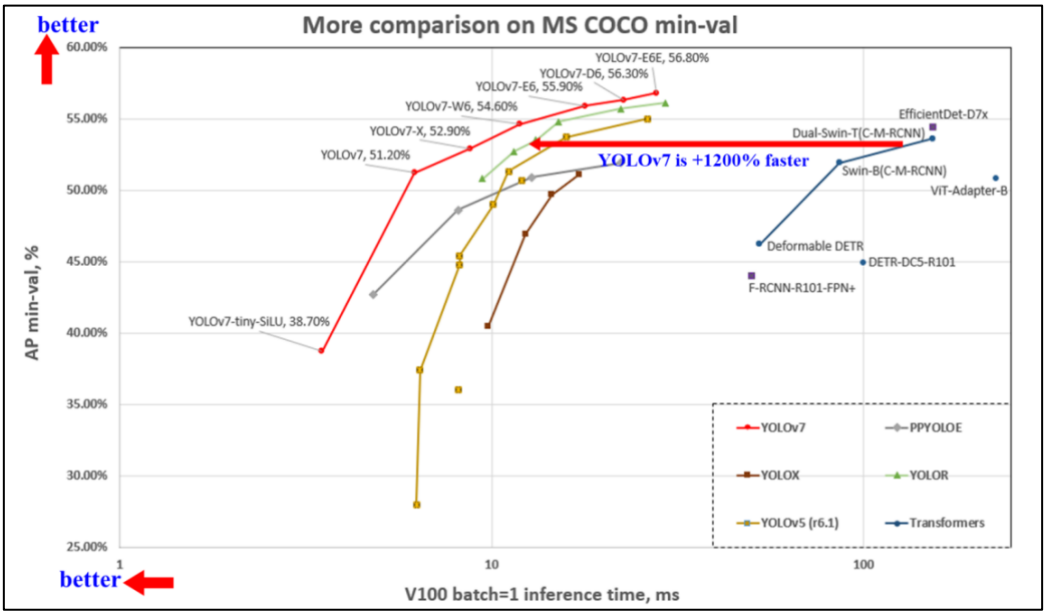


Figure 11: YOLOv7 vs other real-time detectors on MS COCO dataset [33]

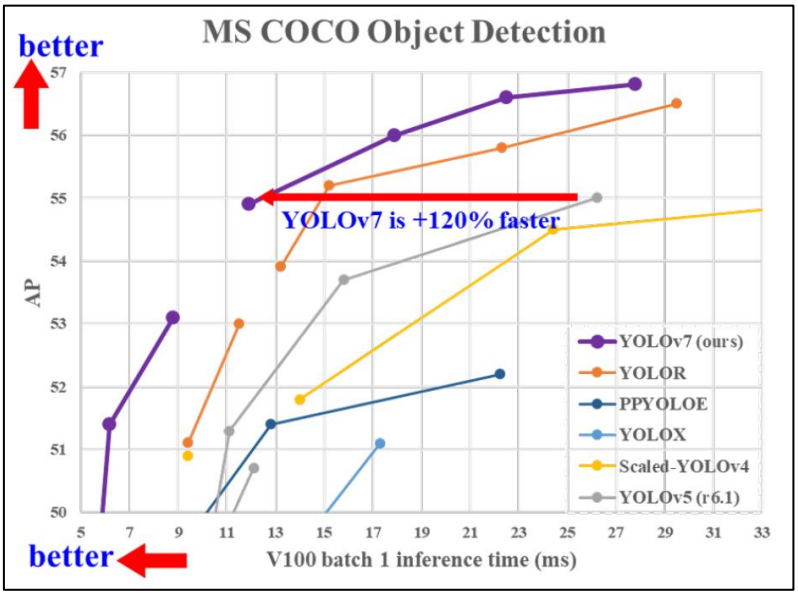


Figure 12: YOLOv7 vs other single-stage detectors on MS COCO dataset [33]

2.2 Pedestrian Trajectory Prediction

2.2.1 Classic Methods

Object trajectory prediction is a task that challenged engineers for many decades. One of the notable early techniques that was introduced to address this problem was the Kalman filter, which predates some of the modern neural network-based approaches by several decades. This technique was first described and developed in various papers by Rudolf Kálmán in the early 1960s [34, 35] and was used by NASA for trajectory analysis of rockets in the following years [36]. The Kalman filter enables us to predict the value of a certain unknown variable, based on measurements of known variables. For example, the Kalman filter enabled NASA to accurately estimate the position of aircraft using noisy data such as on-board inertial sensor measurement and ground-tracking data.

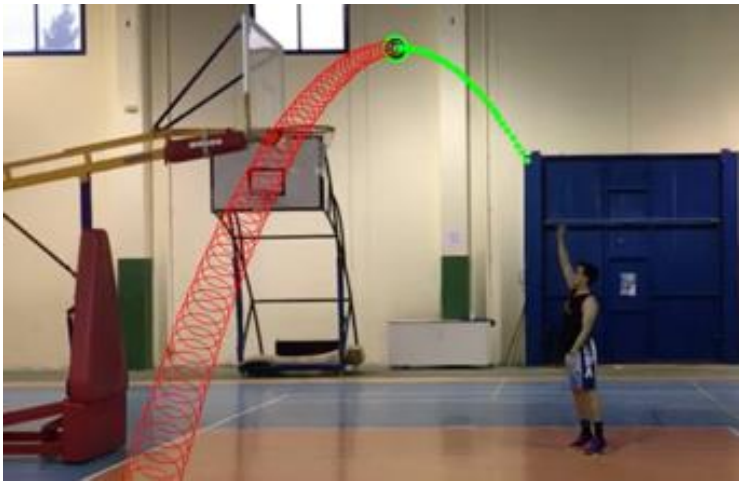


Figure 13: Basketball trajectory tracking using Kalman filter [37]

Similarly, the Kalman filter can be used to predict the future trajectory of an object using measurements such as previous position and velocity, even if these measurements are noisy [38]. However, one of the downsides of the Kalman filter is that it assumes that the system dynamics are linear and that the measurement noise is Gaussian. In practice, this means that the Kalman filter might do quite well at predicting the trajectory of a projectile fired from a cannon but will struggle to model the unpredictable movements of a pedestrian. In recent years, however, there have been efforts to combine the Kalman filter with modern neural network-based approaches to tackle trajectory prediction problems more effectively, such as KalmanNet, introduced by G Revach et al. [39, 40].

Additionally, there are various knowledge-based approaches, which have been used in the past to better understand and predict the movements of pedestrians. One such approach is the Social Force Model (SFM), which was introduced in 1998 by D Helbing and P Molnár [41], which aimed to gain a mathematical understanding of the social forces which affect how a pedestrian move, especially around other pedestrians. According to the SFM, some of the social forces that act on pedestrians include the desire to reach their destination, the need to avoid collisions with other pedestrians and obstacles and the desire to maintain personal space between others to avoid social discomfort. While these social forces might sound obvious, the ability to model them mathematically is crucial for training a computer to understand the movements of pedestrians. Additionally, a simple knowledge-based velocity model [42], which assumes that pedestrians move at a constant speed in constant direction, actually proves to be a decent pedestrian trajectory predictor, even outperforming some of the modern neural network-based approaches.

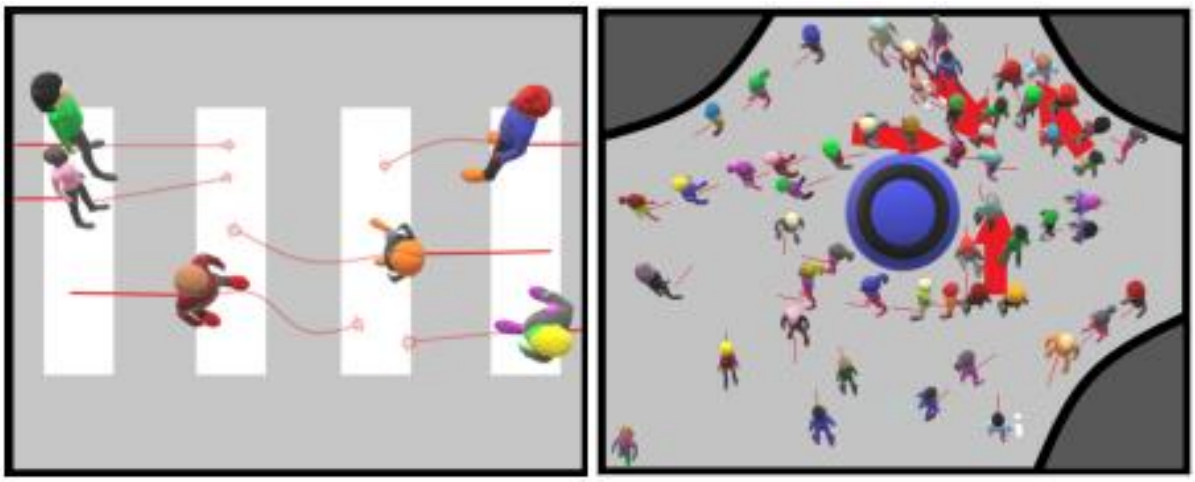


Figure 14: Scenes where social forces impact pedestrian trajectory [43]

Additionally, in 2015 B Volz et al. released a paper [44] which outlined a new model for estimated feature relevance in learning pedestrian behaviour at pedestrian crossings. This study used statistical methods, not neural networks, to determine which of a list of features including distance to curb, orientation and speed had the most significant impact in determining whether the pedestrian would cross the road. While these knowledge-based approaches are certainly effective for gaining an understanding of pedestrian behaviour, they are best used in combination with deep learning approaches to yield the best results in pedestrian trajectory prediction [43].

2.2.2 Long Short-Term Memory

The problem of pedestrian trajectory prediction, where we want to predict the future location of a pedestrian based on past information, can be classified as a sequence-to-sequence problem [45]. While this is covered at length in section 3, in summary, my model will take a sequence of frames of video and pedestrian locations as input and return a sequence of predictions for future pedestrian locations as output. For this reason, we need an encoder-decoder architecture [46]. The encoder block will take the sequence of video frames and bounding boxes and compress the sequence down into a single fixed-length vector. This vector is then fed into the decoder, which outputs its sequence of predictions. This is the basis upon which all sequence-to-sequence deep-learning models are based, but the network which is contained inside the encoder and decoder blocks can vary. Up until quite recently, the encoder and decoders tended to be some variation of a recurrent neural network.

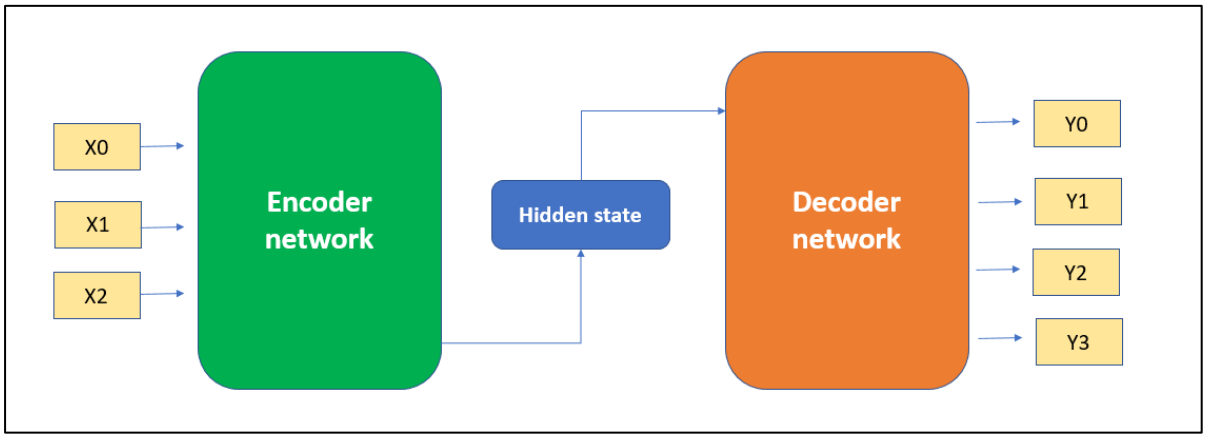


Figure 15: Encoder-decoder architecture [47]

Traditional recurrent neural networks (RNNs) were developed in the late 1980s [48-50], and were intended to address the problem of temporal dependencies in sequential data. Traditional feedforward neural networks, which were popular at the time, were not well-suited to handling sequences of data where the order and context of the elements was important. The RNN introduced a feedback loop, which enabled the network to learn patterns and dependencies within sequences, such as those found in time series data or natural language. However, these traditional RNNs suffered from the vanishing & exploding gradient problem (discussed at length in section 3), which made it ineffective at learning long-term dependencies in data [51-53]. For example, a natural language model might be able to use

dependencies from a few words previous but might struggle to use dependencies from several sentences previous.

It was in direct response to the vanishing & exploding gradient problem that Hochreiter & Schmidhuber introduced the long short-term memory network in 1997 [54]. This network used a series of gates to control the information which was stored in long-term memory (see section 3), thereby greatly increasing the network's ability to learn long-term dependencies. Additionally, the work of F Gers and F Cummins in 2000 [55] was pivotal in improving the performance of the LSTM, by introducing the forget gate. This gate controls what information is discarded by the LSTM block, as it makes way for new information to be stored in long-term memory. For example, in a natural language model, the forget gate might forget the gender of a certain subject when it encounters a new subject in the text. The impact of this research has seen a revival in recent years, growing steadily each year since 2015 [56].

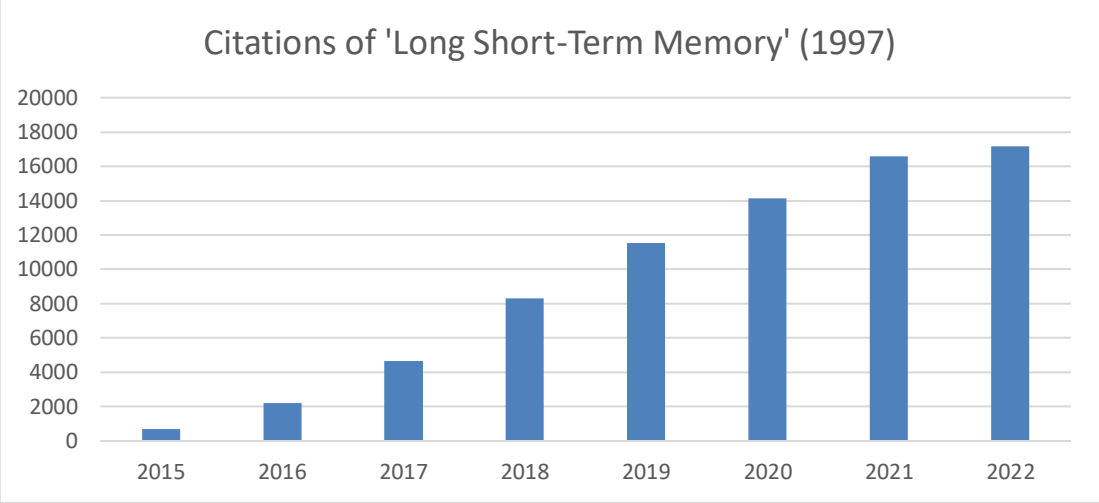


Figure 16: Citations of Hochreiter's LSTM paper from 2015 to 2022 [56]

A breakthrough in the pedestrian trajectory prediction space occurred in 2015, when X Shi et al. introduce the convolutional LSTM (ConvLSTM) network [57]. Before this, LSTMs took sequences of 1D vectors as input and could learn temporal dependencies within data. With the advent of ConvLSTM, they could take multidimensional tensors as input, and learn spatiotemporal dependencies. By combining this with a pretrained feature extractor, such as VGG16, we can identify patterns in space and time in a sequential video feed. This is a massive development in the pedestrian trajectory prediction space, as we can now use cues such as changes in pedestrian pose, gaze & body language to predict their future position. While the

original paper focused on the application of rainfall forecasting, there have since been studies which applied the same architecture to pedestrian detection, with promising results [58, 59]. In 2021, Tommy Browne [60], UL 2021 graduate in MsAI, proposed using a pre-trained VGG16 to extract features from images of pedestrians, cropped to an enlarged version of their bounding box, and inputting this feature map into a ConvLSTM to capture spatiotemporal patterns.

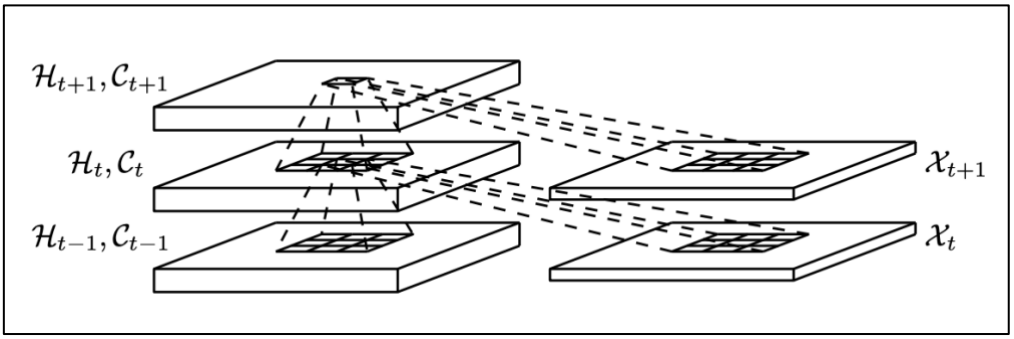


Figure 17: Inner structure of ConvLSTM [61]

While these LSTM-based pedestrian prediction models were effective at making predictions on a single pedestrian in isolation, they could not effectively account for the social forces which undoubtedly have a significant impact on pedestrian trajectory. This shortcoming of existing LSTM networks is what Alahi et al. aimed to address with their introduction of the Social-LSTM in 2016 [62]. This model jointly predicted the paths of all people in a scene by considering the knowledge-based rules and social conventions that humans follow as they navigate crowded environments. This model built on the ideas proposed by Helbing and Molnar in their Social Force Model paper, which we discussed earlier [41]. This model outperformed the state-of-the-art pedestrian prediction models on many public datasets and opened the door for further research into this area. Unfortunately, this paper focused on pedestrian detection from a fixed camera with a bird's-eye view of the scene, so Alahi's algorithm does not directly translate to my application, where the camera is at the height of a vehicle and moves around with the car. However, this research provides a good proof of concept for the potential to model social forces on pedestrian trajectory and shows that it could be possible to apply the same logic to an autonomous vehicle application.

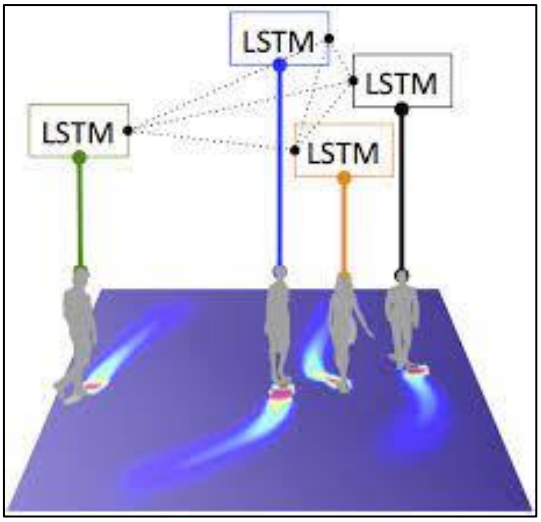


Figure 18: Predicted distribution of future trajectories as heat-map [62]

In recent years, incorporating scene context into LSTM-based pedestrian trajectory prediction models has also garnered significant attention. For example, if there is a physical obstacle such as a wall or a road, an effective trajectory prediction model should take this scene context into account before making a prediction on a pedestrian's future position. It was with this in mind that Manh and Alaghband proposed the Scene-LSTM in 2018 [63]. This model superimposes a two-level grid structure onto a scene, for a coarse and fine spatial granularity. It uses these to learn the contextual information from the scene, such as where physical obstacles are, and thereby the most common path for a pedestrian to take in a scene. Again, this research focuses on a fixed camera with a bird's-eye view. However, it proves that it is important for a trajectory prediction model to be aware of scene context when creating an effective pedestrian trajectory prediction model.

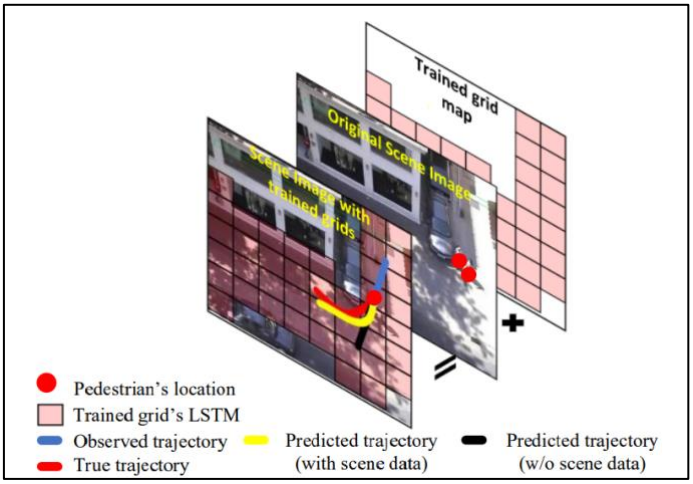


Figure 19: Scene compliant pedestrian predictions using Scene-LSTM [63]

Another major development in the field of pedestrian trajectory prediction and sequence-to-sequence models more generally was the introduction of attention mechanisms, which were introduced by Bahdanau et al. in 2016 [64]. Before their introduction, encoder-decoder architecture models would only allow the encoder to pass a fixed-length vector to the decoder to represent the input sequence. Bahdanau's paper conjectured that this was a bottleneck in improving the performance of these models, and proposed to extend this by allowing a model to automatically soft-search the input sequence for specific information which would inform the current prediction, without having to form these parts as hard segments explicitly. Because the original paper focused on natural language processing, this meant that the model could search specific words in the source sentence that are relevant to predicting a target word. For example, it could look back over input text to determine the gender of a given subject.

In the following years, the attention mechanism approach was applied to time-series prediction problems such as pedestrian trajectory prediction. In 2017, Sadeghian et al. built on the Social-LSTM and attention mechanisms released in the previous year to introduce a novel approach to pedestrian tracking [65]. This model leveraged a soft attention mechanism to learn and track multiple cues with long-term dependencies, which significantly improved performance. By dynamically weighing the importance of these cues (appearance, motion, gaze, pose, scene context), the model could adapt to challenging tracking scenarios, such as occlusions, camera motion and changing appearance of pedestrians, and outperformed the state-of-the-art models on many of the benchmark datasets. Also, in 2018, Vemula et al. released Social Attention, which applied the attention mechanism specifically to social interactions between pedestrians, to gain a better understanding of which surrounding agents humans attend to, when navigating in a crowd [66].

2.2.3 Generative Adversarial Networks

Another interesting type of network which has been used for the task of object trajectory prediction is the generative adversarial network (GAN), which was introduced by Goodfellow et al. in 2014 [67]. GANs are a type of unsupervised learning network, which is capable of generation of novel data and realistic simulations. The GAN is made up of two sub-models: a generator and a discriminator. Put simply, the job of the generator is to generate fake sample inputs, and the job of the discriminator is to determine whether a sample is "real", or a synthetic sample created by the generator. This dynamic can be thought of as a two-player adversarial game, where the "loser" of each round has their weights and biases adjusted. For example, if a GAN is being trained to write poetry, the discriminator will receive a sample poem and must determine if it is a "real" poem, or if it was written by the generator. If it guesses correctly, the generator has its weights and biases adjusted and if it guesses incorrectly, the discriminator does. This continues until the generator eventually produces sample inputs which cannot be reliably distinguished as "fake" by the discriminator, or sometimes even by humans.

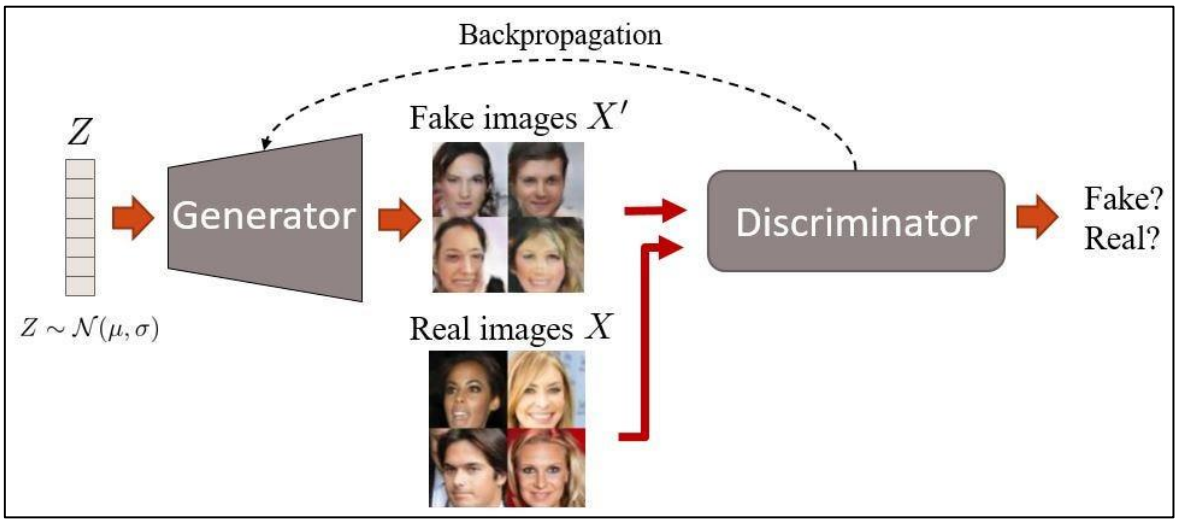


Figure 20: Visualisation of a GAN which aims to generate images of human faces

GANs are capable of many tasks other than image or text generation, including video frame prediction. Using a GAN in conjunction with a sequential network like an LSTM, we can provide the network with a sequence of video frames and use it to predict what the next frame in this sequence will look like. Using this principle, it's easy to see how a GAN can be applied to our task of pedestrian trajectory prediction. One of the earliest and most influential

works which applied an LSTM-GAN network to predict pedestrian trajectory was the Social-GAN, introduced by Gupta et al. in 2018 [68]. Using an LSTM encoder-decoder network to generate predicted trajectories for pedestrians, and then feeding these into a discriminator to distinguish between synthetic and genuine samples, the Social-GAN was able to predict pedestrian trajectories with greater accuracy than its predecessor, the Social-LSTM. According to their evaluation, Social-GAN outperforms Social-LSTM in terms of prediction accuracy for certain datasets. In particular, Social-GAN demonstrates improvements in average displacement error (ADE) and final displacement error (FDE) metrics. This improvement can be attributed to the generative nature of GANs, which allows the model to produce diverse and plausible future trajectories.

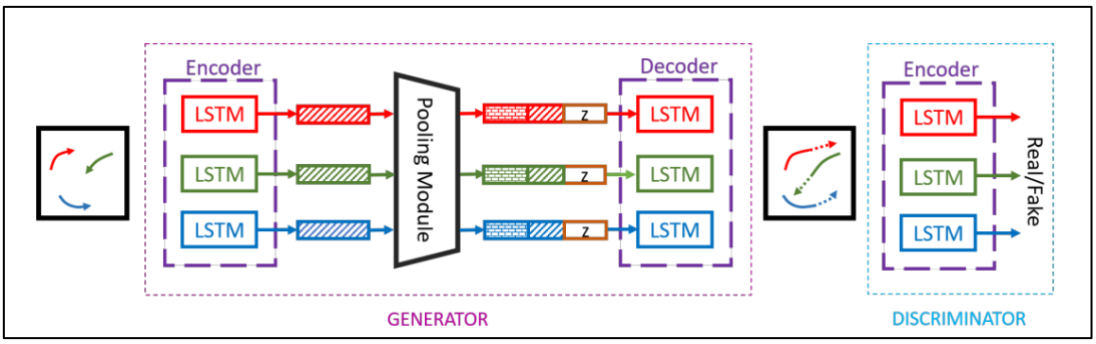


Figure 21: System overview of Social-GAN [68]

Another major advantage of GANs over alternative trajectory prediction approaches is its ability to handle uncertainty by generating multimodal predictions. In the context of pedestrian trajectory prediction, the GAN can generate multiple plausible future trajectories for each pedestrian in a given scene, along with estimated probabilities for each trajectory [69]. This was explored more deeply by Sadeghian et al. who introduced SoPhie, an attentive GAN which is compliant to both social and physical constraints [70]. In contrast, many traditional trajectory prediction approaches, such as a basic LSTM, often produce single deterministic predictions, which may not adequately capture the full range of paths that the pedestrian could take. This limitation of basic LSTMs can lead to suboptimal or even unsafe predictions, especially in applications such as autonomous driving.

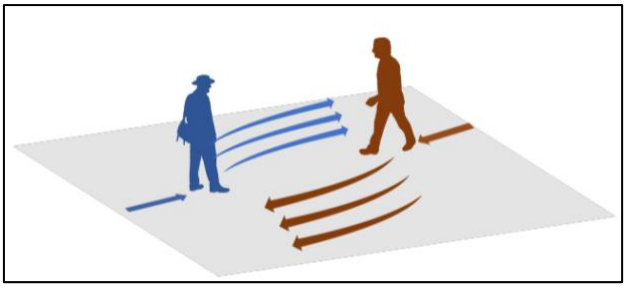


Figure 22: Many possible paths for each pedestrian to take [68]

This probabilistic functionality is crucial when dealing with the inherent uncertainty of human motion. For example, when a car is approaching a group of young children who are playing with a ball at the side of a road, a good prediction model needs to be aware of the small probability that a child might run onto the road after the ball, and slightly reduce the cars speed accordingly. While this might not be the most likely path for the children to take, it is a possibility that the pedestrian predictor should consider. This probability calculation is something that is constantly happening in the mind of a human driver and it is important that a good pedestrian trajectory predictor should have the same capabilities.

2.2.4 Transformers

One of the most exciting developments in recent years of artificial intelligence research has been the introduction of the transformer network by Vaswani et al. with their ground-breaking paper “Attention is All You Need” (2017) [71]. Like LSTMs and other RNNs which we discussed earlier, transformers are well suited to sequence-to-sequence (seq2seq) tasks, as they were developed for the field of natural language processing. Transformers make up the backbone of many of the language models that we have seen burst into the collective social consciousness in the past year, such as GPT-4 (the model behind ChatGPT) [72, 73], as well as Google’s BERT [74].

One key difference between the transformer and recurrent neural networks, such as LSTMs, is that the transformer does not rely on sequential processing on input data. Instead, it leverages self-attention to model dependencies and relationships between elements in the input and output sequences. This enables transformers to massively outperform LSTM-based models in terms of training times, because of their ability to leverage parallel GPU processing power. In order to capture temporal dependencies in the data, transformers instead use

positional encoding, whereby each element in the models input, such as a word in a language model, has a fixed length vector containing positional information.

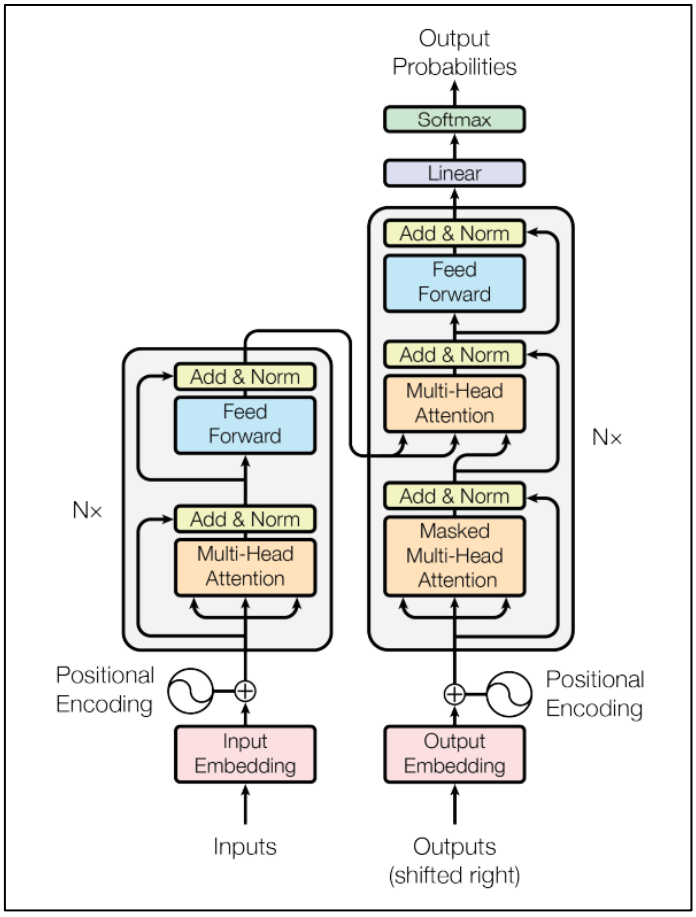


Figure 23: Transformer model architecture [71]

The multi-head attention mechanism within transformers, seen in the figure above, allows the model to simultaneously focus on different aspects of the input sequence by computing multiple self-attention representations in parallel. This improves the model's ability to capture various relationships and dependencies between input elements, enabling transformers to learn even longer-range dependencies than LSTMs, which were designed specifically for this purpose. Transformers have also popularised the paradigm of pre-training and fine-tuning in seq2seq tasks. Models like BERT and GPT are pre-trained on massive amounts of data using unsupervised objectives and can then be fine-tuned on specific seq2seq tasks with relatively small amounts of labelled data, often resulting in state-of-the-art performance, without retraining a full model at massive computational expense.

Because transformers are still relatively new and were originally intended to be used for language processing, there has not been a lot of research into their application to pedestrian trajectory prediction, compared to LSTMs and GANs. However, technically speaking, natural language processing and time series prediction are similar tasks from a machine learning standpoint, as they are both sequence-to-sequence models, and so follow an encoder-decoder architecture (covered more deeply in section 3.2.1). For this reason, there is lots of excitement around transformers and their potential to revolutionise the field of object trajectory prediction.

The earliest research papers into applying transformer networks to the task of pedestrian trajectory prediction were released in 2020 [2, 75]. Giuliari's paper introduced a simple transformer-based model which does not account for social forces between pedestrians, and outperformed LSTM and GAN based models of the same variety in terms of average displacement error (ADE) and final displacement error (FDE). Yu's paper incorporated social forces and crowd motion dynamics into their model, called STAR, and outperformed several state-of-the-art LSTM and GAN-based models, including Social-LSTM [62] and SoPhie [70], as shown in the table below.

	Performance (ADE/FDE)					
Deterministic	ETH	HOTEL	ZARA1	ZARA2	UNIV	AVERAGE
LR	1.33/2.94	0.39/0.72	0.62/1.21	0.77/1.48	0.82/1.59	0.79/1.59
LSTM	1.13/2.39	0.69/1.47	0.64/1.43	0.54/1.21	0.73/1.60	0.75/1.62
S-LSTM[1]	0.77/1.60	0.38/0.80	0.51/1.19	0.39/0.89	0.58/1.28	0.53/1.15
CIDNN[49]	1.25/2.32	1.31/1.86	0.90/1.28	0.50/1.04	**0.51/1.07**	0.89/1.73
SocialAttention [45]	1.39/2.39	2.51/2.91	1.25/2.54	1.01/2.17	0.88/1.75	1.41/2.35
TrafficPredict [38]	5.46/9.73	2.55/3.57	4.32/8.00	3.76/7.20	3.31/6.37	3.88/6.97
SR-LSTM [53]	0.63/1.25	0.37/0.74	**0.41/0.90**	0.32/**0.70**	**0.51**/1.10	0.45/0.94
STAR-D	**0.56/1.11**	**0.26/0.50**	**0.41/0.90**	**0.31**/0.71	0.52/1.15	**0.41/0.87**
Stochastic	ETH	HOTEL	ZARA1	ZARA2	UNIV	AVERAGE
SGAN† [16]	0.81/1.52	0.72/1.61	0.34/0.69	0.42/0.84	0.60/1.26	0.58/1.18
SoPhie*† [40]	0.70/1.43	0.76/1.67	0.30/0.63	0.38/0.78	0.54/1.24	0.54/1.15
STGAT† [21]	0.65/1.12	0.35/0.66	0.34/0.69	0.29/0.60	0.52/1.10	0.43/0.83
STAR†	**0.36/0.65**	**0.17/0.36**	**0.26/0.55**	**0.22/0.46**	**0.31/0.62**	**0.26/0.53**

Table 2: Comparison of Yu's transformer-based pedestrian trajectory prediction model to state-of-the-art. STAR-D denotes deterministic version of STAR. [2]

In addition to impressive accuracy numbers, using a transformer-based model for trajectory prediction also has some benefits in terms of functionality. One such benefit is the way that it deals with missing observations from input data. That is, when there are certain time steps where there are no co-ordinates provided for the location of a pedestrian in the frame. This could happen, for example, if a pedestrian walks behind an obstacle such as a large vehicle and is obstructed from view for a few time steps. In recurrent neural networks such as LSTM, this is typically dealt with by designing ad-hoc extensions for filling in estimates for missing entries (known as 'hindsighting' [76]). Transformers, on the other hand, have no need to fill in missing entries in the data, since it does not operate under the assumption that all observations are taken at equidistant intervals. Instead, the transformer uses positional encoding vectors to convey when an observation took place.

However, there are some disadvantages to transformer networks compared to recurrent neural networks such as LSTMs. Firstly, transformers generally have a much higher memory and computational complexity than sequential networks, especially for longer sequences. This is because the complexity of the attention matrix is quadratic in the input sequence length, i.e., N^2, while sequential networks are linear in the input sequence length [71, 77]. This leads to much higher computational cost and thus much higher power consumption. This can be an issue for a car's on-board pedestrian detector, as a large GPU could end up draining the car's battery. Additionally, because of this limitation, there is a limit on the length of the input sequence that a transformer network, while LSTMs can theoretically handle input sequence of infinite length.

Chapter 3 Background Theory

3.1 Object Detection

In this section, I will start by briefly covering some of the fundamental ideas behind object detection in deep learning, before moving on to some of the theory of single-stage object detection. This section assumes some prior knowledge of backpropagation and gradient descent as they apply to the training of deep neural networks.

3.1.1 Convolutional Neural Networks

The primary building block behind almost all modern object detection algorithms is the convolutional neural network (CNN), which is based on the ideas introduced by LeCun et al. in 1989 [78]. The specialty of CNNs lies in their ability to learn spatial dependencies in data, enabling them to detect patterns, such as those in images. CNNs are deep neural networks with many hidden layers, including pooling layers, activation layers, fully connected layers, and most notably, convolutional layers. Each of these convolutional layers contains several filters, with each filter responsible for detecting a certain pattern in the data. In the context of image recognition, filters in early layers are typically responsible for edge or corner detection. As we progress deeper into the network, we might find more sophisticated filters that can detect specific objects or features, such as eyes, mouths, and hands. These filters are not hardcoded but are instead learned by training the CNN on a large dataset of images using backpropagation to update weights and biases to minimise error.

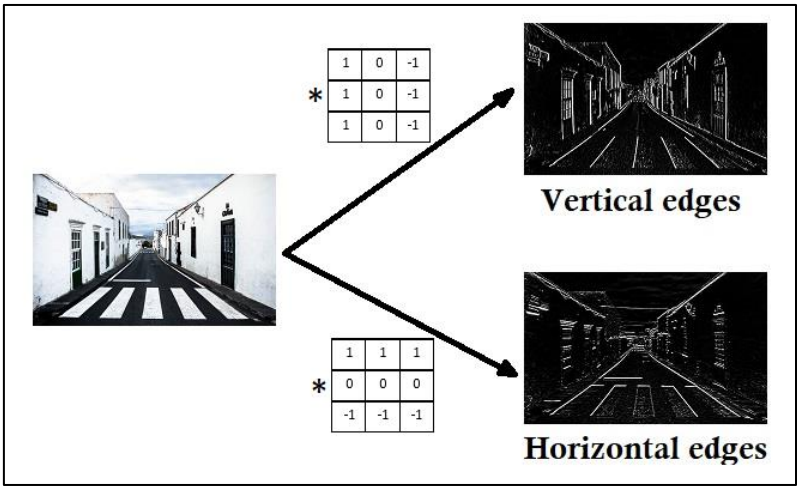


Figure 24: Visualisation of basic edge detector layers in CNN [79]

For several decades after LeCun's research, convolutional neural networks did not have much practical application for image classification because of their high computational cost and complexity. This began to change around 2012, when Krizhevsky et al. released AlexNet [10], a deep CNN that achieved record accuracy at classifying over 1.2 million high-resolution images from the ImageNet dataset into 1000 different classes.

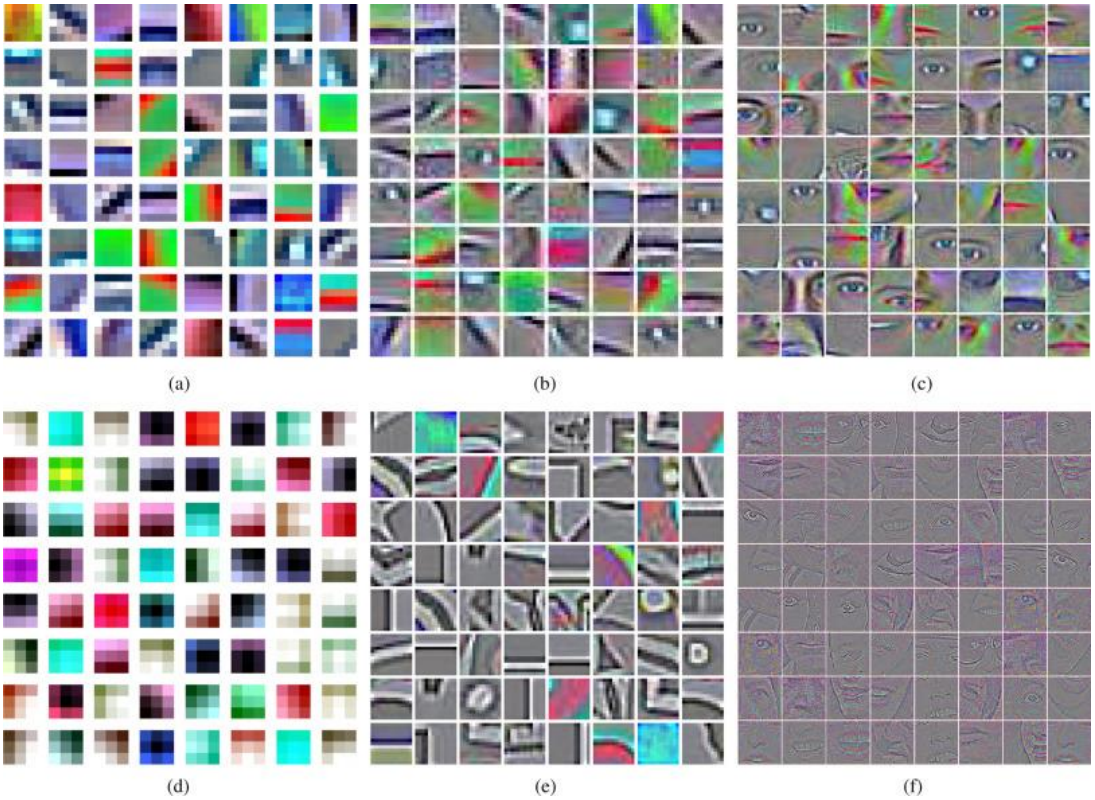


Figure 25: Visualisation of features learned from facial images [80]

The success of AlexNet spurred a full-blown revolution in the convolutional neural network and image classification space. Researchers have developed various new architectures and techniques, such as VGGNet [81], GoogLeNet (Inception) [82], and ResNet [83], which have further improved the performance of CNNs in image classification and other computer vision tasks. These architectures introduced concepts such as deeper networks, inception modules, and residual connections to make the training process more efficient and accurate. Additionally, transfer learning has enabled models to be pre-trained to perform a specific task, such as VGG16 for image classification, allowing others to leverage their knowledge without needing to train a model themselves, which can be computationally expensive.

3.1.2 Single-Stage Detection

Prior to the introduction of single-stage object detectors, existing detectors were, for the most part, just repurposed images classifiers. The detector would be made up of two distinct networks: one which proposes regions which it thinks contain an object, and another which classifies the contents of this region using a pre-trained classifier, such as VGG16. While this performed well in terms of accuracy, it was computationally expensive and slow, which made it ill-suited for the task of real-time object detection from a video feed. It was exactly for this reason that the single-stage object detector was developed. Two of the most influential works on this type of algorithm were released in 2015: You Only Look Once (YOLO) [27] and Single Shot Detector (SSD) [28]. For the purpose of this paper, I will only be focusing on YOLO, although the algorithms are very similar.

Unlike the two-stage detection approach, the single-stage algorithm combines the tasks of object localisation, classification and confidence scoring into a single neural network. In this single-stage algorithm, the entire process is streamlined, and predictions are generated in a single forward pass though the network. This also enables easier end-to-end optimisation of the entire network, which is much more difficult with two-stage algorithms. The YOLO model takes in an input image of fixed size (448 x 448 pixels in YOLOv1) and divides the image into an $S \times S$ grid. If the centre of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each cell predicts B bounding boxes and confidence scores for those boxes. The confidence scores reflect the probability that the model thinks that the bounding box contains an object, as well as how accurately drawn the bounding box is. Specifically, the confidence score can be calculated as $P(object) * IoU_{pred}^{truth}$, where IoU_{pred}^{truth} is the area of the intersection of the predicted bounding box and the truth bounding box, divided by the area of the union of these two bounding boxes.

Figure 26: Input image divided into $S \times S$ grid [27]

Each grid cell also predicts C conditional class probabilities, $P(class_i|object)$. These probabilities are conditioned on the grid cell containing an object, so even if the previous step determined that there is no object in the cell, class probabilities are still calculated regardless. Only a single set of class probabilities is calculated per grid cell, regardless of the number of bounding boxes in each cell, B. This results in an output feature map with dimensions $S \times S \times (B * 5 + C)$. During training, YOLO learns to minimize the difference between the predicted bounding box coordinates, confidence scores, and class probabilities and the ground truth values. The loss function combines the errors for the bounding box coordinates, the confidence scores and the class probabilities. YOLO's regression-based approach enables the model to predict bounding boxes more efficiently and in a single forward pass, compared to classification-based methods that often require separate steps for region proposal generation and classification.

Finally, at test time, we multiply the conditional class probabilities with the individual box confidence predictions for each bounding box, as follows:

$$P(class_i|object) * P(object) * IoU_{pred}^{truth} = P(class_i) * IoU_{pred}^{truth}$$

This gives us class-specific confidence scores for each bounding box, encoding the probability of that class appearing in the box and how well the predicted box fits the object.

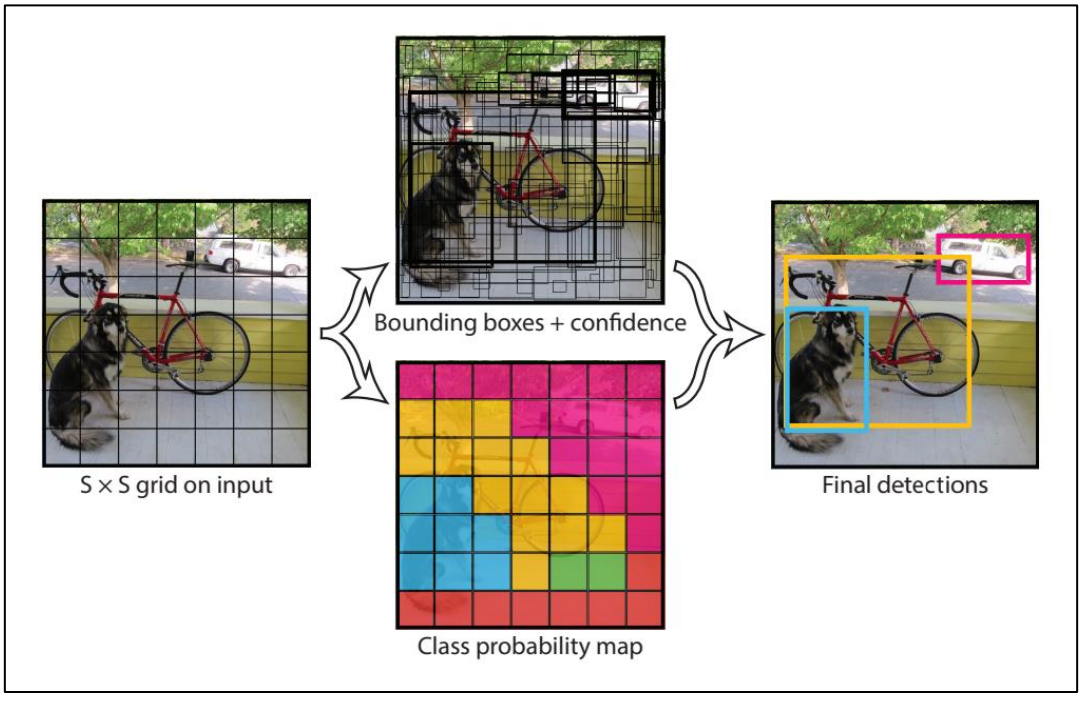


Figure 27: Overview of YOLO network [27]

3.2 Object Tracking & Prediction

3.2.1 Sequence-to-Sequence Model

While, on the surface, the task of pedestrian trajectory prediction may seem quite similar to pedestrian detection, in reality it is a different problem entirely, which requires a totally different approach. In the case of object detection, we dealt with individual frames of video, which had a fixed resolution (1920x1920). However, for an effective pedestrian trajectory prediction model, we will need to use many consecutive frames of video, and the number of frames we will have to deal with can vary case by case. The prediction that our model makes will also be a sequence, where each frame's prediction is dependent on the frames that came before it. This means that we will need a sequence-to-sequence (seq2seq) model. The problem is that deep neural networks can only be trained on vectors of a fixed dimension, and not on variable length sequences.

For us to train a deep neural network which will enable us to predict the trajectory of pedestrians, we must first encode the variable length input sequence into a fixed length vector. We can then decode this vector with a separate network to give us a sequence of predictions for a chosen number of output frames. We can achieve this using an encoder-decoder architecture, which was introduced by K Cho et al. [46], as well as I Sutskever et al. [45], both in 2014.

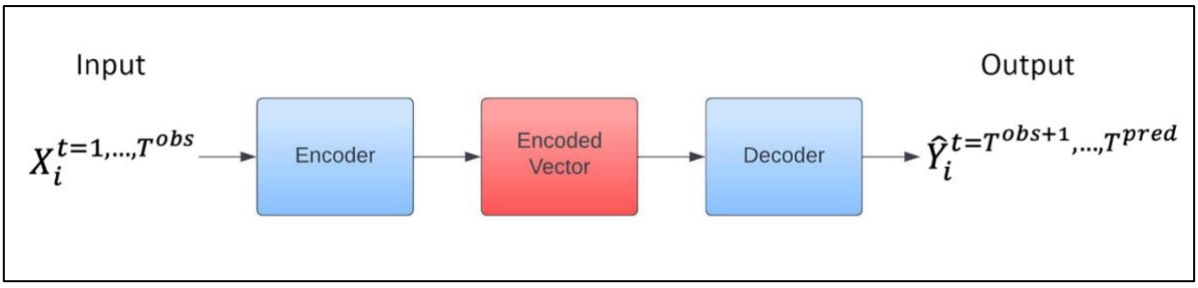


Figure 27: Encoder-Decoder architecture for a sequence-to-sequence model

As you can see from the above figure, there are three main blocks in the encoder-decoder model:

- Encoder
- Hidden Vector
- Decoder

The internal workings of the encoder and decoder can vary, and I will cover some of the different options in the following sections. At a high level, the encoder takes a sequential input (e.g. a string of words, a series of frames of video, stock price data), and outputs a fixed length vector which is essentially a compressed representation of the sequence. This vector is then fed into the decoder, which uses a similar network to the encoder, to give us the output.

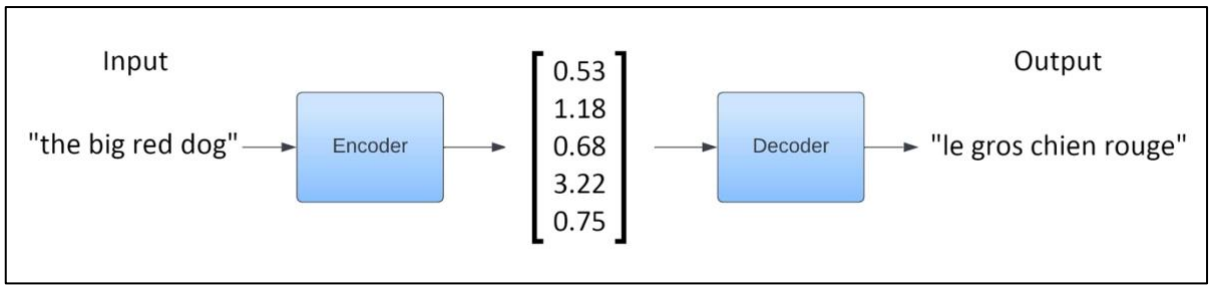


Figure 28: Encoder-decoder architecture for machine translation

To further illustrate this architecture, I will use the example of text translation from English to French. As you can see in the above figure, the encoder compresses the input sequence down to a latent representation vector, which can be decoded by the decoder to produce the output sequence. The encoder-decoder architecture is used today by Google Translate and many other machine translation models [84]. In the following sections, I will explain some of the models which have been used in the past inside the encoder and decoder blocks, as well as some of the cutting-edge networks which are revolutionising sequence-to-sequence modelling.

3.2.2 Recurrent Neural Networks

The first and most basic network that can be found inside the encoder and decoder blocks is the recurrent neural network. Early versions of recurrent neural networks were introduced by a group of researchers at UC San Diego in the late 1980s [48, 49], most notably by J Elman (1990) [50]. They aimed to solve a major problem with previous neural networks: the inability to use previous information to inform future decisions and therefore the inability to handle sequential data.

The key feature of the RNN is that it has feedback connections that allow the output of the previous time step to be fed back into the network at the next time step. This allows the network to capture temporal dependencies in the data, such as the context of a word in a sentence. Recurrent neural networks can therefore be used in various sequential applications such as machine translation, image captioning and sentiment analysis.

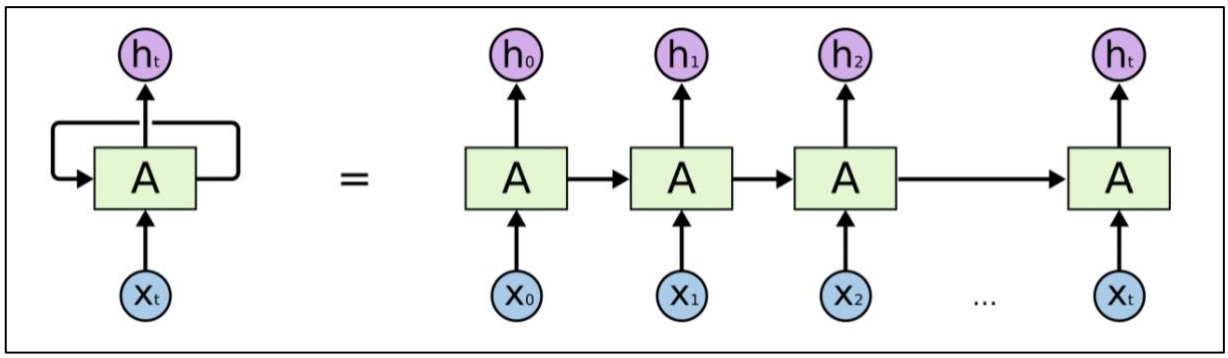


Figure 29: An unrolled vanilla recurrent neural network [85]

As you can see in the above figure, at each time step t, the input to the network is the current input x(t) and the output from the previous time step h(t-1). The network computes a weighted sum of these inputs and applies a non-linear activation function to produce the output h(t) for that time step. This output is then used as input for the next time step. While there are several variations of the RNN, the most referenced is the vanilla RNN or Elman network, which has the following equations:

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$

$$y_t = \sigma_y(W_y h_t + b_y)$$

Variables and functions:

- x_t: input vector
- h_t: hidden layer vector
- y_t: output vector
- W, U and b: parameter matrices and vector
- σ_h and σ_y: activation functions

As you can see from the above equations, the weights and biases (W, U, b) are not dependent on time and are equal for every time step. This is the root cause of the biggest shortcoming of vanilla recurrent neural networks: the vanishing and exploding gradient problem. As the number of time steps in the sequence increases, the values in the weights matrices are continuously multiplied by themselves. For values less than 1, this leads to a vanishing gradient (of loss with respect to weights) and for values greater than 1, this leads to an exploding gradient. This problem is explored in-depth by Hochreiter (1991) [86] and Bengio et al. (1994) [51], among others [52, 53].

The practical effect of the vanishing gradient effect is that the vanilla recurrent neural network is not very effective at learning long-term dependencies, and thus can only use very recent information to inform its current decision. Returning to the example of machine translation, the model might be able to extract contextual information from a few words back in the sentence but will be less effective if the contextual clues are too far back in the text. This can become a major problem in machine translation, especially when dealing with homonyms. For example, “Limerick are playing Cork tomorrow. Where is the match?” vs “I have candles for the cake. Where is the match?”.

Another drawback of the vanilla RNN is that all the computation must be performed sequentially, because each time step is dependent on the output of the time step that comes before it. This makes recurrent neural networks computationally expensive and relatively slow to train. This also means that they cannot take advantage of the parallel computing advantages of modern GPUs.

3.2.3 Long Short-Term Memory

Long Short-Term Memory networks – typically referred to as LSTMs – are a special kind of recurrent neural network which are more capable of learning long-term dependencies than the vanilla RNN. LSTMs were introduced by Hochreiter & Schmidhuber (1997) [54], and have been further refined and by many people in following work, most notably F Gers et al. [55]. LSTMs are explicitly designed to avoid the vanishing and exploding gradient problem and are thus much more effective at learning long-term dependencies. All recurrent neural networks are made up of a chain of repeating modules of neural networks. It is inside these repeating modules where LSTMs differ from vanilla RNNs, as shown in the two figures below.

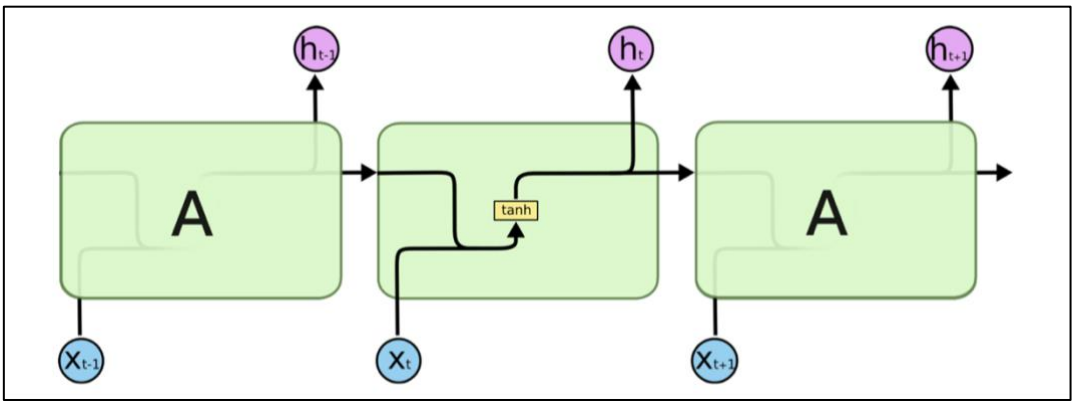


Figure 30: Repeating module of a vanilla RNN [85]

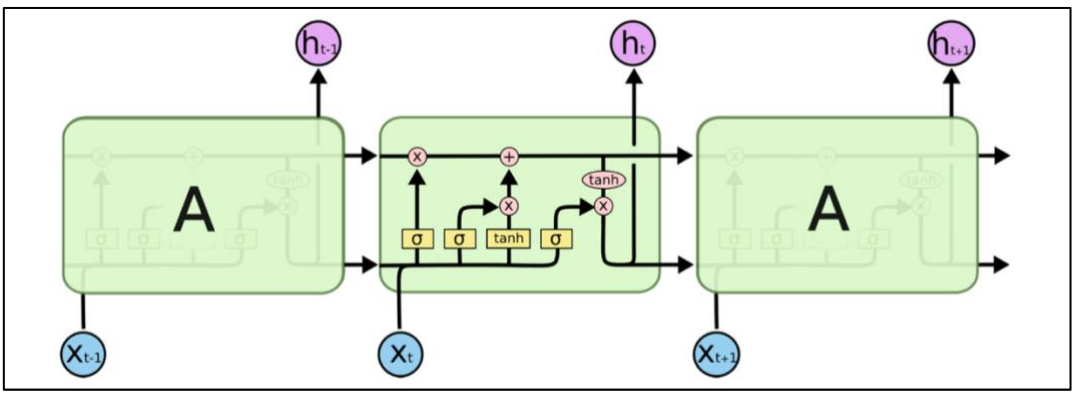


Figure 31: Repeating module of an LSTM [85]

As you can see from Figure 30 above, the vanilla RNN has a very simple structure: usually a single layer such as a $tanh$ layer. The LSTM, on the other hand, is a bit more complex, with four network layers interacting with each other, as I will outline below.

The key to the LSTM is the cell state, the horizontal line running through the top of the block diagram. The cell state is analogous to a conveyor belt, which carries long-term memories along the entire chain, with only some linear changes made to it. Because the changes to the long-term dependencies are linear and not exponential like in the vanilla RNN, the LSTM is immune from the vanishing and exploding gradient problem.

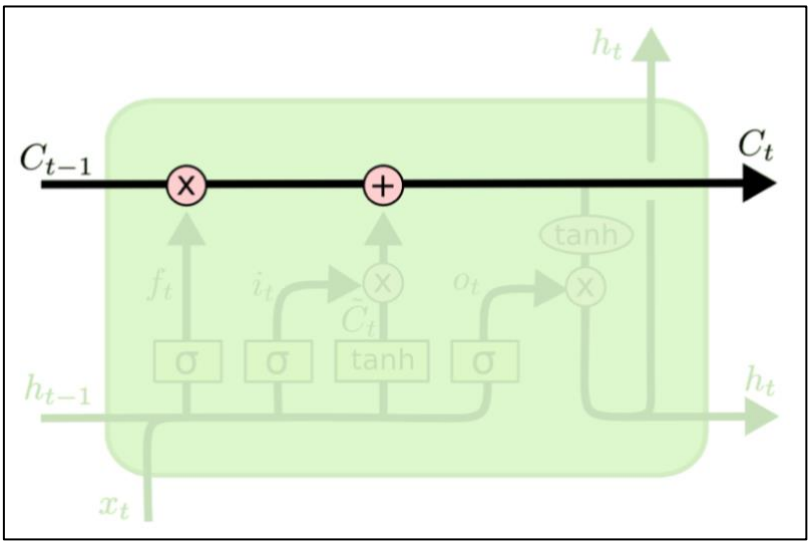


Figure 32: LSTM cell state [85]

The LSTM does need to decide what information it allows to make changes to the cell state. This is where the gating mechanism comes in. The LSTM has three gates: the forget gate, the input gate and the output gate. As shown in the below figure, each gate is made up of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer outputs a number between 0 and 1, where zero means “let nothing through the gate” and one means “let everything through the gate”.

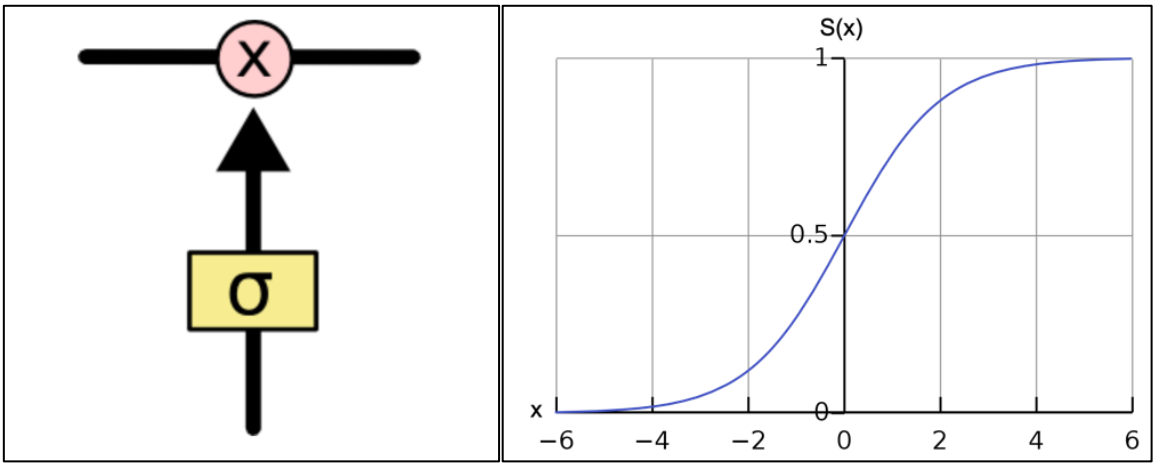


Figure 33: LSTM gate and sigmoid function plot [85]

The first gate to consider in the LSTM block is the forget gate, which decides which information already in the cell state (long-term memory) we want to throw away, and which information we want to keep. It looks at the previous output, h_{t-1}, and the current input, x_t, and outputs a number between 0 and 1 for each number in the cell state, where 1 represents "completely keep this information", and 0 means "completely forget this information".

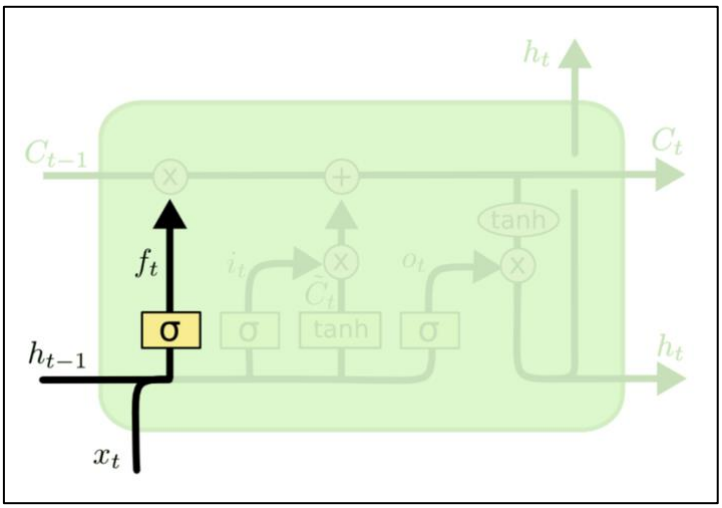


Figure 34: LSTM forget gate mechanism [85]

Going back to our example of machine translation, the cell state might include the gender of the current subject, so that the proper pronouns can be used. When we come across a new subject, we might want to completely forget the gender of the old subject. In this case, we might want to set the forget layer output equal to zero for the number representing gender, to completely throw away the old gender information and make space for the new information, which will be added in the next stage. The forget layer output can be represented with the following expression:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

Variables and functions:

- x_t: current input vector
- h_{t-1}: previous output vector
- W_f: forget layer weights
- b_f: forget layer biases
- σ: sigmoid activations function

The next step is to decide what information we want to store in the cell state. This step has two parts. First, a sigmoid layer called the 'input gate layer' decides which values we want to update. Then, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to our cell state. The outputs of these two layers will get combined using a pointwise multiplication operation in the next step to create an update to the cell state. In the context of the machine translation example, we might want the input vector to have a value of 1 for the value representing gender, because we want to add in the new gender information for our new subject, which is stored in the candidate cell state vector.

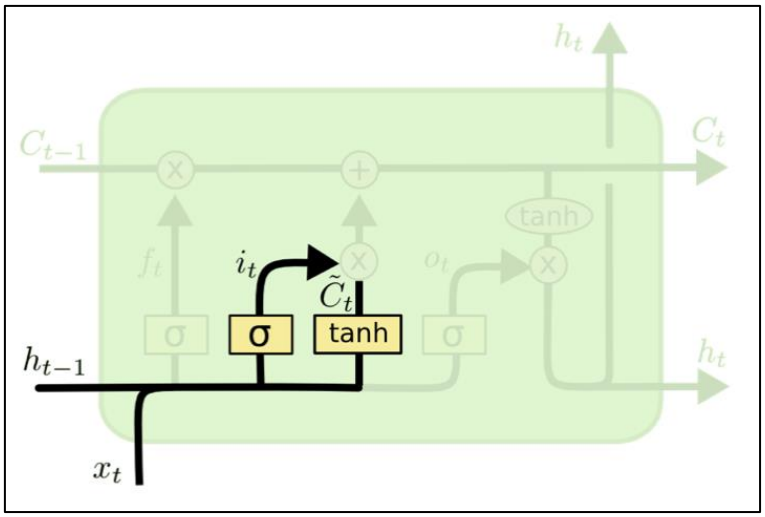


Figure 35: LSTM input gate and candidate cell state layers [85]

We calculate the input gate layer output, i_t, and the candidate cell state vector, $\tilde{C}_t$, with the following expressions:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Variables and functions:

- x_t: current input vector
- h_{t-1}: previous output vector
- W_i: input layer weights
- W_C: candidate cell state layer weights
- b_i: input layer biases
- b_C: candidate cell state layer biases
- σ: sigmoid activations function

Next, we need to update the old cell state, C_{t-1}, into the new cell state, C_t. This is just a case of executing pointwise mathematical operations on the vectors we've just calculated. We multiply the old state by f_t, in order to forget the values we decided to forget earlier. We then add $i_t * \tilde{C}_t$, the new candidate values, scaled by how much we decided to update each state value. Using the example of the machine translation, this is where the gender of the older subject is removed from the cell state and the new gender is added to the cell state.

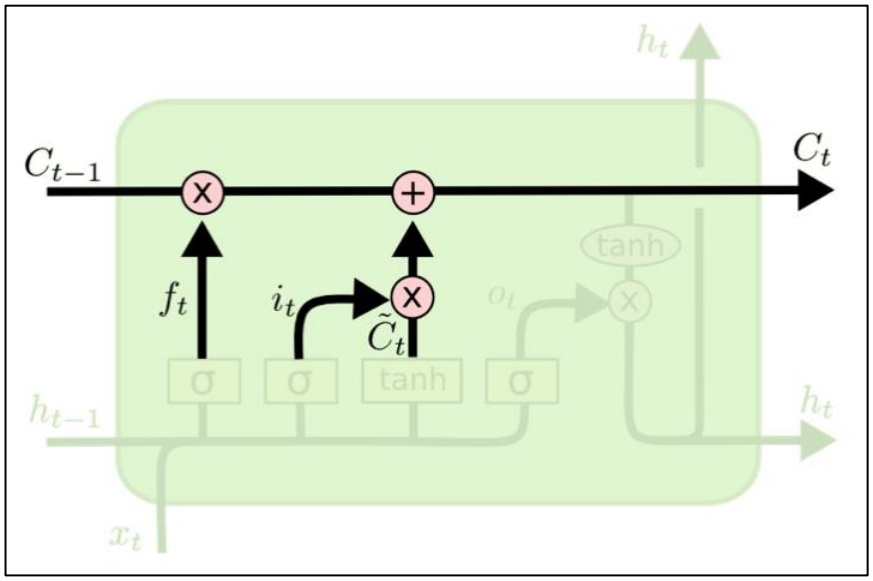


Figure 36: Updating the LSTM cell state [85]

The expression for the new cell state vector is given by the following, where C_t is the current cell state and C_{t-1} is the previous cell state:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Variables and functions:

- f_t: current forget vector
- C_{t-1}: previous cell state vector
- i_t: current input layer vector
- $\tilde{C}_t$: current candidate cell state vector

The final layer of our LSTM block is where we decide what we want to output. The output layer vector is once again a calculated using the previous output, h_{t-1}, and the current input, x_t, before passing through a sigmoid layer to give us values between 0 and 1. This output layer vector is then combined with a tanh-filtered version of the cell state, C_t, using a pointwise multiplication operation to give us our output, h_t.

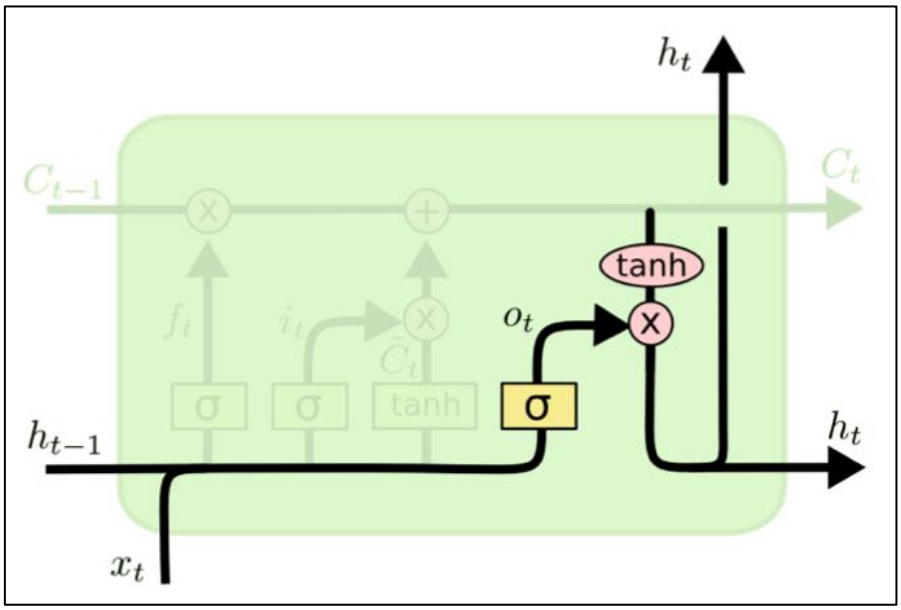


Figure 37: LSTM block output layer [85]

Finally, the output layer vector, o_t, and the LSTM output vector, h_t, are given by the following expressions:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

Variables and functions:

- x_t: current input vector
- h_{t-1}: previous output vector
- W_o: output layer weights
- b_o: output layer biases
- σ: sigmoid activations function
- C_t: cell state vector

Using this more complex internal series of interacting layers, the LSTM eliminates the vanishing gradient problem, and can therefore outperform the vanilla RNN by an order of magnitude or even more in terms of learning long-term dependencies, depending on the specific task and architecture [87].

There are, however, some drawbacks associated with the LSTM network. Firstly, due to their added complexity of vanilla RNNs, they are even slower to train than the already slow RNNs. Also, because each block of the network is dependent on the output of the previous block, LSTMs must be trained sequentially, and cannot take advantage of the parallel computing capabilities found in modern GPUs. Fortunately, there is new sequence-to-sequence model architecture that aims to solve these problems while also further improving on the LSTM's long-term dependency learning abilities: the transformer, which I covered more deeply in chapter 2.

Chapter 4 Methodology

In this chapter, I will address the different steps taken to perform the creation of a pedestrian detection and trajectory prediction system using imagery from a car's on-board camera. In this chapter, I will cover:

- the language and coding environment I have used;
- the dataset I have chosen to use;
- the approach I have taken to build my two models.

4.1 Language & Environment

I decided to write my project using the Python programming language because of its abundance of machine learning and image processing libraries, such as PyTorch, TensorFlow, PIL and OpenCV. I wrote my code in Python Notebooks because I found the interactivity very useful for my experimentation with the data. I used the Google Colab coding environment because it enabled me to remotely use Google's servers and GPUs so that I didn't have to run the computationally expensive model training on my own hardware.

4.2 Data Collection & Processing

The dataset I have chosen is the Waymo Open Dataset, which is a public research dataset released by Waymo, a subsidiary of Google. This is a set of 2030 video segments of 20 seconds each, collected at 10Hz. The dataset contains label data for vehicles, pedestrians, cyclists and signs. There is some diversity of scenes, varying from urban to suburban, daytime and nighttime, including some scenes with construction and poor weather conditions. However, because the data is only collected in San Francisco, Mountain View and Phoenix, which are geographically and culturally very close to each other, the model may end up being quite overfit to this particular training environment [88]. Some of the example scenes from the Waymo Open Dataset are seen in the figure below.

Figure 38: Sample scenes from the Waymo Open Dataset

I downloaded several GB of data from the Waymo Open Dataset Google Cloud bucket, using the `gsutil` command in Google Colab. Because the data from Waymo Open Dataset is stored in .tfrecord files, I had a bit of processing work to do to extract more human-readable data from them. After spending some time trying to extract the data myself by following documentation and proto files from the official GitHub repo, I discovered a Python library called `simple_waymo_open_dataset_reader`, which greatly simplified the process. Using this library, I extracted the front camera images as JPEGs and stored class, bounding box and unique ID data for every labelled object in text files, as shown below.

```
!cat labels/s0118f0117.txt

2  0.343592  0.529995  0.370895  0.548088  0a5b558e-e50c-4319-a1ba-2a45da674c88
2  0.282077  0.536081  0.315302  0.548581  2d7dad90-9d66-4d06-a3a1-855c66e17eaf
2  0.396882  0.535259  0.424185  0.554667  32a8e318-8180-4b9f-902a-b984bfae62d5
2  0.198769  0.543154  0.303541  0.575062  459aeb99-e5b2-4f13-9009-d0a95e31eac3
2  0.342604  0.539864  0.359381  0.556641  6a9ca3ec-4c27-4488-8fc1-f1ac4f9c9637
2  0.497542  0.524403  0.526818  0.549733  71964865-4611-486b-bccf-3918131202f1
2  0.575010  0.529337  0.634880  0.551377  7f0ff5ee-3aec-4486-aef4-b47effada5c3
0  0.525174  0.525390  0.532082  0.538877  a00a367a-377b-4e12-ba98-141c757b85a9
```

Figure 39: Example label file for a given video frame

For my pedestrian trajectory prediction model, I want to narrow my dataset down to sequences of video in which a given pedestrian is present for 20 consecutive frames. This is because I want to use 10 frames (1 second) of past data to inform my predicted trajectory for 10 future frames. Once I narrowed down my data to these pedestrians, I stored their bounding box data for a 20-frame sequence in a text file. I also created a cropped image containing the pedestrian's bounding box, enlarged by a factor of 1.5 in order to capture some scene context from the pedestrian's surroundings.

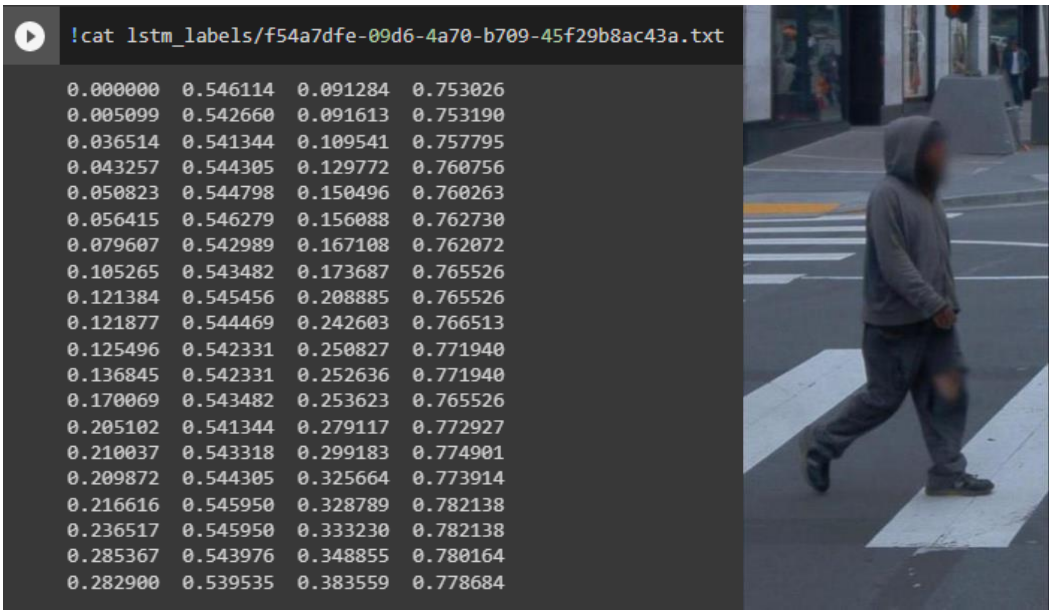


Figure 40: Bounding box data and sample image for pedestrian trajectory prediction model

4.3 Pedestrian Detection Model

To recap, the purpose of my pedestrian detection model is to take individual frames of a car's on-board camera video feed, and in real-time output bounding box data for predicted locations of every pedestrian in the frame. Because of the application of real-time pedestrian detection, I have decided that detection speed is the top priority for my model. For this reason, I have elected to take a single-stage detection approach, and will create my detector using the YOLOv7 algorithm, which has been the fastest object detection model since its release in 2022.

The YOLOv7 algorithm has the best accuracy/speed combination of any detector that operates above 5PFS and can process frames in real-time at a staggering 160 frames per second in some conditions. YOLOv7-E6 object detector (56 FPS V100, 55.9% AP) outperforms both state-of-the-art transformer-based detector SWINL Cascade-Mask R-CNN (9.2 FPS A100, 53.9% AP) by 509% in speed and 2% AP in accuracy, and convolutional based detector ConvNeXt-XL Cascade-Mask R-CNN (8.6 FPS A100, 55.2% AP) by 551% in speed and 0.7% AP in accuracy [33]. This makes YOLOv7 an easy choice for my real-time pedestrian detection model.

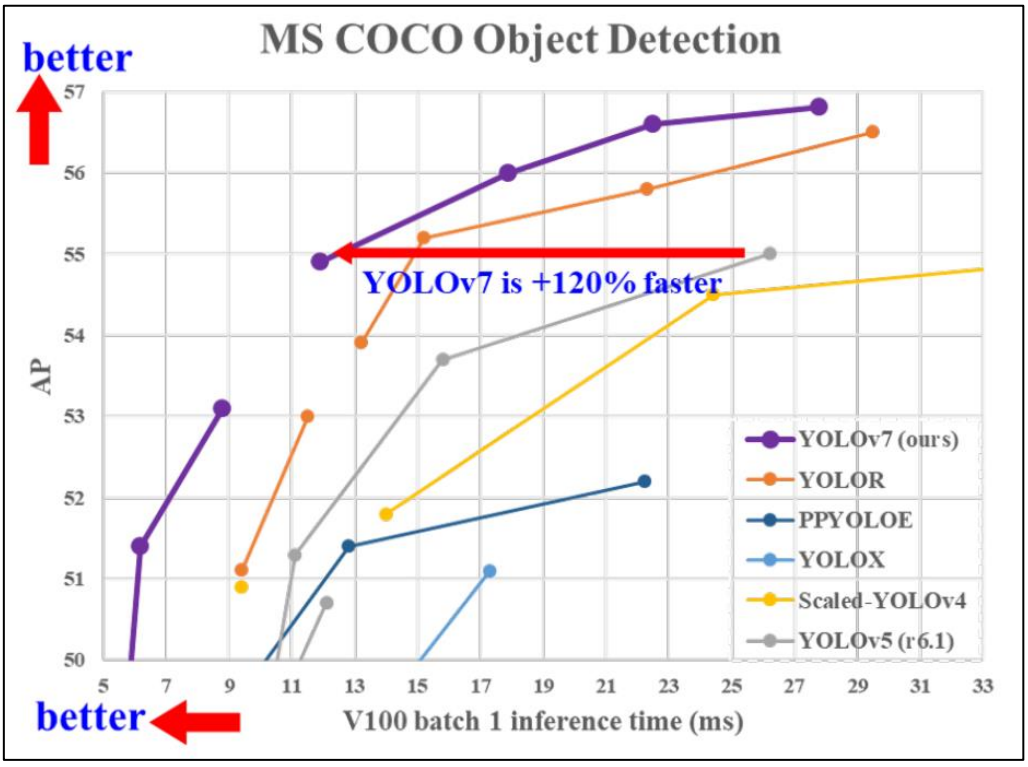


Figure 41: Comparing YOLOv7's inference speed and AP to other single-stage detectors [33]

I went with an 8:1:1 train/test/validation split, which is standard for object detection tasks such as this one. I moved each of these subsets of my data into individual directories. In addition to pedestrians, I directed my detection algorithm to also look for vehicles, cycles and road signs. This information was supplied to the YOLO configuration .yaml file, as shown below.

```yaml
train: /content/images/train/
val:  /content/images/val/
test: /content/images/test/

# number of classes
nc: 4

# class names
names: ['pedestrian', 'cyclist', 'vehicle', 'sign']
```

Figure 42: YOLO configuration .yaml file

I trained my model on a training set of 31,380 images and bounding box label files. I trained my model with a batch size of 10 for 75 epochs. To compare the results and performance of my YOLOv7 model to another baseline example, I will also train a YOLOv5 model using the same dataset and training parameters.

4.4 Pedestrian Trajectory Prediction Model

For the pedestrian trajectory prediction model, I propose an LSTM-based machine learning which take a 10-frame sequence of past video frames with bounding box data and outputs a 10-frame sequence of future bounding box data, conveying the future trajectory of each pedestrian. This model builds on the work of Tommy Browne [60], UL 2021 graduate in MsAI, which proposed using a pre-trained model to extract features from images of the pedestrian in each of the consecutive input frames and inputting these features into a convolutional LSTM network to capture spatiotemporal dependencies in the data.

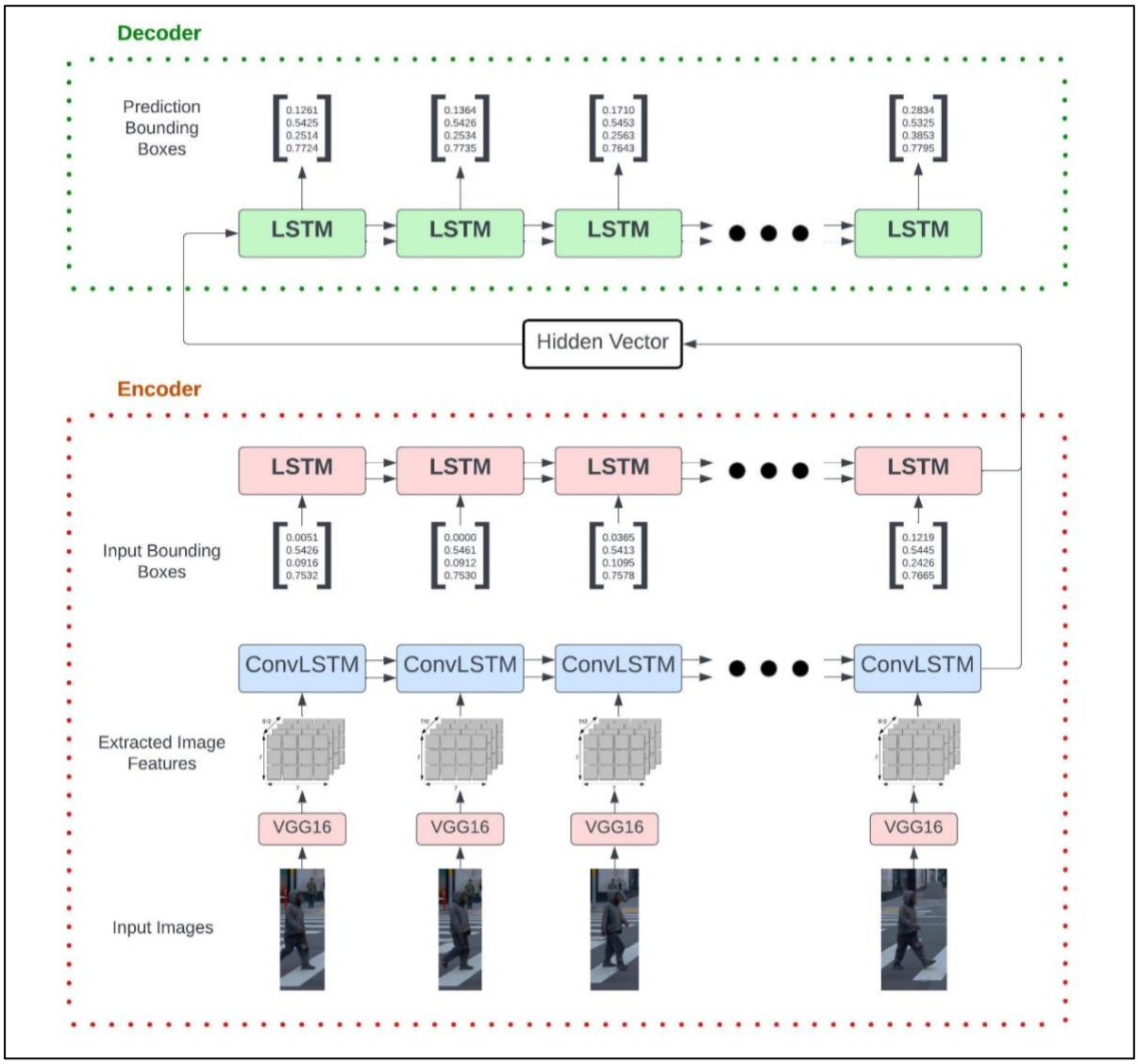


Figure 43: Pedestrian trajectory prediction network block diagram

Shown in the figure above is a high-level look at my pedestrian prediction model. As you can see, the input is made up only of a series of bounding box and image data. The cropped image is acquired by using the bounding box co-ordinates supplied by the previous object detection model and enlarging the box by a factor of 1.5, in order to allow the prediction model to gain some contextual information from the image of the scene surrounding the pedestrian. Each of the cropped images are fed into a pre-trained VGG16 model, which returns a 7x7x512 tensor, which contains a compressed representation of the features extracted from the image. These feature maps are what get input into the convolutional LSTMs, which are responsible for learning spatiotemporal dependencies within these series of feature maps, in order to help inform our models ability to predict pedestrian intention and trajectory. This sequence of convolutional LSTMs returns the latent representation of the feature maps sequence as a fixed length vector.

Similarly, the bounding box data is also fed into LSTMs, although not ConvLSTMs. These conventional LSTMs learn temporal patterns in the sequence of bounding box data, and similarly output a fixed length vector to represent the sequence. This output vector is concatenated with the ConvLSTM output to give us our hidden vector, which is sent to the decoder. The decoder then steps through the data and sequentially outputs predicted bounding box locations for pedestrians for 10 frames into the future.

I implemented my LSTM model using the Keras TensorFlow libraries in Python. The architecture of my model can be seen in the figure below. I trained my model on a training set of 1200 video sequences of 10 frames each, along with 20 consecutive frames of bounding box data (i.e., 10 frames for input and 10 frames for output ground truth). I then tested the model on a test set of 400 images.

To compare the performance of my model to a baseline, I created a basic linear regression model, which assumes that the displacement of each bounding box co-ordinate between each output frame is equal to the average displacement between each of the input frames.

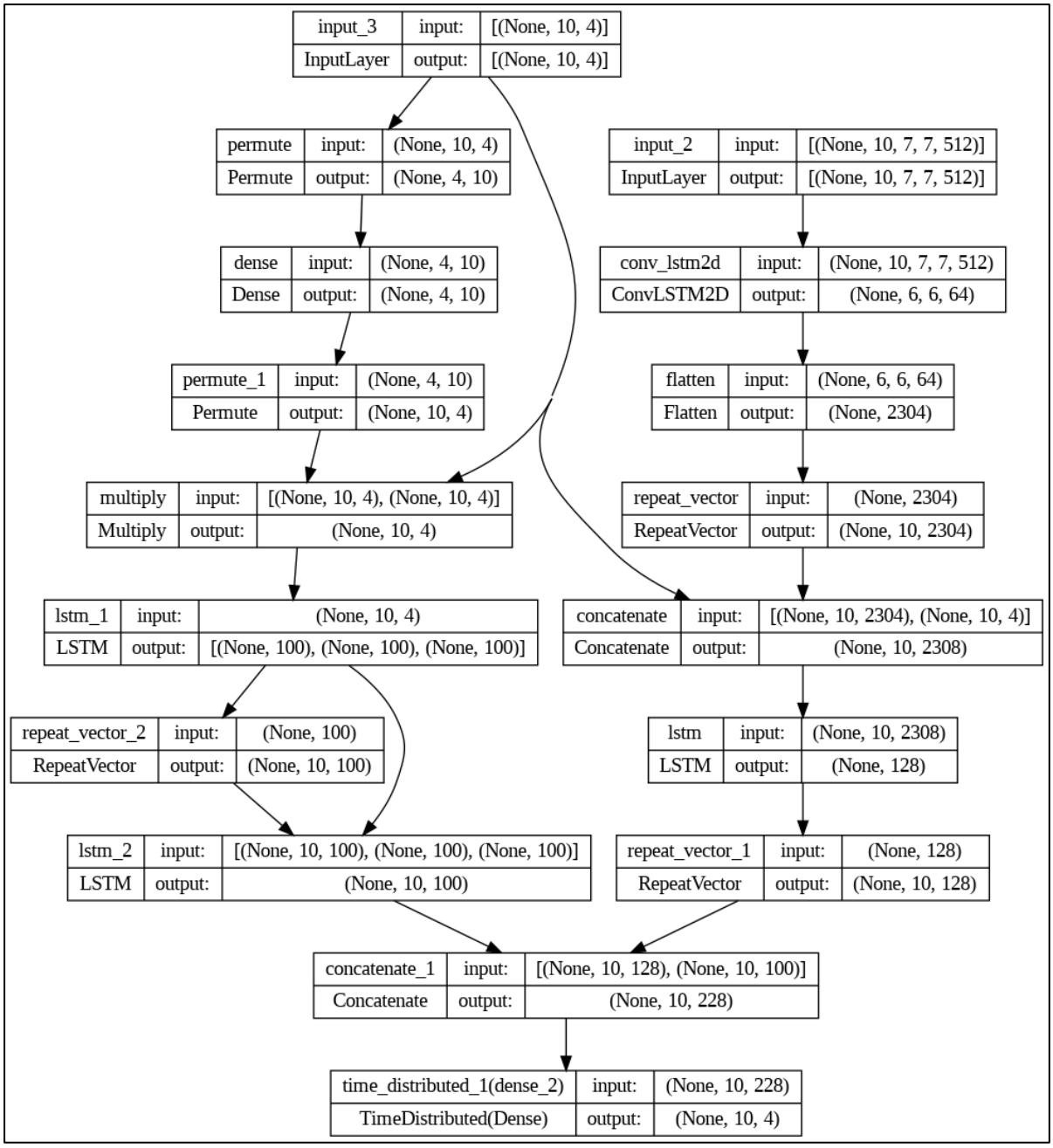


Figure 44: Pedestrian trajectory prediction network summary

Chapter 5 Results

5.1 Pedestrian Detection Model

5.1.1 Quantitative Results

I'm quite happy with the performance of the pedestrian detection model. The test statistics from my YOLOv7-based pedestrian detection model, along with the YOLOv5 baseline model, are as follows:

	YOLOv7	**YOLOv5**
Precision	0.870	0.795
Recall	0.670	0.508
mAP@0.5	0.765	0.597
mAP@0.5:0.95	0.411	0.293
Inference Speed (ms)	16.3	29.3

Table 3: Test Statistics: YOLOv7 (my model) vs YOLOv5

As you can see, my YOLOv7 model significantly outperformed the previous generation in all categories. Notably, the inference speed improved by 13ms on this dataset, enabling an 80% increase in real-time processing framerate. This is a very promising metric for my real-tie pedestrian detector.

Originally, when I saw these results I was disappointed to see that the recall was as low as 0.670. However it should be noted that many of the pedestrians in the Waymo Open Dataset images are quite far in the distance and can be as small as 5-10 pixels tall. While a good pedestrian detector would ideally pick up on these edge cases, I felt that these distant pedestrians disproportionately hurt my model's recall score. When I filtered my results to only include pedestrians with a minimum height of 100 pixels (in a 1920x1920 image), the recall jumped to 0.890.

My model also performed quite well in detection of the other object classes, as shown by the table and figures below.

Class	Instances	Precision	Recall	mAP@0.5	mAP@.5:.95
All	94677	0.885	0.676	0.758	0.449
Pedestrian	19644	0.869	0.671	0.765	0.410
Cyclist	694	0.873	0.591	0.676	0.379
Vehicle	74559	0.914	0.766	0.834	0.557

Table 4: Object detection testing stats for all classes

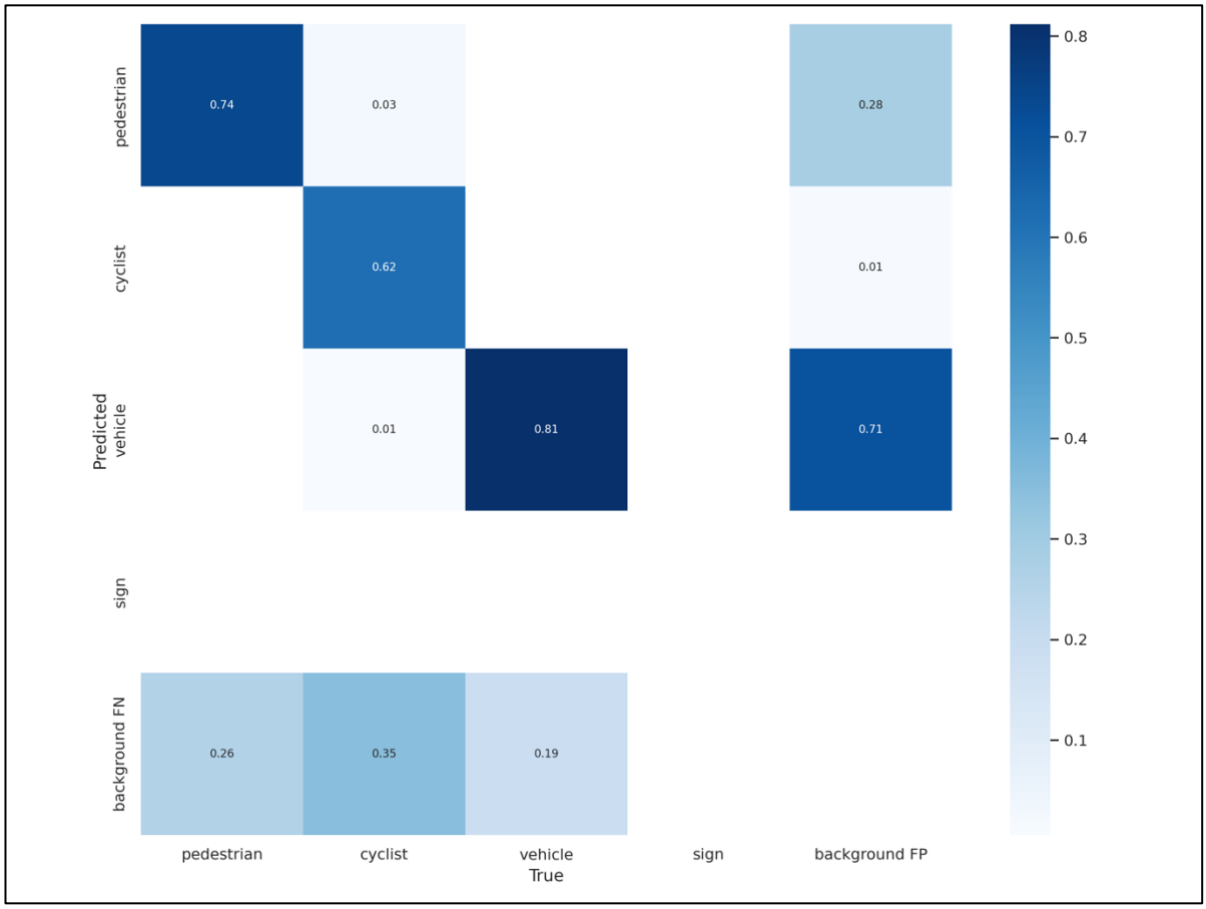


Figure 45: YOLOv7 pedestrian detector confusion matrix

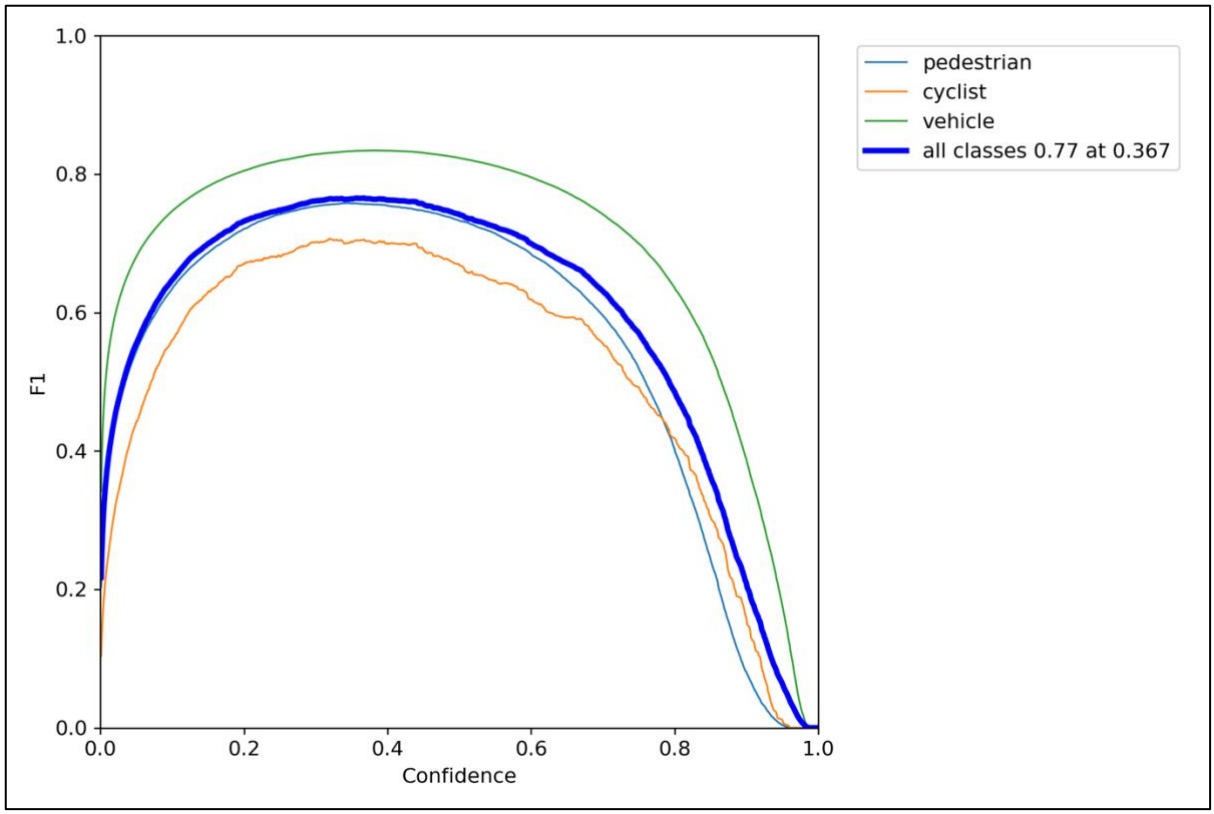


Figure 46: YOLOv7 pedestrian detector F1 curve

As we can see from the above F1 curve, the optimal operating point for the decision threshold is 0.367, which results in an F1 score of 0.770.

We can see that there is quite a steep drop-off in pedestrian detection performance for higher confidence levels. This likely points to a low level of generalisability for the pedestrian detector, compared to the other class detectors in this model. This might mean that there is not enough diversity in the training data, or that my model is overfit to the training set. This steep drop-off also points to an ineffectiveness in handling hard-to-detect samples, which may be affected by factors such as occlusion, varying lighting conditions or smaller scale pedestrians, such as those further in the distance.

The F1 curve does, however, point to much improved performance over the YLOv5 version, which achieved a peak F1 score of 0.62 at a decision threshold of 0.362. The v5 model also saw a steeper drop-off for the pedestrian F1 score for higher confidence levels than the other classes.

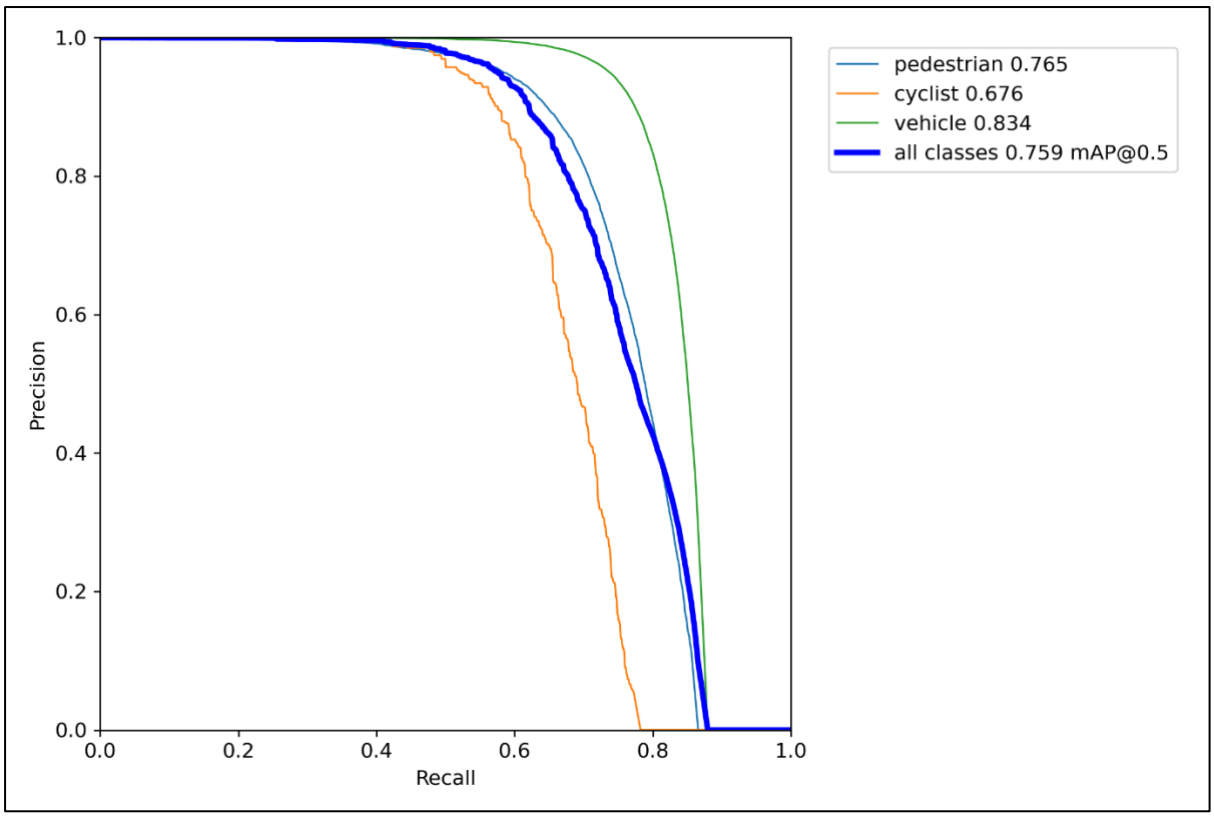


Figure 47: YOLOv7 pedestrian detector P-R curve

As we can see from the above P-R curve, my pedestrian detection model maintains a high precision for most of the recall values up until around 0.6. Above R=0.6, we see a very steep drop-off in precision, especially for the pedestrian detector. This is because the model is increasing its prediction count in an effort to decrease the number of true negatives, and thereby increase recall. However, this massively decreases the precision due to all the additional false positives.

The steepness of this drop-off in precision for higher recall, especially for pedestrians, again points to an inability for my model to pick up on the more difficult to detect samples.

5.1.2 Qualitative Results

Visually, many of the results of the pedestrian detection model are quite impressive. It performs nearly flawlessly in some crowded urban scenes with many potential hazards.

Figure 48: Detection model performing well in crowded scenes

However, the detection model is certainly not perfect. This scene in particular was quite striking to me. The model failed to detect a small child who was partially occluded by a fire hydrant to the left of this picture. The child is then detected a few frames later as the car moves further up the road. A possible solution to this shortcoming could be to use additional sensors other than just a 2D camera, such as radar or lidar for example.

Figure 49: Detection model misses child occluded behind fire hydrant

Additionally, some qualitative data shows some flaws which are inherent to the nature of this detection model which deals with each frame atomically, and does not use past data to inform the current decision. In this example, there are two pedestrians blocked from the view of the camera by a tree to the right, who then come into view a few frames later. Even with a perfect detection model, these pedestrians would likely still go undetected if each frame was handled individually. However, if the model could use data from previous frames to inform the current decision, this problem could be solved. This could be implemented using an LSTM-based model.

Figure 50: Detection fails to detect pedestrians blocked by tree

5.1.3 Analysis

Based on my observation of the pedestrian detector results, both quantitative and qualitative, I am mostly impressed by the performance of the model. It yielded very visually impressive results, handling crowded urban scenes with ease and detecting pedestrians of many different scales. Also, the inference speeds we very impressive, achieving times of up to 16.3 ms/frame, which enables live processing of video above 60 FPS.

However, it's also true that a recall value of 0.670 is absolutely unacceptable for real-world application of an on-board pedestrian detector. Based on the performance graphs above, as well as some analysis of qualitative results, I think that it is clear that this model does not sufficiently detect some of the more difficult samples, such as those which are occluded, in poor light conditions or a smaller scale.

I suggest that to account for these shortcomings, there are two immediate solutions which could improve results:

- Use more input datapoints, such as radar and lidar data, to assist in pedestrian detection. The nature of 2D camera imagery means that poor lighting or weather conditions can have a drastic effect on results. For the application of real-time pedestrian detection, its important to have many data sources to cross-reference.
- Use temporal dependencies to inform pedestrian detection. If this detection model could use data from previous frames to inform the detection decision for the current frame, I think it would yield better results, especially helping in situations where a pedestrian becomes partially or fully occluded for a few frames at a time.

5.2 Pedestrian Trajectory Prediction Model

5.2.1 Quantitative Results

The pedestrian trajectory prediction model dd not perform quite as well as I would have hoped. Here are some of the performance stats for my pedestrian trajectory prediction model, as they compare to a basic linear regression prediction:

	ConvLSTM	Linear Regression
ADE	19.954	23.728
FDE	29.029	43.753

Table 5: ADE & FDE: ConvLSTM (my model) vs Linear Regression

Average displacement error (ADE) is defined as the average L_2 distance from the centre of the predicted trajectories bounding boxes to those of the ground truth.

Final displacement error (FDE) is defined as the L_2 distance from the centre of the final predicted bounding box to the final ground truth bounding box.

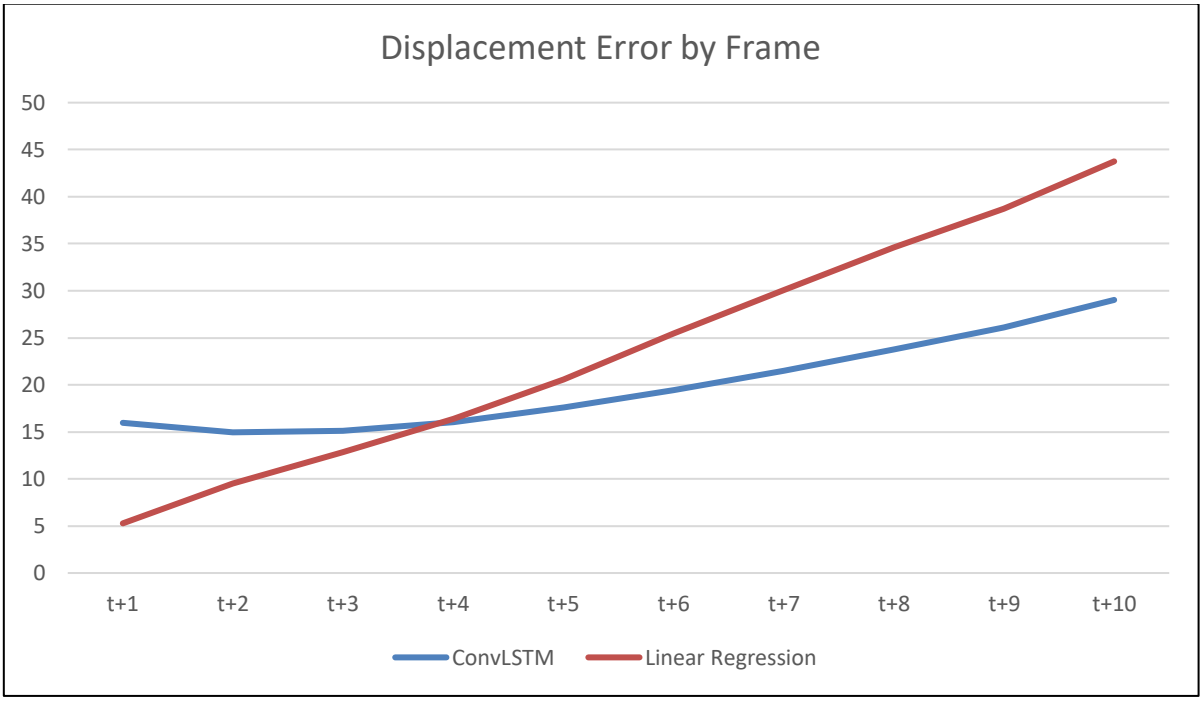


Figure 51: Displacement Error by Frame. ConvLSTM (my model) vs Linear Regression

As you can see from the above figure, the linear regression model actually outperformed my ConvLSTM-based model for the first 3 frames of prediction. However, the linear regression model's displacement error grew linearly over time, while my ConvLSTM model remained stable until t+4 before slowly increasing for each frame.

This could be partially explained by the fact that pedestrian trajectories often exhibit a short-term linearity, i.e., they tend to continue to move at same velocity on same path as previous. This is why a linear regression model may be more suited to shorter-term predictions. However, as the prediction horizon increases, the trajectories tend to become slightly more complex and non-linear. This allows the ConvLSTM to leverage its ability to model spatiotemporal dependencies, and may account for the edge that my ConvLSTM model achieves for frames after t+4.

This disparity during the first 3 frames may also be attributed to overfitting to the training set, or the fact that my training set was relatively small. If my model was too overfit on a relatively small sample of training data, it could have poor generalisation, which is why it was outperformed by the linear regression model for earlier frames. In order to counteract this effect, I could use a larger training set in the future. Additionally, I could edit the model architecture, perhaps adding layers such as dropout or weight regularisation to reduce overfitting.

5.2.2 Qualitative Results

Similarly, the pedestrian trajectory prediction model produced some impressive results when the conditions were favourable. For example, when the car is stopped at a red light.

Figure 52: Prediction model performing well in certain conditions.

However, the results quickly deteriorated outside of ideal conditions. For example, when the car changes speed or direction, the results stray significantly from the ground truth. See figure below where the car approaches a corner. This could be partially counteracted by providing more data, such as car speed & direction, as well as possibly road map data to the LSTM mode. All of these datapoints were provided by Waymo Open Dataset.

Figure 53: Trajectory predictor deteriorates when car changes direction around corner

As you can see in the below figure, the same effect can be seen when the car suddenly changes speed.

Figure 54: Trajectory predictor deteriorates when car changes speed

5.2.3 Analysis

Based on my analysis of the above pedestrian trajectory prediction results, I believe that there are lots of improvements which must be made in order to turn this model into an effective trajectory predictor.

- Use a larger training set. My training set was quite small, with only 1200 pedestrians, and as a result this model has poor generalisation, which partially explains why it was outperformed by the linear regression model for the first few prediction frames.
- Use more input datapoints. Bounding box locations and 2D imagery alone are not sufficient in providing context for an on-board pedestrian trajectory predictor. At the very least, I suggest adding car speed and direction data. This could be implemented by expanding the vector provided to the LSTMs to include speed and direction data as well as bounding box co-ordinates.
- Edit architecture to reduce overfitting.

Chapter 6 Conclusions & Future Work

In this thesis, I investigated the potential for using a computer vision to create a pedestrian detection and trajectory prediction solution, using only 2D imagery from a vehicle's on-board front-facing camera. My approach employed a combination of YOLOv7 single-stage object detection, VGG16 pre-trained image feature extraction and convolutional LSTMs among other things in order to create two distinct pedestrian detection and trajectory prediction models.

The YOLOv7-based pedestrian detection model demonstrated strong performance in most scenarios. However, it faced challenges in detecting small-scale pedestrians, occluded pedestrians, and pedestrians in poor lighting conditions. Despite these limitations, the pedestrian detection model provided accurate bounding box locations as input for the trajectory prediction model.

To build on the existing pedestrian detection model and address its limitations, two possible measures could be explored: (1) incorporating additional input data from other sensors, such as LiDAR and radar, which can provide complementary information to the 2D imagery and potentially improve detection performance for occluded, small-scale, and poorly lit pedestrians; (2) integrating temporal information from previous frames into the model's decision-making process, either by using LSTM-based techniques or other methods, to help detect pedestrians who may be temporarily invisible due to obstacles.

Our trajectory prediction model, which combined image features extracted by a VGG16-based network with bounding box coordinates processed by separate LSTMs, achieved modest performance, outperforming a basic linear regression model only marginally. The predictor performed well when the car moved at a constant speed and direction but faced difficulties in other situations.

Given these findings, future research should explore incorporating additional input data, such as car speed and orientation, to improve trajectory prediction performance. This may enable the model to better account for variations in vehicle motion and more accurately predict pedestrian trajectories in complex situations. Additionally, I would attempt to further account

for social forces due pedestrians and physical scene limitations in order to improve the performance of my pedestrian trajectory predictor in the future.

In conclusion, this thesis has contributed to the fields of pedestrian detection and trajectory prediction by developing and evaluating novel computer vision models. The pedestrian detection model offers a strong foundation for future improvements, while the trajectory prediction model, despite its current limitations, provides a basis for further development and experimentation. By addressing these limitations and building on the insights gained from this research, we can work towards creating more effective and reliable systems for pedestrian detection and trajectory prediction, ultimately enhancing the safety and efficiency of applications such as autonomous vehicles and urban planning.

References

[1] L. Jiao *et al.*, "A Survey of Deep Learning-based Object Detection," ed, 2019.

[2] C. Yu, X. Ma, J. Ren, H. Zhao, and S. Yi, "Spatio-temportal graph transformer networks for pedestrian trajectory prediction," ed, 2020.

[3] Z. Zou, K. Chen, Z. Shi, Y. Guo, and Y. Jieoing, "Object detection in 20 years: A survey," ed, 2019.

[4] P. Viola, M. Jones, A. Jacobs, and T. Baldwin, "Rapid object detection using a boosted cascade of simple features," (in English), *2001 Ieee Computer Society Conference on Computer Vision and Pattern Recognition, Vol 1, Proceedings,* Proceedings Paper pp. 511-518, 2001 2001, doi: 10.1109/cvpr.2001.990517.

[5] P. Viola, M. Jones, and I. C. SOCIETY, "Robust real-time face detection," (in English), *Eighth Ieee International Conference on Computer Vision, Vol Ii, Proceedings,* Proceedings Paper pp. 747-747, 2001 2001.

[6] P. Viola, M. Jones, D. Snow, and I. C. SOCIETY, "Detecting pedestrians using patterns of motion and appearance," (in English), *Ninth Ieee International Conference on Computer Vision, Vols I and Ii, Proceedings,* Proceedings Paper pp. 734-741, 2003 2003.

[7] N. Dalal, B. Triggs, C. Schmid, S. Soatto, and C. Tomasi, "Histograms of oriented gradients for human detection," (in English), *2005 Ieee Computer Society Conference on Computer Vision and Pattern Recognition, Vol 1, Proceedings,* Proceedings Paper pp. 886-893, 2005 2005, doi: 10.1109/cvpr.2005.177.

[8] Tyagi and M. , "HOG (Histogram of Oriented Gradients): An Overview," vol. 2023, ed: Towards Data Science, 2021.

[9] S. Yao, S. Pan, T. Wang, C. Zheng, W. Shen, and Y. Chong, "A new pedestrian detection method based on combined HOG and LSS features," (in English), *Neurocomputing,* Article vol. 151, pp. 1006-1014, MAR 3 2015 2015, doi: 10.1016/j.neucom.2014.08.080.

[10] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," (in English), *Communications of the Acm,* Article vol. 60, no. 6, pp. 84-90, JUN 2017 2017, doi: 10.2165/00129785-200404040-00005|10.1145/3065386.

[11] J. Brownlee, "A Gentle Introduction to Object Recognition with Deep Learning," vol. 2023, ed: Machine Learning Mastery, 2019.

[12] A. Ouaknine, "Review of Deep Learning Algorithms for Object Detection," vol. 2023, ed: Medium, 2018.

[13] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," (in English), *2014 Ieee Conference on Computer Vision and Pattern Recognition (Cvpr),* Proceedings Paper pp. 580-587, 2014 2014, doi: 10.1109/CVPR.2014.81.

[14] M. Sadeghi, D. Forsyth, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, "30Hz Object Detection with DPM V5," (in English), *Computer Vision - Eccv 2014, Pt I,* Proceedings Paper vol. 8689, pp. 65-79, 2014 2014.

[15] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan, "Object Detection with Discriminatively Trained Part-Based Models," (in English), *Ieee Transactions on Pattern Analysis and Machine Intelligence,* Article vol. 32, no. 9, pp. 1627-1645, SEP 2010 2010, doi: 10.1109/TPAMI.2009.167.

[16] K. He *et al.*, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," (in English), *Computer Vision - Eccv 2014, Pt Iii,* Proceedings Paper vol. 8691, pp. 346-361, 2014 2014, doi: 10.1007/978-3-319-10578-9_23.

[17] R. Girshick, "Fast R-CNN," (in English), *2015 Ieee International Conference on Computer Vision (Iccv),* Proceedings Paper pp. 1440-1448, 2015 2015, doi: 10.1109/ICCV.2015.169.

[18] S. Ren *et al.*, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," (in English), *Advances in Neural Information Processing Systems 28 (Nips 2015),* Proceedings Paper vol. 28, 2015 2015.
[19] R. Gandhi, "R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms," vol. 2023, ed: Towards Data Science, 2018.
[20] T. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature Pyramid Networks for Object Detection," (in English), *30th Ieee Conference on Computer Vision and Pattern Recognition (Cvpr 2017),* Proceedings Paper pp. 936-944, 2017 2017, doi: 10.1109/CVPR.2017.106.
[21] J. Hu, L. Jin, and S. Gao, "FPN plus plus : A SIMPLE BASELINE FOR PEDESTRIAN DETECTION," (in English), *2019 Ieee International Conference on Multimedia and Expo (Icme),* Proceedings Paper pp. 1138-1143, 2019 2019, doi: 10.1109/ICME.2019.00199.
[22] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," (in English), *2017 Ieee International Conference on Computer Vision (Iccv),* Proceedings Paper pp. 2980-2988, 2017 2017, doi: 10.1109/ICCV.2017.322.
[23] "Introduction to image segmentation for machine learning," ed. SuperAnnotate, 2021.
[24] A. Pramanik, S. Pal, J. Maiti, and P. Mitra, "Granulated RCNN and Multi-Class Deep SORT for Multi-Object Detection and Tracking," (in English), *Ieee Transactions on Emerging Topics in Computational Intelligence,* Article vol. 6, no. 1, pp. 171-181, FEB 2022 2022, doi: 10.1109/TETCI.2020.3041019.
[25] Z. Cai and N. Vasconcelos, "Cascade R-CNN: High Quality Object Detection and Instance Segmentation," (in English), *Ieee Transactions on Pattern Analysis and Machine Intelligence,* Article vol. 43, no. 5, pp. 1483-1498, MAY 1 2021 2021, doi: 10.1109/TPAMI.2019.2956516.
[26] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," ed, 2013.
[27] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," (in English), *2016 Ieee Conference on Computer Vision and Pattern Recognition (Cvpr),* Proceedings Paper pp. 779-788, 2016 2016, doi: 10.1109/CVPR.2016.91.
[28] W. Liu *et al.*, "SSD: Single Shot MultiBox Detector," (in English), *Computer Vision - Eccv 2016, Pt I,* Proceedings Paper vol. 9905, pp. 21-37, 2016 2016, doi: 10.1007/978-3-319-46448-0_2.
[29] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," ed, 2018.
[30] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, "CenterNet: Keypoint Triplets for Object Detection," (in English), *2019 Ieee/cvf International Conference on Computer Vision (Iccv 2019),* Proceedings Paper pp. 6568-6577, 2019 2019, doi: 10.1109/ICCV.2019.00667.
[31] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," ed, 2020.
[32] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal Loss for Dense Object Detection," (in English), *2017 Ieee International Conference on Computer Vision (Iccv),* Proceedings Paper pp. 2999-3007, 2017 2017, doi: 10.1109/ICCV.2017.324.
[33] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," ed, 2022.
[34] R. E. Kalman, "New results in linear filtering and prediction theory," ed, 1961.
[35] R. E. Kalman, "A new approach to linear filtering and prediction problems," ed, 1960.
[36] L. A. McGee and S. F. Schmidt, "Discovery of the Kalman filter as a practic al tool for aerospace and indutry ", ed: NASA, 1985.
[37] P. Saura. "Kalman Filter with OpenCV for calculation of trajectories in basketball." https://www.youtube.com/watch?v=MxwVwCuBEDA&ab_channel=PabloSaura (accessed 26 Mar, 2023).
[38] C. Prevost, A. Desbiens, and E. Gagnont, "Extended Kalman filter for state estimation and trajectory prediction of a moving object detected by an Unmanned Aerial Vehicle," (in

English), *2007 American Control Conference, Vols 1-13,* Proceedings Paper pp. 4123-+, 2007 2007.

[39] G. Revach, N. Shlezinger, X. Ni, A. Escoriza, R. van Sloun, and Y. Eldar, "KalmanNet: Neural Network Aided Kalman Filtering for Partially Known Dynamics," (in English), *Ieee Transactions on Signal Processing,* Article vol. 70, pp. 1532-1547, 2022 2022, doi: 10.1109/TSP.2022.3158588.

[40] A. Salhi, F. Ghozzi, and A. Fakhfakh, "Estimation for motion in tracking and detection objects with Kalman filter," ed, 2020.

[41] D. HELBING and P. MOLNAR, "SOCIAL FORCE MODEL FOR PEDESTRIAN DYNAMICS," (in English), *Physical Review E,* Article vol. 51, no. 5, pp. 4282-4286, MAY 1995 1995, doi: 10.1103/PhysRevE.51.4282.

[42] C. Scholler, V. Aravantinos, F. Lay, and A. Knoll, "What the Constant Velocity Model Can Teach Us About Pedestrian Motion Prediction," (in English), *Ieee Robotics and Automation Letters,* Article vol. 5, no. 2, pp. 1696-1703, APR 2020 2020, doi: 10.1109/LRA.2020.2969925.

[43] R. Korbmacher and A. Tordeux, "Review of Pedestrian Trajectory Prediction Methods: Comparing Deep Learning and Knowledge-Based Approaches," (in English), *Ieee Transactions on Intelligent Transportation Systems,* Review vol. 23, no. 12, pp. 24126-24144, DEC 2022 2022, doi: 10.1109/TITS.2022.3205676.

[44] B. Volz, H. Mielenz, G. Agamennoni, and R. Siegwart, "Feature Relevance Estimation for Learning Pedestrian Behavior at Crosswalks," (in English), *2015 Ieee 18th International Conference on Intelligent Transportation Systems,* Proceedings Paper pp. 854-860, 2015 2015, doi: 10.1109/ITSC.2015.144.

[45] I. Sutskever *et al.*, "Sequence to Sequence Learning with Neural Networks," (in English), *Advances in Neural Information Processing Systems 27 (Nips 2014),* Proceedings Paper vol. 27, 2014 2014.

[46] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," ed, 2014.

[47] A. Kumar, "Demystifying Encoder Decoder Architecture & Neural Network," vol. 2023, ed. Vitalflux: Data Analytics, 2023.

[48] D. E. Rumelhart and J. L. McClelland, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, 1987, pp. 318-362.

[49] M. I. Jordan, "Serial Order: A Parallel Distributed Processing Approach," in *Advances in Psychology*, 1997, ch. 25, pp. 471-495.

[50] J. ELMAN, "FINDING STRUCTURE IN TIME," (in English), *Cognitive Science,* Article vol. 14, no. 2, pp. 179-211, APR-JUN 1990 1990, doi: 10.1207/s15516709cog1402_1.

[51] Y. BENGIO, P. SIMARD, and P. FRASCONI, "LEARNING LONG-TERM DEPENDENCIES WITH GRADIENT DESCENT IS DIFFICULT," (in English), *Ieee Transactions on Neural Networks,* Article vol. 5, no. 2, pp. 157-166, MAR 1994 1994, doi: 10.1109/72.279181.

[52] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," (in English), *International Journal of Uncertainty Fuzziness and Knowledge-Based Systems,* Article vol. 6, no. 2, pp. 107-116, APR 1998 1998, doi: 10.1142/S0218488598000094.

[53] J. F. Kolen and S. C. Kremer, "Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies," in *A Field Guide to Dynamical Recurrent Networks*, 2001, pp. 237-243.

[54] S. Hochreiter and J. Schmidhuber, "Long short-term memory," (in English), *Neural Computation,* Article vol. 9, no. 8, pp. 1735-1780, NOV 15 1997 1997, doi: 10.1162/neco.1997.9.8.1735.

[55] F. Gers, J. Schmidhuber, F. Cummins, and IEE, "Learning to forget: Continual prediction with LSTM," (in English), *Ninth International Conference on Artificial Neural Networks (Icann99), Vols 1 and 2,* Proceedings Paper no. 470, pp. 850-855, 1999 1999.

[56] G. Scholar. "Citations of "Long short-term memory"." https://scholar.google.com/citations?view_op=view_citation&hl=en&user=tvUH3WMAAAAJ&citation_for_view=tvUH3WMAAAAJ:u5HHmVD_uO8C (accessed.
[57] X. Shi *et al.*, "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting," (in English), *Advances in Neural Information Processing Systems 28 (Nips 2015),* Proceedings Paper vol. 28, 2015 2015.
[58] K. Zhou, Y. Zhu, and Y. Zhao, "A Spatio-temporal Deep Architecture for Surveillance Event Detection Based on ConvLSTM," (in English), *2017 Ieee Visual Communications and Image Processing (Vcip),* Proceedings Paper 2017 2017.
[59] Y. Li, "Pedestrian Path Forecasting in Crowd: A Deep Spatio-Temporal Perspective," (in English), *Proceedings of the 2017 Acm Multimedia Conference (Mm'17),* Proceedings Paper pp. 235-243, 2017 2017, doi: 10.1145/3123266.3123287.
[60] T. Browne and C. Eising, "Prediction of pedestrian trajectories using convolutional LSTMs," ed, 2021.
[61] "ConvLSTM Explained," ed: Papers with Code.
[62] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, F. Li, and S. Savarese, "Social LSTM: Human Trajectory Prediction in Crowded Spaces," (in English), *2016 Ieee Conference on Computer Vision and Pattern Recognition (Cvpr),* Proceedings Paper pp. 961-971, 2016 2016, doi: 10.1109/CVPR.2016.110.
[63] H. Manh and G. Alaghband, "Scene-LSTM: A model for human trajectory prediction," ed, 2018.
[64] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," ed, 2014.
[65] A. Sadeghian, A. Alahi, and S. Savarese, "Tracking The Untrackable: Learning to Track Multiple Cues with Long-Term Dependencies," (in English), *2017 Ieee International Conference on Computer Vision (Iccv),* Proceedings Paper pp. 300-311, 2017 2017, doi: 10.1109/ICCV.2017.41.
[66] A. Vemula, K. Muelling, and J. Oh, "Social Attention: Modeling Attention in Human Crowds," (in English), *2018 Ieee International Conference on Robotics and Automation (Icra),* Proceedings Paper pp. 4601-4607, 2018 2018.
[67] I. Goodfellow *et al.*, "Generative Adversarial Nets," (in English), *Advances in Neural Information Processing Systems 27 (Nips 2014),* Proceedings Paper vol. 27, pp. 2672-2680, 2014 2014.
[68] A. Gupta, J. Johnson, F. Li, S. Savarese, and A. Alahi, "Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks," (in English), *2018 Ieee/cvf Conference on Computer Vision and Pattern Recognition (Cvpr),* Proceedings Paper pp. 2255-2264, 2018 2018, doi: 10.1109/CVPR.2018.00240.
[69] S. Eiffert, K. Li, M. Shan, S. Worrall, S. Sukkarieh, and E. Nebot, "Probabilistic Crowd GAN: Multimodal Pedestrian Trajectory Prediction Using a Graph Vehicle-Pedestrian Attention Network," (in English), *Ieee Robotics and Automation Letters,* Article vol. 5, no. 4, pp. 5026-5033, OCT. 2020 2020, doi: 10.1109/LRA.2020.3004324.
[70] A. Sadeghian, V. Kosaraju, N. Hirose, S. Rezatofighi, and S. Savarese, "SoPhie: An Attentive GAN for Predicting Paths Compliant to Social and Physical Constraints," (in English), *2019 Ieee/cvf Conference on Computer Vision and Pattern Recognition (Cvpr 2019),* Proceedings Paper pp. 1349-1358, 2019 2019, doi: 10.1109/CVPR.2019.00144.
[71] A. Vaswani *et al.*, "Attention Is All You Need," (in English), *Advances in Neural Information Processing Systems 30 (Nips 2017),* Proceedings Paper vol. 30, 2017 2017.
[72] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving Language Understanding by Generative Pre-Training," ed, 2018.
[73] OpenAI, "GPT-4 Technical Report," ed, 2023.
[74] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," (in English), *2019 Conference of the North*

American Chapter of the Association For Computational Linguistics: Human Language Technologies (Naacl Hlt 2019), Vol. 1, Proceedings Paper pp. 4171-4186, 2019 2019.

[75] F. Giuliari, I. Hasan, M. Cristani, and F. Galasso, "Transformer Networks for Trajectory Forecasting," (in English), *2020 25th International Conference on Pattern Recognition (Icpr),* Proceedings Paper pp. 10335-10342, 2021 2021, doi: 10.1109/ICPR48806.2021.9412190.

[76] O. Anava, E. Hazan, A. Zeevi, F. Bach, and D. Blei, "Online Time Series Prediction with Missing Data," (in English), *International Conference on Machine Learning, Vol 37,* Proceedings Paper vol. 37, pp. 2191-2199, 2015 2015.

[77] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient Transformers: A Survey," (in English), *Acm Computing Surveys,* Article vol. 55, no. 6, JUN 2023 2023, Art no. ARTN 109, doi: 10.1145/3530811.

[78] Y. LeCun *et al.*, "Backpropagation Applied to Handwritten Zip Code Recognition," (in English), *Neural Computation,* Article vol. 1, no. 4, pp. 541-551, WIN 1989 1989, doi: 10.1162/neco.1989.1.4.541.

[79] datahacker.rs. "#002 CNN Edge detection." https://datahacker.rs/edge-detection/ (accessed.

[80] "Visualization of features learned from face images." ResearchGate. https://www.researchgate.net/figure/Visualization-of-features-learned-from-face-images-a-b-c-are-from-C-SVDDNet-in-3_fig4_303816027 (accessed.

[81] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," ed, 2014.

[82] C. Szegedy *et al.*, "Going Deeper with Convolutions," (in English), *2015 Ieee Conference on Computer Vision and Pattern Recognition (Cvpr),* Proceedings Paper pp. 1-9, 2015 2015, doi: 10.1109/cvpr.2015.7298594.

[83] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," (in English), *2016 Ieee Conference on Computer Vision and Pattern Recognition (Cvpr),* Proceedings Paper pp. 770-778, 2016 2016, doi: 10.1109/CVPR.2016.90.

[84] Y. Wu *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," ed, 2016.

[85] C. Olah, "Understanding LSTM Networks," ed, 2015.

[86] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen Netzen," ed, 1991.

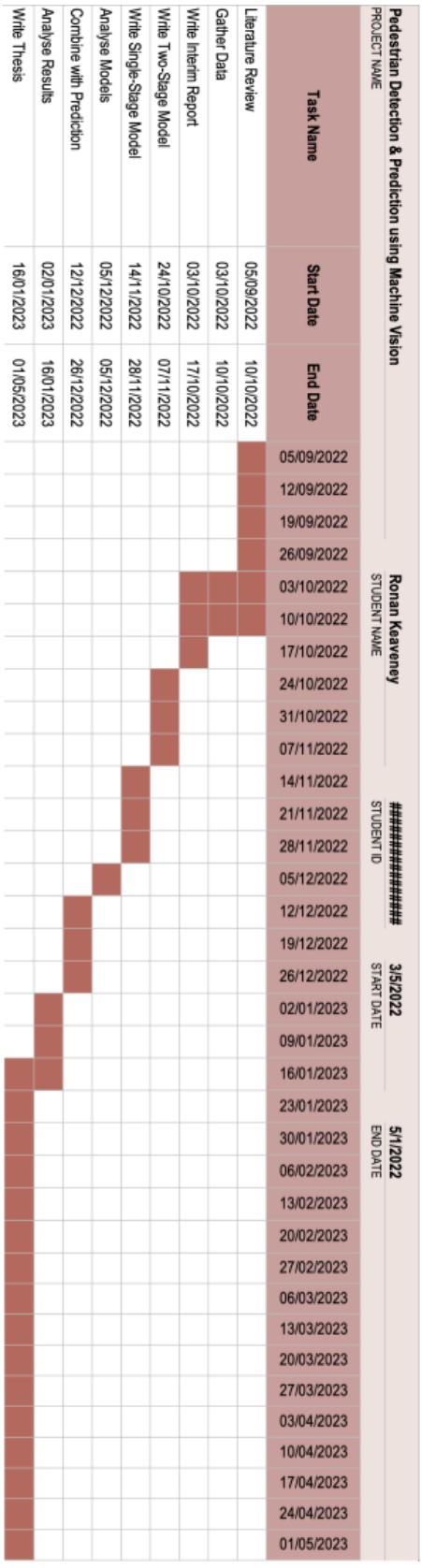
[87] J. Chung, C. Gulchere, H. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," ed, 2014.

[88] Waymo. "Waymo Open Dataset: About." https://waymo.com/open/about (accessed.

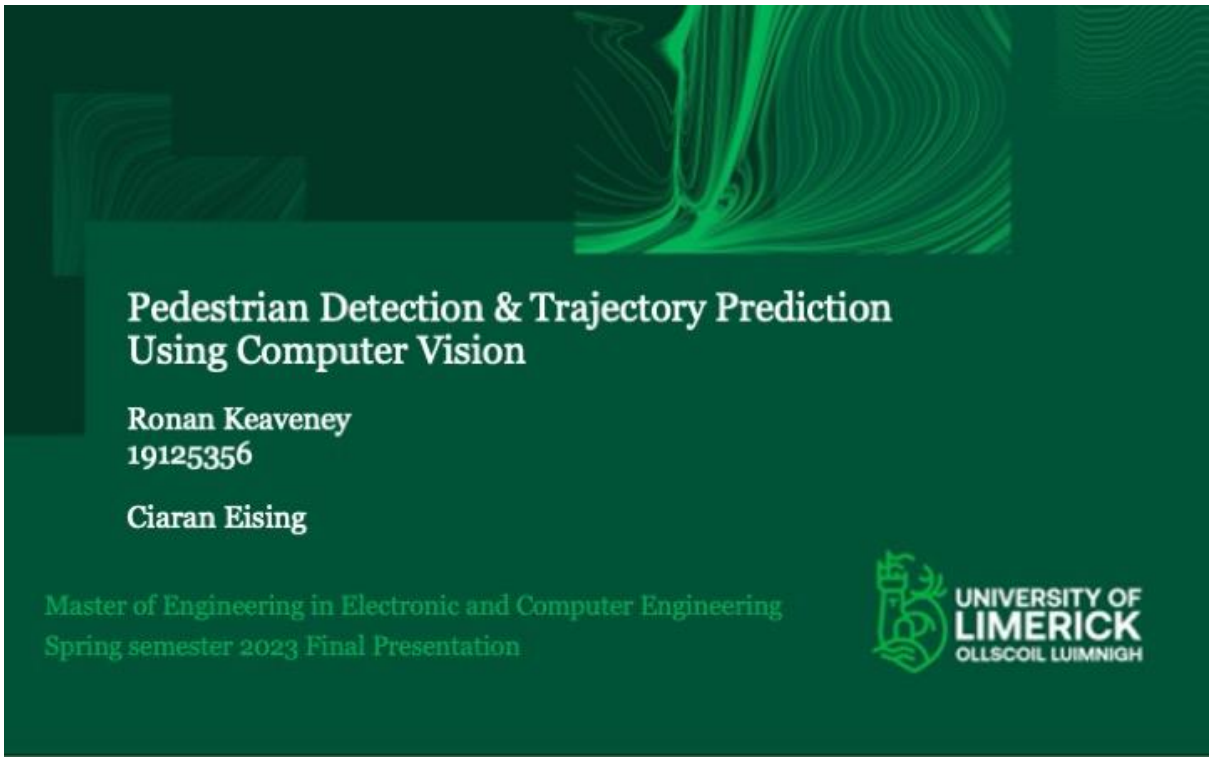
Appendices

- Appendix A: Updated Gantt Chart
- Appendix B: Final Presentation Slides
- Appendix C: Final Poster

Appendix A: Updated project Gantt chart



Appendix B: Final presentation slides



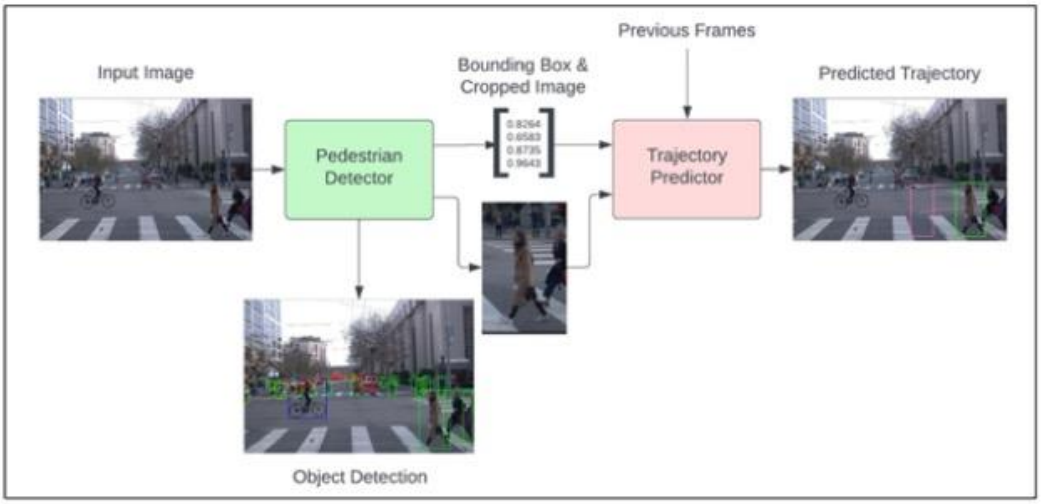
Project Overview

- Over **25%** of road accident fatalities are pedestrians.
- I aim to use computer vision to protect the **most vulnerable** road users.
- 2 components to my solution:
  - Pedestrian **Detection**
  - Pedestrian **Trajectory Prediction**
- Using **only 2D imagery** from a car's front-facing camera from Waymo Open Dataset.

WAYMO

UNIVERSITY OF LIMERICK

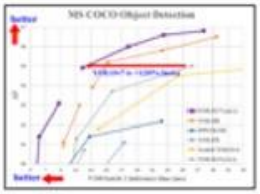
Project Overview



UNIVERSITY OF LIMERICK

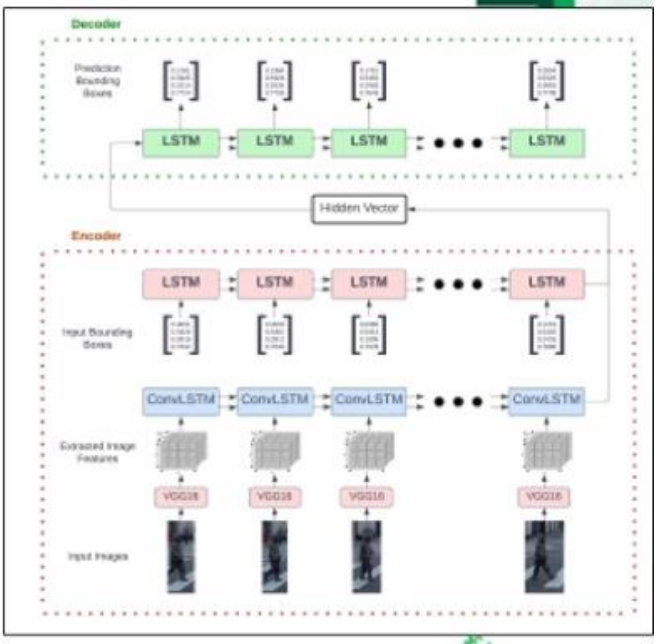
Detector Implementation

- **Detection speed** is critical → single-stage detection
- **YOLOv7**: fastest object detection algorithm
- Trained on over **30,000 images** containing nearly 200,000 pedestrian instances
- **Input:** single frame of 2D video
- **Output:** predicted bounding box of all pedestrians, cyclists, vehicles



Prediction Implementation

- **Encoder-decoder** architecture with LSTMs
- **VGG16** for image feature extraction
- **ConvLSTM** for spatiotemporal dependencies
- **Input**: 10 consecutive frames of pedestrian bounding box co-ordinates and cropped image containing pedestrian
- **Output**: 10 predicted bounding box co-ordinates for pedestrian's future position

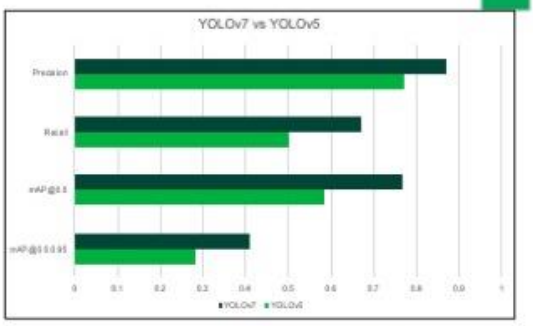


Detector Results

- The pedestrian detector model performed quite well. Here are some summary test statistics.

Precision	0.869	mAP@.5	0.765
Recall	0.670	Inference speed	13.7ms/frame

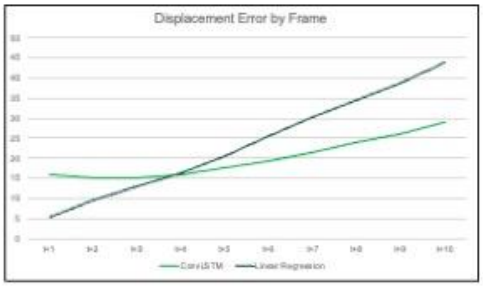
- My model greatly **outperformed** a similar **YOLOv5** model with comparable training and inference speeds.
- On a test set of pedestrians taller than 100 pixels, i.e. within ~40 metres from car, recall jumped to **0.890**.
- Can process live video feed at up to **73 fps.**



UNIVERSITY OF LIMERICK

Predictor Results

- The pedestrian trajectory predictor did not performs as well as I would have liked.
- When car **changed speed or direction**, results deteriorated.
- Just barely outperformed basic **linear regression** model.
- **More input data** for better results:
  - Car speed
  - Car orientation
  - Road map data
  - Radar
  - Lidar



UNIVERSITY OF LIMERICK

Potential uses of the system and future work

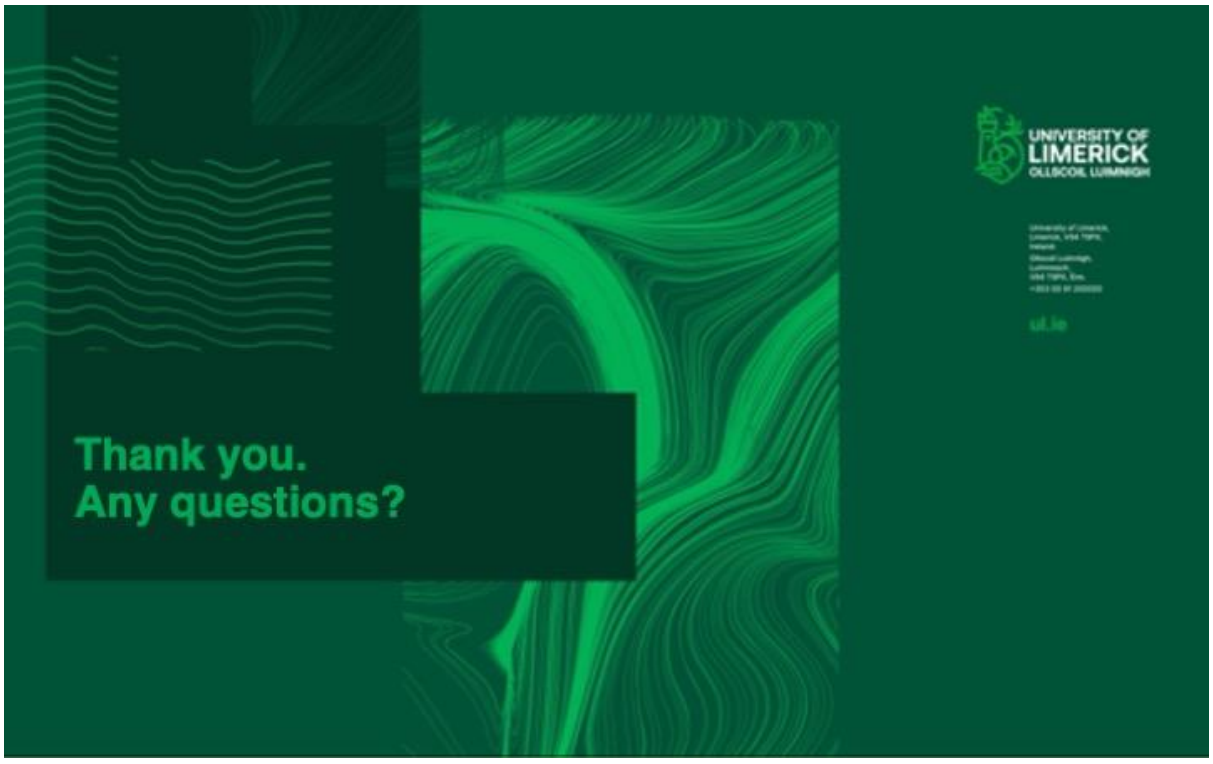
- 2D imagery models should be used **in conjunction** with other detection and prediction techniques.
- To build on this work, one could:
  - Use more input data: speed, direction, road map data
  - Consider transformers instead of LSTMs
  - Account for social forces and scene context

Demonstration

- Example detection of online traffic video. 12-14 ms/frame inference

Conclusions

- Pedestrian prediction is hard!
- 2D imagery is part of, but not the whole solution.
- The future is bright for pedestrian detection and trajectory prediction.



Appendix C: Project poster

Pedestrian Detection & Trajectory Prediction Using Computer Vision

E&CE Department of Electronic and Computer Engineering

Ronan Keaveney

ME Electronic & Computer Engineering

Introduction

In recent years, advances in autonomous driving and advance driver-assistance systems (ADAS) have helped to make our roads a safer place. One of the most important and challenging tasks for this industry is to create a system which can effectively detect pedestrians in the field of view of the vehicle and accurately predict their intentions and trajectory. While this ability comes very naturally to humans, who evolved to be excellent at object detection and tracking, it has proven quite challenging to train a machine to perform the same task effectively.

A crowded urban scene from Waymo Open Dataset

In this project, I explore the potential for using computer vision and AI techniques such as CNNs and convolutional LSTMs to detect and predict the trajectory of pedestrians using 2D video imagery from a car's on-board camera.

Aims

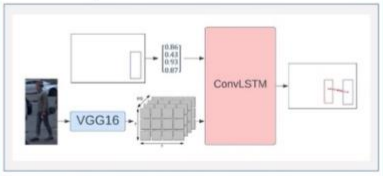
The main aims of this project are to investigate some of the approaches that have been used in the past for pedestrian **detection** and **trajectory prediction**, before developing my own models using the YOLOv7 framework and convolutional LSTM networks.

Method

Here's an outline of the methodology I followed:

- Researched the fields of object detection & trajectory pred iction, before deciding on the approaches that I would take for my project.
- Gather data from Waymo Open Dataset
- Process data: extract photos, bounding box data, object classes.
- Get list of all pedestrians that are present in 20 consecutive frames of video (2 seconds), for trajectory prediction model.
- Developed pedestrian detection using YOLOv7 framework.
- Developed pedestrian trajectory prediction model using pretrained VGG16 feature extraction model and convolutional LSTM network.
- Evaluated performance of both models and visualised data predictions.

For my pedestrian detection model, I decided to go with YOLOv7, a **single-stage detector**, as opposed to a two stage-detector, such as R-CNN. YOLOv7 performs object proposal, classification and localisation all in a single stage and a single network. This results in **detection speeds** much faster than those seen in two-stage models, which is critical for the application of real-time pedestrian detection.



Convolution LSTM model pipeline

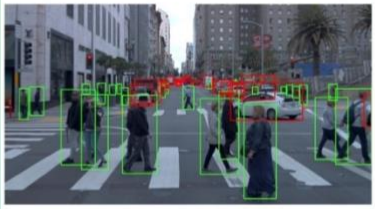
For the trajectory prediction model, I used a convolutional LSTM network. The input to this network was a **10 frame sequence** of bounding box data for the pedestrians location in the image, as well as a fixed-length matrix which contains information on the features extracted from the image of the pedestrian, using a VGG16 pretrained model. The prediction model then outputs predicted bounding box for the next 10 frames in the future.

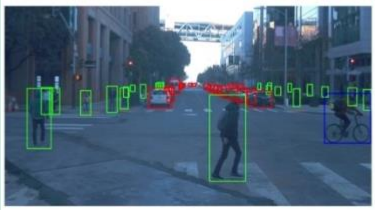
Results

The pedestrian detection model performed quite well. Here are some performance statistics on my test set of 3923 images, followed by some example predictions:

Precision	0.869	mAP@0.5	0.765
Recall	0.670	mAP@0.5:.95	0.410

Also, when I narrowed my prediction down to only pedestrians taller than 100 pixels, i.e. within ~40 metres from car, the recall jumped to **0.890**.





Sample outputs from pedestrian detection model

The trajectory prediction model also produced decent results. Here are some example results:

However, the results greatly deteriorated when the car changed speed or direction, as seen in this photo, where the car approaches a bend.

To build further on this model, I would include data for the speed and orientation of the car in my input data. Both of these datapoints were provided in the Waymo Open Dataset.

Conclusion and personal reflection

In this work, the following goals were achieved:

- Gained a deep understanding of techniques used in object detection & tracking.
- Developed a single-stage pedestrian detection model using the YOLO framework
- Developed a pedestrian trajectory prediction model using a convolutional LSTM network.

If I were to start this project over again, I would:

- Consider using a transformer network instead of convolutional LSTMs.
- Use more datapoints for the prediction model, such as speed, orientation and perhaps some radar and lidar data.

Acknowledgements

I'd like to express my immense gratitude to my supervisor Ciaran Eising, PhD student Daniel Jakab, and past UL Master's student Tommy Browne. Without their knowledge, generosity & patience, this project would not be possible.

UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH