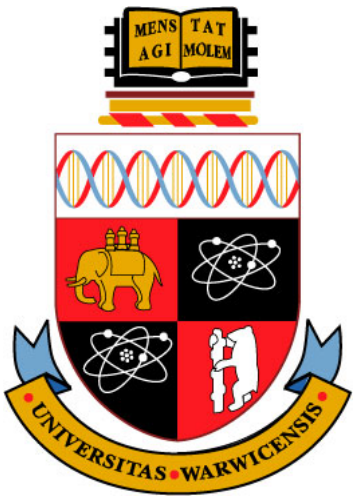


Classification of Pedestrian Activity via Computer Vision
CS310: Final Report

Jan Piotr Piskorski

October 2021 – May 2022

Supervisor: prof. Theodoros Damoulas
External co-supervisor: James Walsh
Department of Computer Science
University of Warwick

Keywords— Convolution, Activity Recognition, Deep Learning, Dataset, Framework, Modularity, Neural Network, Real-Time

Abstract

In this report we present our undergraduate project, which deals with researching and developing a human activity detection framework. We first conduct a brief review of the existing practices in the field. We then explain our modular framework and the artificial neural network models we developed for the final step of it – activity recognition. We show that our solution can achieve promising results at over 70% accuracy while working with near–real-time performance. We conclude the report by presenting the results and analysing ethical concerns connected with the project.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction and Motivation

Our project is concerned with labelling pedestrian activity. In order to do so, video footage is analysed in the way that takes into account not only individual frames, but how the image changes through time. Therefore, activity recognition presented in this project is described as *temporal*.

We undertook this project without any prior experience with deep learning or computer vision, following the departure of our initial project supervisor from the University in September 2021. Following a discussion with the new supervisor we decided to undertake the project described in this report. We initially stated the project objectives as follows:

- Gaining appreciation of machine learning–assisted computer vision methods
- Recognising a set of behaviours of pedestrians from video footage
- Extending the existing image processing solutions by augmenting object detection with analysis of the temporal aspect of video footage
- Incorporating existing solutions, such as road boundary detection or existing object detection methods into the project

We also stated these optional goals:

- Including the derived data into other areas of the Odysseus Project, for example into Air Quality models as covariate information
- Learning a mapping between air quality and local urban environment using air quality sensors situated close to the cameras and then extrapolating the results into other crossings covered by the cameras, but with no air quality sensors in proximity

These objectives were inspired by two projects from the Alan Turing Institute: London Air Quality Project [5] and Project Odysseus [6]. As the

project was taking its current shape and form, we had to reconsider some of the project aims. For example, since we decided that we want to develop a framework that can work independently of the projects it was inspired by, we removed the objective of incorporating the existing solutions specific to those projects. We discarded optional goals and defined the final project aims as follows:

- Develop a piece of software capable of labelling pedestrian activity in provided video footage;
- The labelling of pedestrian activity should be done in real time or near–real time;
- Gain experience in research and development of deep learning models.

1.1 Report outline

The purpose of this report is to describe the process of researching and developing our own activity detection framework, to outline the field and the developed framework itself and to present and evaluate the results of the work done and propose future extensions and developments. In the rest of this chapter we describe the projects undertaken by our supervisors which inspired and motivated this project. Then, in chapter 2, we present a brief review of existing human activity recognition and detection methods, followed by applicable techniques used in the field. In section 2.3 we describe a classifier neural network that we developed to gain necessary working knowledge of deep learning. Finally, in section 2.4 we explore in depth the models and algorithms which are used further in the project.

In chapter 3 we describe how we carried out the planning and progress monitoring of the project. We then introduce the modular architecture of our activity detection framework and design choices. This is followed by description of software tools and libraries as well as hardware used for the project.

In chapter 4 we describe how the modular architecture of our activity detection framework was implemented. We then describe activity recognition neural network that we implemented. We conclude the chapter with the description of how we handle datasets and training and presentation of CCTV dataset that we created for the purposes of evaluation.

The evaluation itself is described in chapter 5. We present results of our work by discussing accuracy and performance of the framework that we developed. We then analyse limitations of our solution and suggest future work that can be carried out in connection to this project.

We conclude our report in the final chapter, chapter 6. There, we first focus on ethical and privacy considerations that are relevant to the project.

We analyse a common example of unplanned data deanonymisation and then describe how our privacy considerations changed as the project developed. In an effort to self-assess our work on this project, we compare the initial and final project plan and mention challenges faced and applied mitigations. The chapter ends with acknowledgements.

Let us now introduce Alan Turing Institute projects which served as an inspiration for this work.

1.2 London Air Quality Project

The aim of London Air Quality project is to understand and aid improving air quality in London. It takes air quality sensor data and combines them with various other data sources to more accurately estimate local air quality and provide forecasts. One of the use cases is finding low-pollution paths for cycling, walking or exercising. The overarching goal is to inform the people and aid policy-makers in making more informed decisions [5].

As far as the scope of our project is concerned, the significance of London Air Quality Project lies within the fact that Project Odysseus, described below, is based upon the same foundations.

1.3 Project Odysseus

Project Odysseus is undertaken at the Alan Turing Institute as an aid for policymakers in making informed decisions on tackling the coronavirus crisis. Its aim is to aid exiting lockdown in a manner that is principled and data-driven [6].

Crucial part of Project Odysseus is a computer vision deep learning pipeline, the purpose of which is to provide near-real time data: pedestrian footfall and distancing, number and type of vehicles. The data used is sourced from London traffic cameras, provided through the JamCams system. The pipeline tackles the problems of camera stability, group detection and estimates social distancing between pedestrians across over 500 intersections in London [21].

1.4 Motivation – need for temporal activity recognition

The aim of this project, Classification of Pedestrian Activity via Computer Vision, was initially motivated by the computer vision and machine learning pipelines described above. The pipeline does not currently take into account the temporal aspect of the analysed footage. Instead, a number

of frames is chosen from the video footage provided, and these frames are analysed independently. Introduction of temporal aspect to the analysis of video footage could be used to label pedestrian activity, such as loitering or walking, with greater accuracy, which can be of use with regards to coronavirus measures, as well as activity beyond lockdown analysis, but within the scope of interest of the London Air Quality project and policymakers. For instance, it can be used to analyse the flow of pedestrians in crowded spaces or road safety.

While initially the project was conceived as an element of Project Odysseus or London Air Quality Project infrastructure, it evolved to be an independent modular action detection framework. In section 5.3 we explore how the developed framework can be integrated into these Alan Turing Institute projects.

Chapter 2

Background

At the beginning of the project, author's background was minimal. Therefore, it was necessary to gain working knowledge in deep learning, computer vision, toolsets and best practices.

2.1 Human activity recognition methods – a brief review

There exists a significant number of human activity recognition methods. They differ by input data available for classification, approach to performing the classification and type of human activities that they focus on [20]. Human activity recognition can be divided into the following categories:

- *Unimodal* – using a single sensor:
  - *Space-time* – activities are represented as a set of spatio-temporal features and trajectories
  - *Stochastic* – statistical models are used to represent human actions, for example Hidden Markov Models
  - *Rule-based* – a set of predefined rules is used to categorise and label the activity appropriately
  - *Shape-based* – involves motion modelling of human body parts
- *Multimodal* – combines features from different sources (sensors):
  - *Affective* – attempts to label activities based on affective states and emotional communication
  - *Behavioural* – utilises gestures, expressions and auditory cues
  - *Social networking* – focuses on human-to-human interactions

Additionally, it is important to note that *detection* deals with all instances of object activities in the frame, while *recognition* deals with one instance at a time.

As this project relies exclusively on video footage for input, the categories that the analysis should be limited to are unimodal: space-time, stochastic and rule-based. Since modelling of human body parts requires relatively high fidelity video, which cannot always be guaranteed in the case of this project, Shape-based activity recognition has also been discarded. Below, we present some existing human activity classification methods as examples.

2.1.1 The classic spatio-temporal unimodal method – optical flow of human figure

This method, in its earliest form, was presented in 2003 by Efros et al. [3]. It relies on optical flow of human figure. It delivers satisfying results in noisy, low-resolution video and its implementation is relatively simple. While its performance is not on par with current state of the art [7], it can serve as a good basis for improved recognition methods based on similar principles.

Another solution that is based on the method described above is discussed in detail in [13]. It uses clustering, which is time consuming. The authors of the original paper claim, that even despite optimisation with random subsampling, the clustering algorithm is not suitable for real-time classification. When incorporating features of the model described in this paper, care needs to be taken to assess the trade-offs between time and result.

Finally, [8] describes methods of representing optical flow fields in a simpler, compressed way: as four heat maps, each denoting how strong the up/down/left/right element of the optical flow vector is. It also mentions creating binary trees to quickly recognise extensive sets of categories, which is not needed for this project, as we are considering cases with limited number of activity labels. GrabCut is a background segmentation algorithm, which can be useful if the model overfits on background. Compression of optical flow fields is also of note.

2.1.2 Trajectory Learning

This method [12] relies on online, real-time analysis of surveillance video footage using Hidden Markov Models to encode dynamics of activities and connecting the routes formed through trajectory clustering.

The method described in this paper could be partially relevant to the project. It analyses human activity based not on the appearance of the

subject, but by the trajectory of their movement. It also takes into account the predictions and anomalies – so that the trajectories of persons' movements can be predicted and unexpected behaviour detected. Therefore, the methods described in this work can be of most use when detecting whether a person crossing the street is doing so in a safe manner.

2.1.3 Deep learning

The following method, described in [9], uses Convolutional Neural Networks to learn representations of identifying data features. According to Le et al. it outperforms the methods above, while giving the possibility of training on unlabelled data. Also, the general idea of using Convolutional Neural Networks as opposed to recognising hand-crafted features would be worth investigating.

2.1.4 Summary

It is important to note that this section only serves as a brief overview of human activity recognition methods. A comprehensive review would constitute a substantial body of work and could form a project of its own. Therefore, for a more detailed and comprehensive review and comparison please refer to these enclosed articles: [20], [11].

2.2 Applicable methods used in Human Activity Recognition

Thanks to background reading, it was also possible to identify and learn some relevant techniques, notably Intersection of Union, which measures similarity of bounding boxes.

2.2.1 Intersection of Union

Intersection of Union (also known as Intersection over Union, commonly abbreviated as IoU) is a ratio that measures similarity of two bounding boxes with regards to their position and shape. It is calculated using the following formula:

$$IoU_{A,B} = \frac{Area(A \cap B)}{Area(A \cup B)}$$

where A and B denote bounding boxes. It is easy to see that the values of the IoU ratio can be between 1, for two identical bounding boxes, through fractional values for partially overlapping bounding boxes, to 0 for non-overlapping boxes.

Intersection of Union sees many uses in computer vision. Within the scope of this project, it is primarily used by SORT tracking algorithm, described in section 2.4.2 below. It is also extensively used as one of the accuracy metrics in evaluating object detection frameworks.

2.3 Practical experiments with deep learning

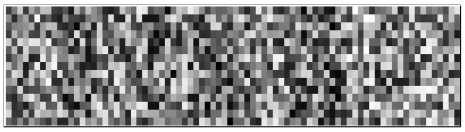
```
Net(
(conv1): Conv2d(3, 16, kernel_size=(5, 5), stride=(1, 1))
(pool): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
(conv2): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1))
(dropout): Dropout(p=0.2, inplace=False)
(fc1): Linear(in_features=89888, out_features=256, bias=True)
(fc2): Linear(in_features=256, out_features=84, bias=True)
(fc3): Linear(in_features=84, out_features=2, bias=True)
(softmax): LogSoftmax(dim=1)
)
```

Figure 2.1: Structure of the binary CNN classifier

In the course of studying basic deep learning methods, as a companion to background reading and literature review, a binary classifier has been developed in order to test freshly learned ideas. The model is a simple convolutional neural network. Its structure consists of two layers of convolutions connected by max pooling, followed by three layers of fully connected MLP. The structure of the model is presented in figure 2.1.

The data used to train the model consisted of footage from different cameras in London, like the data used in the machine learning/computer vision pipeline of the Odysseus project. Two different datasets were used for training different instances of the model:

Training set 1 consisted of data from ten different cameras in the positive set, making up 13% of the entire dataset. As it turned out, this neural network is not capable of classifying images from multiple sensors as positive because of its simple nature. Feeding the network with different positive images confuses the weights. The outcome is, no matter the number of training epochs, of a very similar accuracy (difference within 3%): accuracy of classifying negative set is at 100%, while the accuracy of classifying an image as a member of positive set is below 13% – that is, random. Example accuracy for 400 epochs:



(a) Filter plot

(b) Example image in the Positive set

(c) Positive set element from different sensor

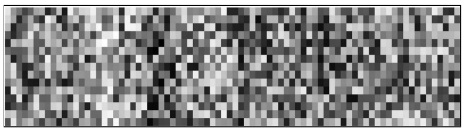
(d) Negative set member

Figure 2.2: Data used in the first dataset for the binary CNN classifier

```
    Accuracy of 0 is 100.0
Accuracy of 1 is 10.204081632653061
Overall accuracy 80.35714285714286
```

Training Set 2 consisted of data from just three sensors. Images from one sensor have been labelled as positive, the remaining 2 as negative. This training set has been prepared following the analysis of the first training set's lack of performance. Two versions of the set have been prepared: a bigger one, consisting of over 24000 images in total, and a smaller one, with around 8000 images. Due to constrained available computational power, it was not feasible to train the model on more than 10 epochs and 50 epochs for bigger and smaller version of the set respectively. The results, however, were impressive: The overall accuracy of the model trained on the bigger set was 99.81% and on the smaller set – 99.57%. Accuracy for bigger dataset:

```
Accuracy of 0 is 100.0
```



(a) Filter plot

(b) Positive set member

(c) Negative set member example

(d) The other sensor in the negative set

Figure 2.3: Data used in the second dataset for the binary CNN classifier

```
Accuracy of 1 is 99.4047619047619
Overall accuracy 99.8143564356435
```

Accuracy for smaller dataset:

```
Accuracy of 0 is 100.0
Accuracy of 1 is 98.60279441117764
Overall accuracy 99.56683168316832
```

2.4 Applicable models and algorithms

In this section we will perform a more detailed description of algorithms that will be relevant later in the report.

2.4.1 Object Detection with YOLO

YOLO, an abbreviation of You Only Look Once, is an object detection framework. It has seen several revisions, notably YOLOv2 [14], v3 [15] and v4 [2]. The aforementioned versions 2, 3 and 4 are fully convolutional neural networks, which means that they do not rely on fully connected layers for classification, instead using feature maps and anchor boxes [14]. As the name of the framework suggests, the idea behind it is to perform the detection in a single step, as opposed to methods in the style of R-CNN, which perform detection in a multi-step process. The following description is true for YOLOv3, unless specified otherwise.

The goal of object detection is to find and identify objects from a set of predefined object classes. YOLO finds an object by locating it on an image and returning the bounding box around it. The network is fully convolutional, so output is generated for each element of a feature map. The output feature map is downsampled by a factor of 32 by convolutional layers. In YOLOv3, as opposed to v2, the output feature map is then upsampled twice to extract additional features at different scales.

So at each scale, the output feature map predicts a fixed number (3 according to an example in the article [15]) of potential bounding boxes with detected objects inside per feature map element. In order to increase stability of the network, the bounding boxes are predicted by the feature map element located in the centre of the bounding box. Moreover, the bounding box sizes are predicted as offsets to predefined anchor boxes (also called priors). Anchor boxes are bounding boxes of predefined size and dimension, hand-picked for each training dataset. To pick the optimal predefined prior sizes, k-means clustering is used.

The shape of the output of YOLO predictions is as follows:

$$[t_x, t_y, t_w, t_h, t_0, t_1, t_2, \cdots, t_n]$$

where t_x, t_y, t_w, t_h correspond to the bounding box position and dimensions, t_0 is the *objectness score* and $t_1, \cdots, t_n$ are probabilities that a potential detected object belongs to a given class. Each element of the feature map outputs 3 (again, as described in the YOLOv3 article) such sets of data. Of note is the objectness score. It denotes the probability that the bounding box contains an object. Since each feature map element outputs a fixed number of bounding boxes, some are bound to be empty.

The description above also applies to YOLOv4, which introduces improvements to the training and recognition process. The authors of v4 call them *Bags of Freebies* and *Bags of Goodies*, which correspond to techniques which only influence training time while improving performance and techniques which only slightly influence detection time while greatly improving performance [2]. Overall, YOLO recognises objects with state-of-the-art

performance while achieving real-time speeds. To achieve it, it uses fully convolutional neural network, with feature maps predicting bounding boxes as offsets to priors.

2.4.2 Object Tracking with SORT

Simple Online and Realtime Tracking, abbreviated as SORT, is a lean object tracking method based on classical algorithms: Kalman filter, intersection of union and Hungarian algorithm [1]. The purpose of tracking is to assign a unique identifier to each object in order to identify it across frames of video footage. SORT is an online tracker, which means that it only takes into account earlier frames in order to obtain results for the current frame. This is in opposition to batch trackers, which obtain results for a given frame based on frames both from the past and future.

SORT assumes constant velocity of tracked objects and constant aspect ratio of their bounding boxes. For each frame, it first predicts new positions for bounding boxes of tracked objects based on their estimated velocity. Then, it uses intersection of union, described in section 2.2.1, to match predicted bounding boxes with actual object positions on the new frame. In case there are multiple overlapping bounding boxes, assignment is solved optimally using Hungarian algorithm. There is also a threshold for minimal IoU overlap, below which tracking identity is discarded. This is done in order to minimise the number of false positives.

In order to estimate the position of tracked objects, Kalman filter is employed. In order to estimate position at time t, the following recursive formula is used:

$$E_t = E_{t-1} + K_G(E_p - E_{t-1})$$

where E_p is the position based on constant velocity estimation, E_{t-1} is the position at previous frame and K_G is the Kalman ratio. Kalman ratio is calculated as follows:

$$K_G = \frac{E_p}{E_p + E_m}$$

Where E_m is the position based purely on measurement, in case of SORT, position of the matched bounding box. Objects that newly entered the field of view for tracking are initialised with $E_p = 0$ in order to account for the fact that initially the velocity of the new object is unknown.

Overall, the SORT method, outlined above, offers performance comparable to other state-of-the-art methods, while being online and significantly faster, at roughly 260 frames per second [1].

Chapter 3

Methodology and Architecture

In this chapter we will discuss the approach and toolset chosen for the project. We will start by outlining the methodology which was chosen to aid day-to-day and week-to-week planning and generally make the work structured. Afterwards, we will describe the overarching design and each component of software architecture which forms the basis of this project. This will be followed by justification of the chosen toolset and a brief description of hardware used for implementation and testing.

3.1 Methodology

From the beginning it has been agreed that weekly meetings should be organised. These meetings served the purposes of tutoring, monitoring and planning. They usually included questions and discussion on progress and challenges or background reading, especially in the earlier phases of the project. We would also discuss progress and agree on a plan for the following week.

While we have not explicitly chosen nor discussed choosing to adhere to any particular widely known named project management methodology, the methodology can be with confidence called "iterative". This is due to the structure and content of weekly meetings. Therefore the time between the meetings would constitute an informal "sprint".

As we have already mentioned in the introduction in chapter 1, we have started this project with no prior experience with machine learning, deep learning or computer vision. Because of that, substantial amount of time had to be accounted for for background reading, gaining working knowledge and crucial experience of deep learning, followed by familiarisation with tools and methods standard in modern computer vision research.

3.2 Architecture

In this section we will discuss the software architecture which forms the basis of the software part of the project. From now on we will address it as "framework". First, we will focus on its modular design, followed by describing the purpose of each module.

3.2.1 Modularity

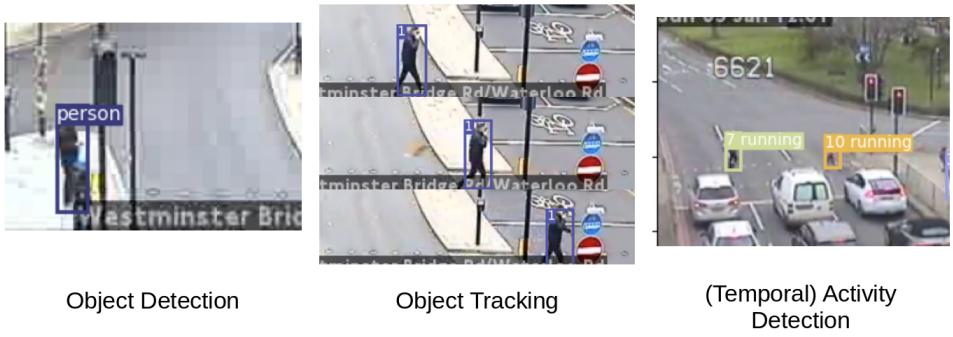


Figure 3.1: Modular design of the activity detection framework

Let us first briefly consider some of the project's main objectives:

- It has to be a recognition system, which means that it has to detect all occurrences of a given set of actions in the given footage;
- The recognition model has to take into account the temporal dimension;

A bit of thought on the objectives above leads to the following:

- In order to recognise an action, we need to identify an agent which can perform a given action, for example a person;
- If the object recognition above is performed on a set of static images, namely frames of video footage, we need a way to track objects, so as to identify an agent which performs given action across frames.
- The data available for activity recognition are the footage, either individual frames or as a whole, dimensions and position of the bounding boxes around the objects (agents) recognised and the position over time, that is *movement of the bounding boxes.*

Following this analysis, we decided to design a modular three-step architecture, a data pipeline for videos. The modules are Object detection, Object tracking and Activity recognition or detection. The purpose of each of the modules maps directly with the analysis above: Object detection handles identifying agents, which perform actions to be recognised, for example people; Object tracking ensures that the agents are identified across frames. Finally, activities can be assigned an action label by the activity recognition model. This design makes it straightforward to experiment with different object detection models or tracking algorithms.

3.3 Software and tools

As far as Deep Learning is concerned, Python is a de-facto standard programming language. It is used by most of research mentioned in this report. Python version used during development of the project is 3.9.7. Notable Python frameworks, tools and libraries used in this project include:

- Jupyter 4.9.2 – Jupyter, more notably Jupyter Notebook, is a tool which allows literate programming and evaluation of snippets of code. It was useful for prototyping new elements of the framework and carrying out experiments. In the final project Jupyter notebook files are used instead of plain Python where it is useful to visualise progress, see intermediate results next to applicable code or plot figures.

- Matplotlib 3.5.1 – This package, mainly its library pyplot, is used for plotting to visualise loss and accuracy during training and together with seaborn for illustrating models performance for evaluation.

- NumPy 1.21.5 – Numpy arrays are de-facto standard for exchanging complex multi-dimensional data between libraries, namely Pandas, PyTorch, Pillow and Scikit. NumPy's internals are written in C, which makes it faster than standard native Python data.

- OpenCV 4.5.5 – OpenCV is a computer vision library. It is used for conversion between colour (three-channel) and black-and-white (single-channel) image data, saving cropped video clips in dataset preparation and reading video footage from files.

- Pandas 1.4.1 – Used for loading CSV files for dataset processing.

- Pillow 9.0.1 – Pillow is a fork of PIL, Python Imaging Library. It is mainly used for image data conversion between NumPy and OpenCV.

- pip 20.3.4 – pip is the package manager for Python. There exists a different package manager designed with complex projects and data

analysis/machine learning in mind. It is called Conda. During early stages of development Conda was used instead of pip. However, pip was finally chosen due to the fact that Conda was slow and could not resolve some combinations of versions of dependencies which pip handled without any issues. Also, author's personal preference is lean solutions and pip is arguably leaner than Conda.

- PyTorch 1.10.2 – PyTorch is a Deep Learning library, which allows building neural network models, performing complex computation tasks on GPU.
- Scikit-learn 1.7.3 – Scikit is used mainly for calculation of model evaluation metrics.
- seaborn 0.11.2 – This package is essentially an extension to Matplotlib which aids plotting more complex visualisations, such as confusion matrices.
- tqdm 4.62.3 – The purpose of this package is mainly informative and decorational. It displays a progress bar for dataset processing and during training and evaluation of the model. It also estimates time elapsed.

We use Git for code version control. A private Github repository has been created and used for backups. Due to substantial size of some datasets, the backups on Github are not comprehensive. We are aware that this poses a serious risk in case of hardware failure, so an additional local backup is stored on a USB hard drive.

Internal project notes were taken using Markdown, which is a simple markup language. It can be converted into LaTeX using tools such as Pandoc. This allowed keeping all project data in one place. The author is used to note taking in physical notebooks, which could have caused problems recreating the research and development process for this report.

3.3.1 Choice of object detection method

Object recognition methods considered for this project included different revisions of YOLO [14] [15] [2] and Faster R-CNN [16]. The main difference between YOLO and R-CNN is the approach to object detection: YOLO performs everything as a single run of a convolutional neural network, focusing on performance, while Faster R-CNN uses a two-step approach, aiming to maximise accuracy. While both methods claim to be real-time or almost real-time, their performance and results differ significantly. Faster-RCNN claims speeds of approximately 5 frames per second. YOLO, depending on the version, claims from slightly under 30 to over

50 frames per second. Comparison of backbones, discussed in [15], shows that YOLOv2's Darknet-19 has the most time-efficient backbone at 171 fps, YOLOv3's Darknet-53 follows at 78 fps, while R-CNN's ResNet-101 achieves 53 fps. We opted for YOLOv3. While it achieves less accurate results than YOLOv4, it is significantly faster than v4 and R-CNN and the results of v3 are much better than those of YOLOv2. It has been noted in [1] that tracking performance strongly depends on detection accuracy, so speed and accuracy have to be balanced to achieve both high quality predictions and satisfactory near–real time performance. Technically YOLOv4 at 23 to 38 fps could achieve higher quality predictions than chosen YOLOv3, but we need to take into account that object detection is not the only model taking computation time in this framework.

3.3.2 Choice of object tracking method

The object tracking method was chosen based on the Multiple Object Tracking challenge (MOT challenge) [10]. Its results show that the majority of well-performing trackers cannot be used in real-time environments. This is due to the fact that their performance is in most cases a mere couple of frames per second and the "batch" nature of the trackers: while calculating the tracking identifiers of the objects, they take into account frames both in the past and in the future relative to the frame currently processed. SORT, that is Simple Online and Realtime Tracker, has been chosen for several reasons. First of all, its performance is state-of-the-art at 260 frames per second. Secondly, it is an online tracker, so it calculates results for a given frame only based on the frames from the past. Finally, it is a lean solution based on classical algorithms, which matches our personal preference. There exist various modifications of SORT, such as DeepSORT [23], which sacrifice some performance to reduce the number of identity switches. For our purpose, identity switches are not a problem since, as we will describe soon, we need to track the object for at most 60 frames anyway.

It is worth noting that the framework is modular and, as we will describe in chapter 4, it is relatively straightforward to adapt object detection and tracking methods to work with the framework. Therefore, we would like to think of the object detection and tracking methods chosen as examples rather than definitive, firm, unchangeable choices.

3.4 Hardware used

Hardware used for development and tests of the entire framework and activity recognition model in particular was a laptop with 16GB of RAM and GeForce 3050 RTX mobile graphics card with 4GB of Video RAM.

Chapter 4

Network description and Implementation

In this chapter we will explain the implementation of the entire activity detection framework. We will then focus on the implementation of the activity recognition models and handling of data.

4.1 Architecture Implementation

In this section we will focus on implementation of the modular activity detection framework.

4.1.1 The main driver code

The main file of the framework, which performs the detection on a video, is called `main_detect.py`. Due to wrappers, described below, it is a relatively short and simple algorithm (see Algorithm 1).

Algorithm 1 The Main detection of the framework

$Video \leftarrow$ the video file
$Frame \leftarrow$ the first frame of $Video$
$Tracks \leftarrow$ an empty list
while $Video$ is not at last frame **do**
    $Detections \leftarrow$ ObjectRecognitionWrapper($Frame$)
    $Track \leftarrow$ ObjectTrackingWrapper($Frame, Detections$)
    $Tracks \leftarrow Tracks \cup \{Track\}$
    $Frame \leftarrow$ the next frame of $Video$
end while
$Actions \leftarrow$ ActivityRecognitionWrapper($Video, Tracks$)
Draw and output $Actions$

The algorithm first loads needed models into memory, then proceeds with the processing. It iterates through the frames of the video to recognise people on each frame. The recognised bounding box data is then passed through a tracking algorithm to identify the bounding boxes across frames. The intermediate output of the tracking algorithm for each frame is saved for future use. Finally, activity recognition is performed. The wrapper is provided with raw video footage and tracking data and handles data preparation. At the end, the results are saved or displayed.

4.1.2 Object Recognition wrapper

The Object Recognition wrapper object has to implement the following methods:

- `__init__()` – the class constructor. The constructor should initialise and load Object Recognition model into memory. It can have optional arguments if needed. They should be optional and provided with defaults, though.
- `detect(image)` – This method accepts one argument, `image`, which represents an image, probably a frame of video footage, in NumPy array format. It returns a NumPy array of detections.
- `display(image, detections)` – This method accepts two arguments: image in matplotlib-compatible format, for example as a NumPy array, and detections, that is a list of detections, probably the output of the `detect(image)` method. It displays the image and plots the bounding boxes with captions on the image. This method is only used for testing purposes, it is not necessary to run `main_detect.py`.

In the code provided, the file `yolo3wrapper.py` implements Object Recognition wrapper for YOLOv3. `detect(image)` returns detections in a NumPy array containing a list of detections. Each detection is of the following format:

$$[x_1, y_1, x_2, y_2, c, cls]$$

where x_1, y_1, x_2, y_2 are bounding box coordinates, c is detection confidence and cls is the class identifier. Class identifier is important since YOLOv3 used in this example is pretrained on the entire COCO dataset. It is therefore important to filter out any detected objects which are not people. Detection also discards bounding boxes of size below a certain threshold. By default this has been set to the width of 4 and height of 6, but this can be changed by passing optional arguments to the `__init__()` method. This is due to the fact that at such resolution it is unlikely to get any meaningful activity labels.

4.1.3 Object Tracking wrapper

The object tracking wrapper has to implement the following methods:

- `__init__(classes_to_track, video)` – the class constructor. The constructor should initialise, set up and load the model. It must accept two arguments. The first one, `classes_to_track`, is an array of integer object class identifiers. The Object Tracking will only track recognised objects which are of a class included in this array. The second argument, `video`, is a video object, described in subsection 4.1.4 below.

- `track(image, detections)` – The purpose of this method is to update the tracker with a new frame. It accepts two arguments: `image` – an image in the format of a NumPy array, probably the new frame of the analysed video footage; `detections` – output of `detect(image)` method from the Object Recognition wrapper. It returns a NumPy array with bounding boxes from the frame (`detections`) assigned to unique tracking identifiers.

- `display_frame(image, tracks)` – Again, this method is not strictly required by `main_detect.py`. It is used mostly for testing and debugging purposes. The purpose of this method is to draw the frame passed as the first argument (`image`) and plot onto it bounding boxes labelled with tracking identifiers, passed as the second argument (`tracks`).

It is important to note that the tracking wrapper object is supposed to save intermediate tracking data for a particular video footage between the calls of `track` method. That is why the video is passed as an argument to the class constructor and not to the `track` method. This makes each instance of the object tied to tracking one particular video footage. This approach works well with simple, lean trackers, but is suboptimal if the model for tracking needs a lot of time to be loaded into memory. This design of Object Tracking wrapper constrains Object Tracking frameworks to online frameworks.

In the code provided, the file `sortwrapper.py` implements Object Tracking wrapper for Simple Online and Realtime Tracker. `track(image, detections)` returns tracked bounding boxes for the frame given in the following format: the NumPy array contains a list of bounding boxes represented as such:

$$[x_1, y_1, x_2, y_2, id]$$

where $x_1, y_1 x_2, y_2$ are bounding box coordinates, while id is the tracking identifier. There are no guarantees that the number of bounding boxes

that are passed in `detections` to be tracked will be the same as the number of bounding boxes returned by the tracking method. This is firstly due to the fact that some input bounding boxes may be filtered out because of not being in `classes_to_track` list. The other reason is that the implementation of sort used makes no such guarantees [1].

4.1.4 VideoWrapper object

The purpose of the video object is to represent video footage in a way that is easy to use from the other wrapper classes. VideoWrapper's main aim is to make OpenCV procedures more legible. It was initially a part of the Object Tracking wrapper, but we decided to move it to a separate class. This decision was made on the following grounds: to allow greater encapsulation, useful for privacy, as described in privacy considerations (see subsection 6.1.3); and because of good practice of object oriented programming to implement classes as single-purpose objects.

The `VideoWrapper` class is implemented in the `videowrapper.py` file. It implements the following methods:

- `__init__(videopath)` – The class constructor. It initialises an OpenCV `VideoCapture` object. The argument `videopath` is technically optional, as it loads a sample video by default.

- `load_video(videopath)` – This method loads a new video to the instantiated object. The behaviour is undefined when a VideoWrapper object is first initialised with one video footage, then passed as an argument to Object Tracking and then the video footage is changed using this method. There are no safeguards to prevent this and it has not been tried, but we foresee possible strange behaviour.

- `set_next_frame(frame_number)` – This method accepts one argument – `frame_number` – which is the number of the frame to be returned by one of the `get_next_frame` methods. This method will set the frame number to the last frame of the footage if the frame number passed as an argument is too big.

- `get_frame_count()` – This method returns an integer containing the length of the video in frames.

- `get_video_resolution()` – This method returns a tuple of integers – width and height of the video footage in pixels.

- `get_video_framerate()` – This method returns an integer containing the framerate of the video in frames per second.

- `get_next_frame_nparray()` and `get_next_frame_cv2()` are methods which return the next consecutive frame of the video. The order of the frames returned by these methods can be changed by calling the `set_next_frame` method. The `..._nparray` version of the method returns the frame as a NumPy array while the `..._cv2` version returns an OpenCV-format image.

4.1.5 Activity Recognition wrapper

Activity Recognition wrapper is a wrapper for the activity recognition models, developed for this project. It is not used for training, only for evaluation. The wrapper must implement the following methods:

- `__init__(action_classes)` – The class constructor. It accepts a single argument `action_classes` which is an array of strings representing activity names to be used to label recognised activities. It should initialise the model and load it into memory. The constructor can have optional arguments if they have a default value.

- `predict(video, saved_tracks)` – This method returns a dictionary where keys are bounding box tracking identifiers and values are action label predictions. It accepts two arguments: `video`, which is a `VideoWrapper` object and `saved_tracks`, which is an array of outputs of Object Tracker's `track` method for each frame. It is the responsibility of this method to ensure proper data preparation and prediction.

In the code provided, the Activity Recognition wrapper is implemented for two versions of the `ActionCNN` 3-dimensional convolutional neural network. Both versions will be described in section 4.3 below. The wrappers are saved in files `actioncnnwrapper.py` and `actioncnnposwrapper.py`. They both implement another pair of helper methods:

- `data_preparation_batch(video, saved_tracks)` – This helper method, as the name suggests, performs data preparation, further described in section 4.3.2.

- `predict_no_data_preparation(data_array)` – This helper method accepts only one argument, `data_array`, which contains data prepared for activity recognition. It returns the exact same dictionary of bounding box tracking identifiers and action labels as in the `predict` method. It not only runs the model, but also processes the output to this dictionary form.

4.2 Activity Recognition model based on 2-dimensional Convolutional Neural Network

The first model for activity recognition failed. It was based off of the idea that frames of the video can be "squeezed" into a singe dimension, allowing for a standard 2-dimensional convolutional neural network to classify it. We will skip its description as it is irrelevant to the final project. It is important to note that failure of this model necessitated some special planning and mitigation, described in section 6.2.2.

4.3 Activity Recognition model based on 3-dimensional Convolutional Neural Network

The second model developed for activity recognition turned out to perform well, as it will be discussed in chapter 5.

4.3.1 Basic model

The basic model, depicted in figure 4.1, is a small and rather common convolutional neural network. The only noticeable difference is that it is 3-dimensional, as opposed to 2-dimensional. The network uses Maximum pooling and Rectified Linear function. Its input is a $60 \times 60 \times 20$ tensor, for 60 frames times 20×60 frame resolution. It outputs a vector of size 6, each field being the probability that the input belongs to an activity class, for a total of 6 possible classes.

4.3.2 Data Preparation

Data preparation is a process which turns the video footage and a list of tracked bounding box coordinates into the format accepted by the model. The data preparation steps are as follows:

1. Crop the video and obtain a set of videoclips cropped to bounding box contents
2. Resize the obtained videoclips to 20×60 resolution
3. Pad the time of the clips: loop each of them until 60 frames long
4. Save each prepared clip

```
 ActionCNNModel(
Sequential(
(0):  Conv3d(1, 16, kernel_size=(2, 3, 3), stride=(1, 1, 1))
(1):  BatchNorm3d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(2):  ReLU()
(3):  MaxPool3d(kernel_size=(1, 2, 2), stride=(1, 2, 2),
padding=0, dilation=1, ceil_mode=False)
(4):  Dropout(p=0.5, inplace=False)
(5):  Conv3d(16, 32, kernel_size=(3, 2, 2), stride=(1, 1, 1))
(6):  BatchNorm3d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(7):  ReLU()
(8):  MaxPool3d(kernel_size=(2, 2, 2), stride=(2, 2, 2),
padding=0, dilation=1, ceil_mode=False)
(9):  Dropout(p=0.5, inplace=False)
(10):  Conv3d(32, 64, kernel_size=(2, 2, 2), stride=(1, 1, 1))
(11):  BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(12):  ReLU()
(13):  MaxPool3d(kernel_size=(4, 3, 3), stride=(4, 3, 3),
padding=0, dilation=1, ceil_mode=False)
(14):  Dropout(p=0.5, inplace=False)
(15):  Flatten(start_dim=1, end_dim=-1)
(16):  Linear(in_features=1536, out_features=128, bias=True)
(17):  ReLU()
(18):  Dropout(p=0.5, inplace=False)
(19):  Linear(in_features=128, out_features=6, bias=True)
)
)
```

Figure 4.1: The basic model structure

4.3.3 Later addition – bounding box position tracking

Bounding box position tracking in this model was realised by adding a single fully-connected layer alongside the convolution. This fully-connected layer accepts the input of length 240, which stands for 4 elements per frame. These elements are $[x_1, y_1, x_2, y_2]$ – the coordinates of the top left and bottom right corner of the bounding box at each frame. The network is otherwise the same as in the basic model. The full description of the network is shown in figure 4.2.

```
 ActionCNNPositionModel(
(cnn_network):  Sequential(
(0):  Conv3d(1, 16, kernel_size=(2, 3, 3), stride=(1, 1, 1))
(1):  BatchNorm3d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(2):  ReLU()
(3):  MaxPool3d(kernel_size=(1, 2, 2), stride=(1, 2, 2),
padding=0, dilation=1, ceil_mode=False)
(4):  Dropout(p=0.5, inplace=False)
(5):  Conv3d(16, 32, kernel_size=(3, 2, 2), stride=(1, 1, 1))
(6):  BatchNorm3d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(7):  ReLU()
(8):  MaxPool3d(kernel_size=(2, 2, 2), stride=(2, 2, 2),
padding=0, dilation=1, ceil_mode=False)
(9):  Dropout(p=0.5, inplace=False)
(10):  Conv3d(32, 64, kernel_size=(2, 2, 2), stride=(1, 1, 1))
(11):  BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(12):  ReLU()
(13):  MaxPool3d(kernel_size=(4, 3, 3), stride=(4, 3, 3),
padding=0, dilation=1, ceil_mode=False)
(14):  Dropout(p=0.5, inplace=False)
(15):  Flatten(start_dim=1, end_dim=-1)
)

(bbox_tracking):  Sequential(
(0):  Linear(in_features=240, out_features=60, bias=True)
(1):  ReLU()
(2):  Dropout(p=0.5, inplace=False)
)

(out_network):  Sequential(
(0):  Linear(in_features=1596, out_features=128, bias=True)
(1):  ReLU()
(2):  Dropout(p=0.5, inplace=False)
(3):  Linear(in_features=128, out_features=7, bias=True)
)

)
```

Figure 4.2: The model structure with the addition of bounding box tracking

4.4 Datasets

The aim of this section is to describe how the dataset used for training and evaluation of the model was preprocessed. In addition to the KTH dataset [17], an additional small custom dataset was prepared as well. The training process is also outlined.

4.4.1 Dataset preprocessing

KTH Dataset consists of video clips divided into 6 classes: boxing, hand-clapping, handwaving, walking, jogging, running. Each video clip contains on average four examples of these activities. In order to use the dataset with our models, it is necessary to divide footage into single examples. While doing so, we crop the videos to bounding boxes and save their location over time alongside. Please refer to algorithm 2.

The script is saved as `KTH-Split.ipynb` for a version that does not save bounding box location; and `KTH-Split-tracked.ipynb` for a version that does. Keeping both versions is a convenience and their principles are otherwise identical.

Algorithm 2 Dataset preprocessing

```
Files ← a list of videos and timestamps of the beginning of the examples
for all Video, Timestamps ∈ Files do
    for all Timestamp ∈ Timestamps do
        Frame ← the first frame of Video
        Tracks ← an empty list
        while Video is not at next Timestamp do
            Detections ← ObjectRecognitionWrapper(Frame)
            Track ← ObjectTrackingWrapper(Frame, Detections)
            Tracks ← Tracks ∪ {Track}
            Frame ← the next frame of Video
        end while
        CroppedVideo ← CropToBoundingBoxes(Video, Tracks)
        Save(CroppedVideo)
        Save(Tracks)
    end for
end for
```

We assume that the output of the tracking algorithm is error-free and we save a single video clip for each activity example.

4.4.2 CCTV Dataset

In addition to KTH dataset, a small custom dataset was prepared for evaluation purposes. It consists of two activity classes: standing and walking. It was created by choosing by hand video files where all the people were standing or walking for their entire duration. The script that was used to generate the dataset is called `Standing_Set_Gen.ipynb`. The dataset output by this script is already preprocessed for training the model.

Algorithm 3 Generation of CCTV dataset

```
for all Video in Videos do
    Frame ← the first frame of Video
    Tracks ← an empty list
    while Video is not at last frame do
        Detections ← ObjectRecognitionWrapper(Frame)
        Track ← ObjectTrackingWrapper(Frame, Detections)
        Tracks ← Tracks ∪ {Track}
        Frame ← the next frame of Video
    end while
    CroppedVideos ← CropToBoundingBoxes(Video, Tracks)
    for all CroppedVideo in CroppedVideos do
        Save(CroppedVideo)
    end for
end for
```

This method of dataset creation is fairly efficient if we find videos of crowds waiting to cross the street or walking. The quality of video clips obtained is low, but it is on par with the data given as an input to the classifier, so it should be suitable for training.

4.4.3 Training

Both models are small, so their training does not require particularly large resources. In fact, data preparation takes much longer to compute than the training itself. The model was trained for 300 epochs with a learning rate of 0.0002. Smaller learning rate sometimes led to a dying ReLU problem. The optimiser chosen was Adam. Data was fed to the models in batches of 32 and the weights were backed up every ten epochs. As it is visible in figures 4.3 and 4.4, there was slight overfitting, but it did not end up being a major issue. The accuracy was steadily increasing as the epochs passed, as shown in figures 4.5 and 4.6. The training took between 40 minutes and an hour to complete.

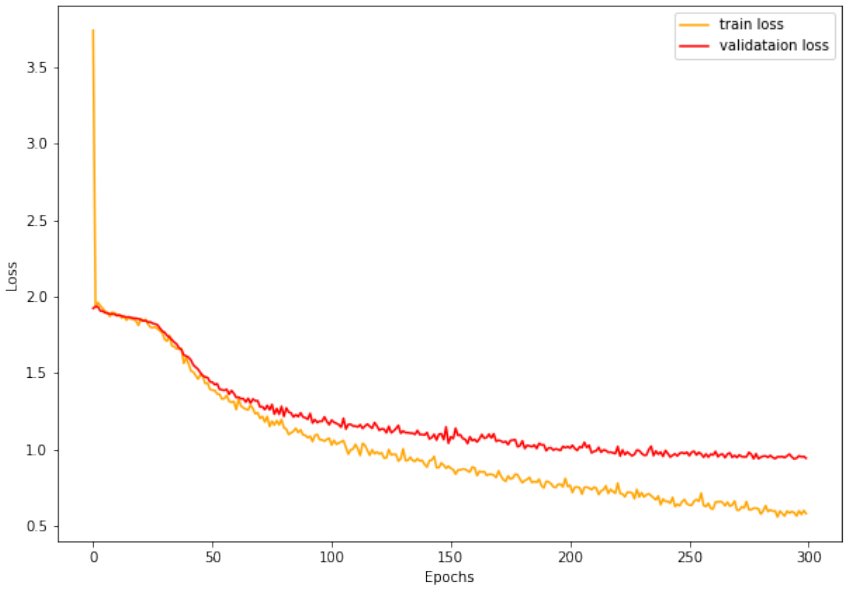


Figure 4.3: Plot of training and validation loss for the basic model

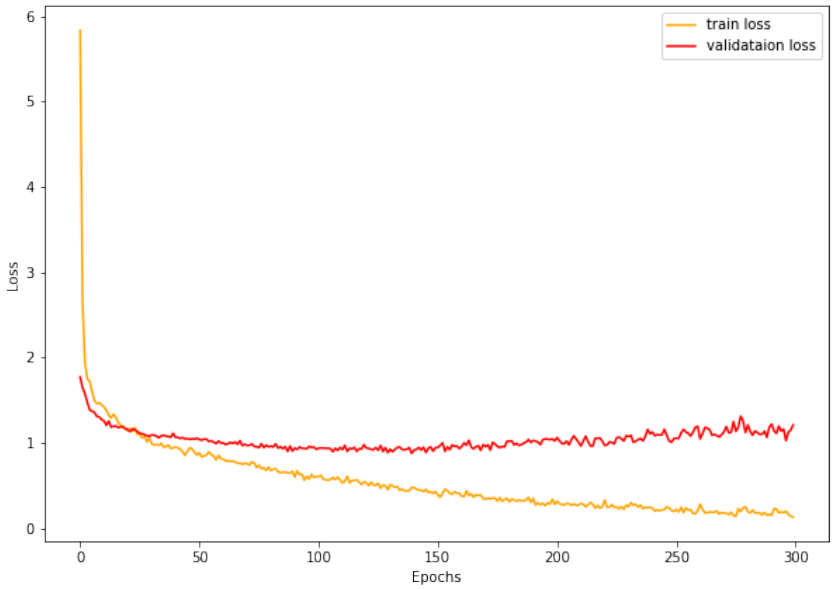


Figure 4.4: Plot of training and validation loss for the model with bounding box tracking

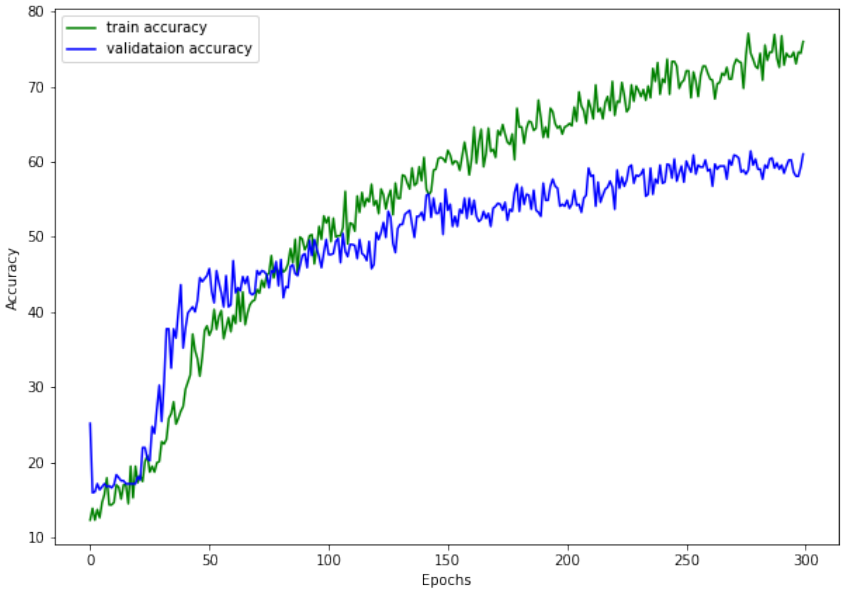


Figure 4.5: Training accuracy for the basic model

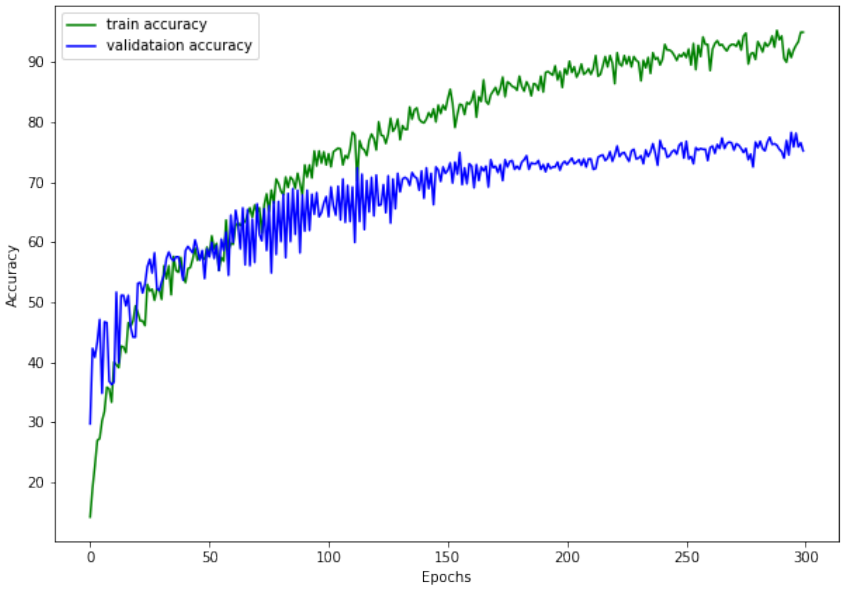


Figure 4.6: Training accuracy for the model with bounding box tracking

Chapter 5

Evaluation

The purpose of this section is to present how our human activity detection framework performs. We will mainly focus on measuring the accuracy and performance of the 3-D convolutional activity recognition model, as it was designed and trained specifically as a part of this project. We will then discuss limitations of the model and the entire framework and conclude with future work.

5.1 Results

In testing the model we use the following metrics:

- Accuracy – the percentage of correctly labelled features in the testing dataset
- Precision – this metric assesses how many recognitions of each of the recognised activity classes are correct; in other words it is a measure of validity of results.
- Recall – this metric assesses how many of the activities of each of the classes were "caught"; in other words it measures sensitivity of the model to recognise each activity class; the completeness of the results obtained.

Precision and recall are presented both as global values and for each recognised activity. In addition to that, confusion matrix is shown to visualise what kinds of mistakes the model makes.

5.1.1 Basic 3D Convolution Results

The basic 3D convolutional neural network was the only model presented during the presentation of the project in March. It was initially trained

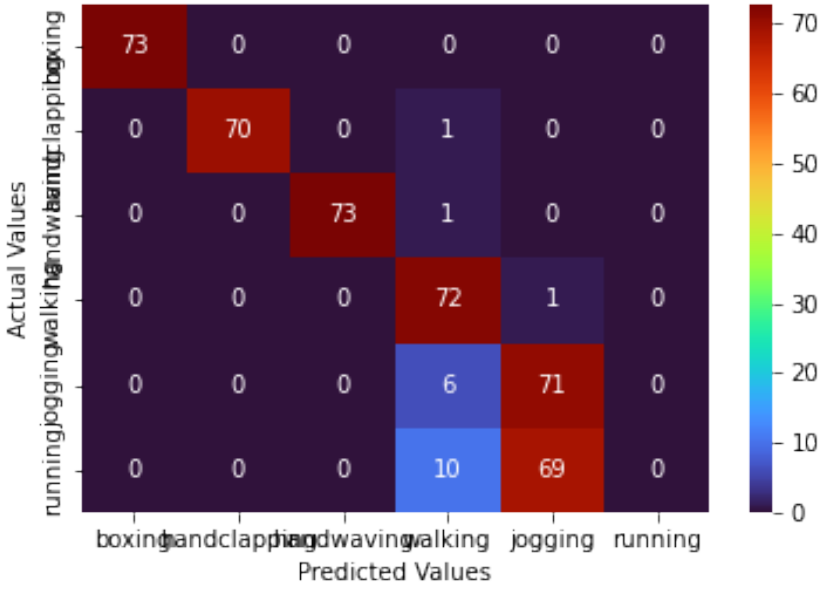


Figure 5.1: Confusion matrix for the basic 3D cnvolution – results obtained with random sampled training, testing and validation subsets of KTH

using randomly selected disjoint subsets of KTH dataset in the following ratios: training – 0.64, testing – 0.2, validation – 0.16 of the entire dataset. We decided to take this approach since the training, testing and validation sets suggested by the authors of KTH were almost equal in length, so there was seemingly little data for training. The upside of this approach is that – indeed – the network trained on this dataset performed well at **81%** accuracy. However, since the subsets were custom and randomly selected, each training of the model gave different accuracy (with differences of around 10%).

We decided to perform the final testing and measurements using the division of the data suggested by the KTH authors. As expected, the accuracy of the model dropped significantly, positioning itself at **56.4%**. Precision and recall were similar, at **58.5%** and **56.6%** respectively.

Metric	boxing	handclapping	handwaving	walking	jogging	running
Precision	0.58	0.58	0.77	0.61	0.49	0.48
Recall	0.79	0.75	0.49	0.25	0.56	0.54

The table above presents precision and recall for each activity class recognised by the model. It is worth noting that since classification is done purely on the contents of the bounding boxes, results of classes which look

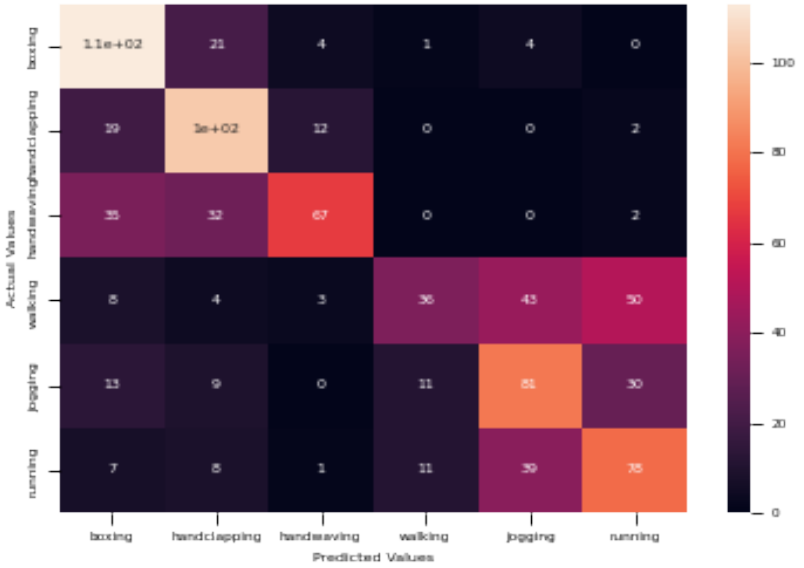


Figure 5.2: Confusion matrix for the basic 3D convolution – results obtained with training, testing and validation sets as suggested by the authors of the KTH dataset [17]

similar (jogging and running) have lower recall and precision. The confusion matrix in figure 5.2 shows that the most misclassified classes are those which look visually similar.

5.1.2 Bounding Box Tracking Results

During the project presentation in March it was suggested that we should try adding bounding box tracking to the model. It greatly improved the recognition results – the accuracy raised to **70.5%**, precision was **71.4%** and recall – **70.4%**.

Metric	boxing	handclapping	handwaving	walking	jogging	running
Precision	0.87	0.64	0.76	0.69	0.59	0.73
Recall	0.83	0.81	0.50	0.75	0.61	0.73

We see not only a sharp rise in precision of jogging and running recognition and recall of walking, but also a sharp precision increase in boxing. This is expected as the main differences between these classes lie in the speed of the agent movement. In case of boxing, we see reduced confusion with handwaving.

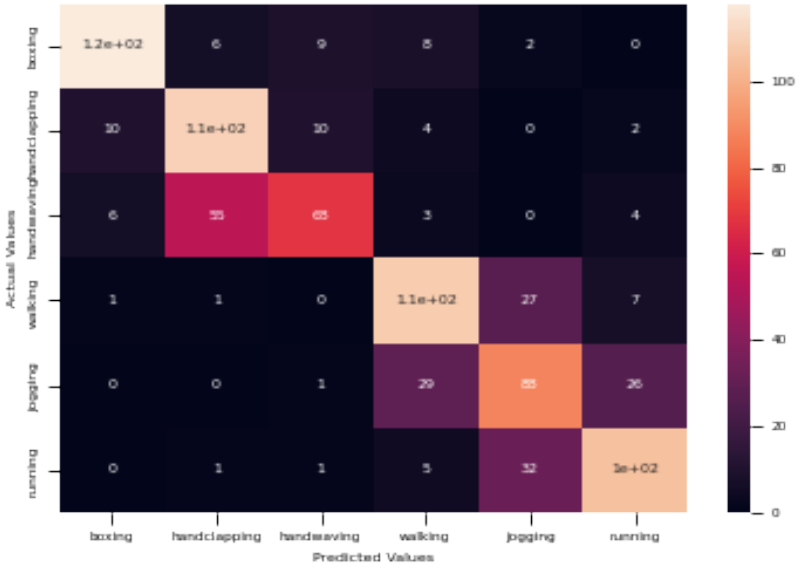


Figure 5.3: Confusion matrix for the 3D convolution with bounding box tracking – results obtained with training, testing and validation sets as suggested by the authors of the KTH dataset [17]

5.1.3 Reducing footage resolution

Since the results obtained with JamCams CCTV footage were highly unsatisfying (compare figure 5.4), we decided to investigate how much we can reduce the resolution of footage and still get sensible results. The original resolution of KTH clips is 160×120. We tried reducing it to 80×60 and 60×40. In the case of 60×40 the resolution was too low for YOLO to recognise the people, so the results presented here were obtained using the 80×60–resolution footage.

Metric	boxing	handclapping	handwaving	walking	jogging	running
Precision	0.75	0.43	**0.81**	0.40	0.27	0.23
Recall	0.55	0.58	**0.95**	0.51	0.43	0.24
Prec+Track	0.91	0.52	0.27	0.20	0.08	0.19
Recall+Track	0.23	0.50	0.42	0.02	0.03 **0.67**	

The overall accuracy was **41.1%** for the basic model and **29.6%** for the model with bounding box tracking. The basic model's precision was **48.3%** and its recall was **40.3%**. When it comes to the model with bounding box tracking, the precision was **36.1%** and the recall was **30.9%**. The results are interesting, as the basic model performed better than the model with

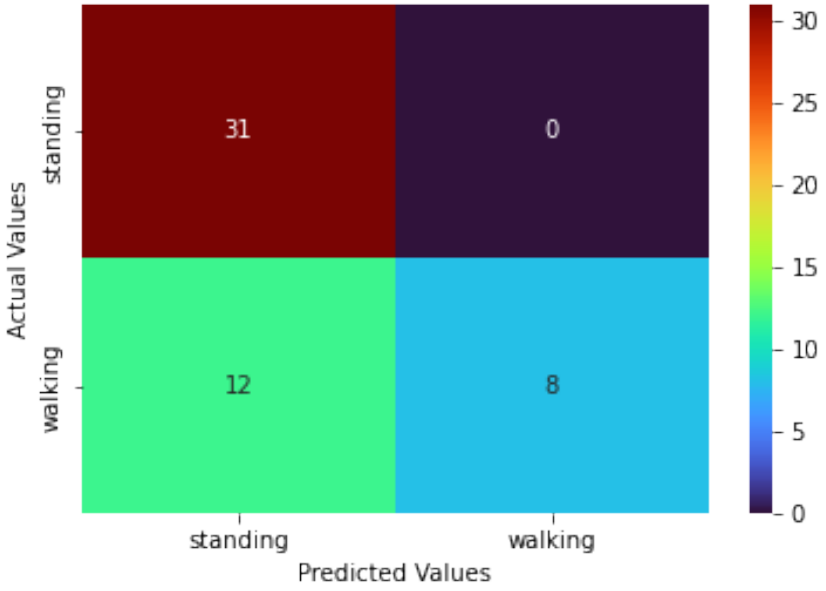


Figure 5.4: Confusion matrix for the basic 3D convolution. It was obtained by training the model on the custom CCTV dataset described in section 4.4.2

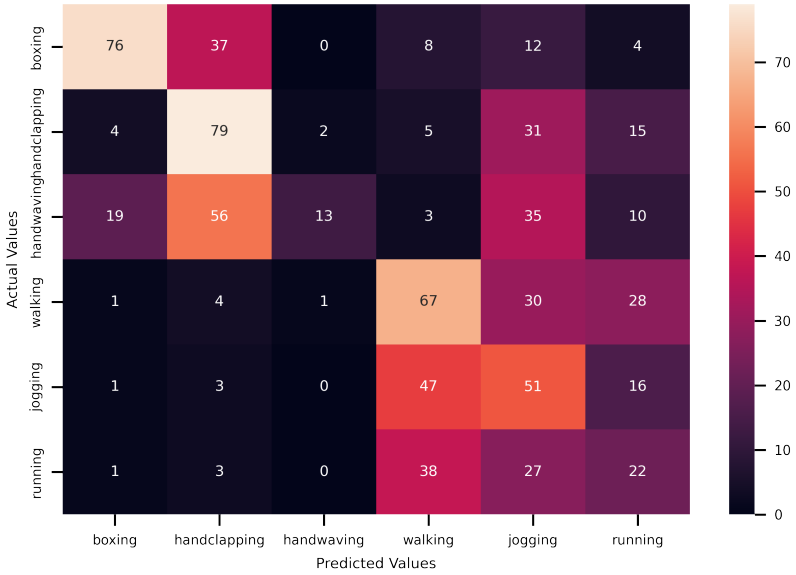


Figure 5.5: Confusion matrix for the basic 3D convolution – training and validation sets as suggested by KTH authors [17]; testing set of reduced quality

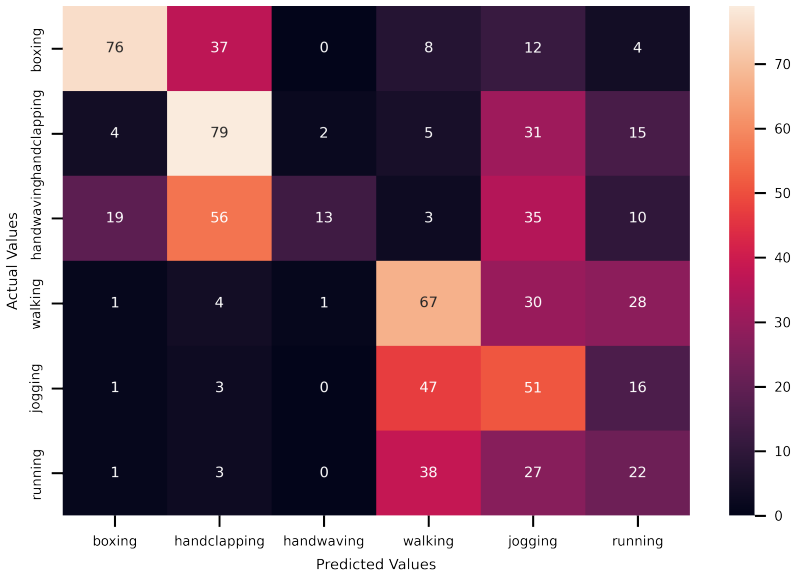


Figure 5.6: Confusion matrix for the 3D convolution with bounding box tracking – training and validation sets as suggested by KTH authors [17]; testing set of reduced quality

tracking. The outliers here are: surprisingly high handwaving precision and recall in the basic model; and relatively high recall of the running class in the model with tracking.

We hypothesise that at this reduced footage resolution the object tracking was highly inaccurate and affected the results negatively, hence lower performance of the "tracking" model.

To sum up, the global metrics for models tested in different conditions are as follows:

Model	Data	Accuracy	Precision	Recall
Basic	KTH	0.564	0.585	0.566
Tracking	KTH	**0.705**	**0.714**	**0.704**
Basic	LowRes	0.411	0.483	0.403
Tracking	LowRes	0.296	0.361	0.309

The framework performed at **25–28** frames per second, making it suitable near-real time use. Since the convolutional neural network has to analyse the full 60 frames, the latency of this model in online settings would be approximately 2–3 seconds.

5.2 Limitations

As any framework, this project has some limitations, presented below.

5.2.1 Footage quality

As shown in detail in section 5.1.3, model accuracy drops in lower resolution of the footage. This is due to the fact that recognition is performed on the visual contents of the bounding boxes. Usually, human figure occupies a small fraction of the video frame, therefore the resolution of its bounding box is low. The model limits bounding box resolution to 60×20 pixels anyway, so there is a limit to how much the model accuracy can be improved by improving the resolution of the input video.

5.2.2 Bounding box tracking

While the basic version of the model does not have such limitation, the model with bounding box tracking has to be provided with stationary footage. The tracking of bounding boxes is basic and handles its movement with regards to the position on screen rather than in the real world. Therefore a non-stationary footage would completely change the way that the movement of the bounding box is perceived by the model.

5.2.3 Types of activities recognised

The way this framework works, that is classifying short snippets of low resolution video footage, makes it suitable for a particular kind of human activities: short, repetitive and self-contained. Short, because the architecture of the model limits analysis to 60 frames; repetitive, because in case the footage from the bounding box is shorter than 60 frames, its contents are augmented by looping; self-contained because no information from outside of the bounding box is provided to the classifier. Therefore activities such as walking, standing, dancing would probably get a higher accuracy than for instance sitting down on a chair.

5.3 Future work

The potential of this project is still far from exhausted. There are several ideas and possibilities which are not in scope of the project due to time constraints, but would be interesting or beneficial to explore:

- **LSTM Networks** – Replacing the action classification module with an LSTM-based version. This would allow for a fully online version of the framework.

- **Integration into existing projects** – Using Docker to create reproducible builds of the framework would make it much easier to include in the infrastructure of existing projects. In this case, worth mentioning are the projects which were a direct inspiration for this one – London Air Quality project and Project Odysseus.
- **Preparing more wrappers** – It would be interesting to investigate performance of the framework with different object recognition or tracking algorithms and compare results.

Chapter 6

Conclusions

This chapter collects important closing remarks on the project. First, ethics and privacy issues are considered. We follow by briefly outlining how the project plan evolved and how we overcame challenges faced. We close the report with acknowledgements.

6.1 Ethical Considerations

Ethical considerations play an important role in modern research, especially research which handles real-life data and poses a privacy risk. In this part of the report we want to explore the project from the prespective of privacy and ethics. We'll first consider a case of failed data anonymisation, followed by the description of how the privacy considerations were chaining thought the project. We will conclude by describing a design of a system which incorporates the framework presented in this report in a safe manner.

6.1.1 Case Study

When dealing with public datasets and handling of big data in general, it is important to consider the ethics and possible misuse by malicious agents of data obtained using the means described in this report. The following paragraphs outline a popular case of failed data anonymisation. It is done in order to highlight possible ways to de-anonymise data and undertake reasonable mitigation.

Taxicab dataset

The case study focuses on the so-called Taxicab dataset. It is a dataset containing data of every taxi journey in New York City in 2013. The data consists of the exact time and place of pick up and drop off for every trip, as well as the fare and tip, hashed license and medallion number of the taxi cab

[22]. The data was published as a result of a request of Christopher Whong, an Open Data advocate. Mr. Whong filled in a request in accordance to Freedom of Information Law. Although the data appeared anonymous, since the identifying data of the vehicles were hashed and no data on the passengers was made public, it was used in several ways to disclose various pieces of information that were not in public domain.

First of all, the license plate and medallion number of the taxi cabs were hashed using an MD5 algorithm, long considered unsafe at the time the data was published [18]. Moreover, hashing is a suitable means of data anonymisation only if the pattern of the data is unknown. When the pattern is known, the number of possible combinations that the hash can represent is limited and de-hashing with a brute-force method is possible. In this case, both the license plate and the medallion number could be easily deanonymised as they followed strict patterns. These deanonymised data could allegedly be matched with personal data of the driver [4]. The information obtained in this way, for example combined with historical data of particular cab's common routes, could be used by a malicious agent.

The second problem with this dataset has to do with the exact pick up and drop off locations and times. Let us suppose that there is a malicious agent who wants to target a specific vulnerable group of people, for example a specific religious community or an LGBT people. The agent may identify trips to or from a specific place, for example a church or LGBT club, and infer, with some certainty, place of residence of people who frequent it.

Another issue, identified by Anthony Tockar [19], deals with the privacy of celebrities. Due to availability of time-stamped paparazzi photos of celebrities entering or exiting New York's taxis, it is possible gather data on their trips, such as origin, destination and the tip amount. The taxis in New York City display their medallion number in various places, so the number is legible on some of the photos. This, together with de-hashed version of the dataset enables malicious agent to identify a trip and gather the aforementioned pick up, drop off locations and tip amount. It has also been claimed [4] that there exist serious discrepancies in data with regards to tips paid in cash. This can affect not only celebrities' privacy, such as revealing their places of residence and places they frequent, but also one can see potential accusations of greed etc.

Possible Mitigations

We argue that it is not possible to mitigate all of the risks identified above by employing additional data anonymisation techniques. The vehicle identification could be easily mitigated by assigning a random identification number to each taxi cab, instead of hashing the real values with an unsafe hashing algorithm. It can be said that due to the nature of the data hashed,

any hashing algorithm would be prone to a brute-force de-hashing attack. On the other hand, the pick up and drop off locations cannot be obfuscated without affecting the data quality. Instead of providing the exact location data, approximate data could be used instead.

While the availability of this data can be useful for people who analyse the city traffic or architecture and urban design specialists, we argue that it is dangerous for privacy to collect detailed data in this form at all. It would be much safer to isolate time of the day, approximate source and destination from trip cost and tip data and discard trip date and vehicle identification. Data collected in this way would be way less prone to deanonymisation while remaining interesting sources for those who want to analyse them without malicious intents.

6.1.2 Evolution of ethical considerations following progress of the project

In this subsection we will outline how our awareness of the need of considering the privacy implications of our human activity detection framework evolved. As mentioned in chapter 1 and further described below in section 6.2, the initial plan of the project was vague, therefore it was not possible to consider all the privacy implications from the outset.

We initially assumed that it is sufficient to rely on ethical and privacy considerations of Project Odysseus [6]. At that point we assumed that our project will only be of use within the scope of this project, used only on anonymised footage that Project Odysseus uses. This assumption was essentially flawed, as it soon turned out that the scope of the project will be more general; that one will be able to analyse any footage using it and that it will be possible to do it in near-real time.

Initial evaluation and experiments with the 3-dimensional convolutional neural network–based activity detection showed that the quality of anonymised JamCams footage is insufficient for the activity detection framework developed. In fact, we believe that the increased footage quality can generally positively impact performance of the model. Therefore, additional consideration had to take place.

6.1.3 Isolation through Modularity

Let us first consider the data used and obtained at each module of the framework.

- *Object Detection* – The first module of the framework is Object Detection. This module takes raw video footage and localises agents which perform the soon-to-be-recognised activities. In the most general case we cannot assume the footage is anonymised in any way.

The new information is the location of people on each frame of the video – it is important to note that bounding boxes are not tracked between frames yet.

- *Object Tracking* – The second module of the framework is Object Tracking. Here, the input is consecutive frames of footage with detected people. The output of object tracking consists of the location of each person in the video over time. It is important to note though, that the location and track obtained by object tracking are purely "positions on the screen" and do not reflect spatial positions in real world. We consider this point of the framework to require most care with regards to privacy, as at this point the data caries the most information. It is worth noting that video footage itself is separated form object tracking – the bounding box positions over time ("tracks") are stored separately from the footage. A crucial moment is extraction of the bounding box contents.

- *Activity Detection* – Before we analyse activity detection itself, let us consider the process of extracting bounding box contents. At this point, footage is not only reduced to bounding boxes' contents, but also its quality is reduced to 20×60 pixels and colour channels are removed, leading to a black-and-white footage. This reduces the possibility to recognise a particular person to the shade of their clothing, as for instance the face is reduced to a mere dozen pixels. Activity Detection itself handles less data than the previous module, Object Tracking, as it takes into account only the short excerpts of bounding box content and movement of the box.

This project does not aid or facilitate tracking of people across multiple cameras. In fact, "tracking" refers to bounding box identification, rather than actual tracking of people in the physical world, and is technically limited to 60 frames thanks to the design of the Activity Detection model. However, combined with external data, such as timestamps and placement of CCTV cameras it is technically possible to track movement of an individual between the cameras by the colour of their clothing. It is however possible to do with Object Detection itself, rather than our framework, and so is not in scope of our analysis – we have no power to mitigate it.

As mentioned previously, each of these modules works independently. In critical use cases each module can be run on a separate machine to minimise the amount of data available in the same place.

It is clear that the analysis performed by a single person is bound not to be exhaustive. There is always a possibility that some database or dataset, when combined with data obtained through this model, may be used maliciously. The purpose of the analysis above is to show that we

took care to mitigate identified potential problems. We truly hope that our framework will be used in a responsible and privacy-respecting manner. We therefore welcome any further feedback regarding privacy and ethical considerations in connection with human activity recognition in the scope of this project and in general.

6.2 Management

We will now briefly describe how the project evolved during our work on it.

6.2.1 Evolution of the plan of work

The initial plan accounted for the fact that the scope of the project was not yet known. Therefore it was rather vague.

Week	**Event**
T1 W2	Submission of this document
T1 W2-4	Literature review on Machine Learning and Computer Vision
T1 W5-6	Familiarising with relevant existing technologies and methodologies used within the Odysseus project
T1 W7-10	Exploring the possibilities of incorporating temporal aspect into object classification
T1 W9	Submission of Progress Report
	Christmas Holidays
T2 W1	Writing the first outline/draft of the Final Report
T2 W2-W7	Developing a solution to recognise temporal activities of pedestrians, such as walking, gathering, loitering, entering or exiting a building.
T2 W9-W10	Project Presentation
	Easter Holidays
T3 W1	Submission of Final Report

In December, at the time when the Progress Report was due, we had the literature review ready and the experimental binary classification model described in section 2.3 complete. In the second term the main focus was put on developing the main pedestrian activity detection framework, training the models. During this time we also prepared the project presentation. Following the advice received during the presentation, we conducted additional development by adding bounding box position tracking to the model and made additional tests. The work on the project concluded with writing this final report.

6.2.2 Challenges

Since the initial plan, the project has experienced a number of issues, closely described below. Some of them have a source in the fact that the initial Project Specification had to be delivered within mere days of the decision to change the subject of the project.

Hurried Planning

Due to changing the subject of the project at the last moment, Project Specification had to be written in one day. This resulted in a vague specification and work schedule relied mostly on tasks that were agreed on during weekly meetings. Another drawback is that it was not possible to assess the progress of the project with precision. In the progress report we attempted to estimate whether the project was on track. We later made use of TO-DO lists to track tasks due.

Insufficient Background

The initial plan for the project was to develop a tool for seamless audio splicing with prof. Tanaya Guha. However, upon Tanaya's departure from the University, the project has been assigned to prof. Theodoros Damoulas. Due to a different area of expertise of Theodoros, it has been decided to change the subject of the project at the last moment. However, the author had no prior background in deep learning or computer vision. Therefore, as a mitigation, a significant amount of project time has been assigned to background reading and literature review. This made it possible to gain the experience needed to complete the project.

Communication issue

The issue identified in December during the work on the Progress Report concerned communication between the author and the Project Supervisors. The concern raised was that weekly meetings are unfocused and unproductive. It was mentioned that the results of work are not discussed in enough detail or presented at all; that problems to be discussed were not stated in a way rigorous enough to warrant an insightful and meaningful answers; that the structure of the meetings was unplanned and therefore chaotic. As a mitigation, the following measures were introduced:

- Rewriting of handwritten personal notes to Markdown documents inside the project git repository
- Making the project git repository available for Project Supervisors

- Sharing the weekly meeting agenda with Supervisors via email in the evening preceding the meeting
- More rigorous presentation of work during weekly meetings – including summary of the notes taken and analysis of results

This issue tackled the methodology of work, and not the progress itself, therefore it did not impact the schedule. The weekly meeting agenda did not affect the amount of time needed to prepare for the weekly meetings, as it had always been prepared ahead of the meetings; we just concluded that it was a good idea to share the agenda with the supervisors. Rewriting the handwritten personal notes to markdown helped with preparing the project presentation in March and the Final Report.

Underperformance of the 2-dimensional convolutional neural network

The model that we initially planned to implement for activity recognition did not give any good results. Its output was essentially close to random. We had to mitigate by designing another model, which turned out to be a 3-dimensional convolutional neural network and alter the schedule to allow some time to implement and test it. The modular nature of our framework helped a lot and allowed us to reuse a big portion of the code. Time was reassigned from preparing the March presentation, which had to be written and rehearsed in three days instead of a week. The mitigation succeeded and the performance of the new model turned out to be satisfactory. The presentation, although made more risky by this decision, went well as well.

6.3 Acknowledgements

First, I need to acknowledge people whose work made this project possible: Joseph Redmon and Ali Fahradi – the authors of YOLOv3 and Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos and Ben Upcroft – the authors of Simple Online and Realtime Tracking.

I extend my gratitude to my supervisors, Theo and James, whose patient support and suggestions were vital to progressing with the project.

Finally, I want to thank my family and friends whose support and encouragement helped me bring this project to completion in these difficult times.

Bibliography

[1] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, sep 2016.

[2] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[3] Efros, Berg, Mori, and Malik. Recognizing action at a distance. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 726–733 vol.2, 2003.

[4] Alex Hern. New york taxi details can be extracted from anonymised data, researchers say. `https://www.theguardian.com/technology/2014/jun/27/new-york-taxi-details-anonymised-data-researchers-warn`, 2014. [Online; accessed 01-May-2022].

[5] The Alan Turing Institute. London air quality. `https://www.turing.ac.uk/research/research-projects/london-air-quality`, 2020. [Online; accessed 01-May-2022].

[6] The Alan Turing Institute. Project odysseus. `https://www.turing.ac.uk/research/research-projects/project-odysseus-understanding-london-busyness-and-exiting-lockdown`, 2020. [Online; accessed 01-May-2022].

[7] Mihir Jain, Jan C Van Gemert, and Cees GM Snoek. What do 15,000 object categories tell us about classifying and localizing actions? In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 46–55, 2015.

[8] Zhuolin Jiang, Zhe Lin, and Larry S. Davis. A unified tree-based framework for joint action localization, recognition and segmentation. *Computer Vision and Image Understanding*, 117(10):1345–1355, 2013.

[9] Quoc V Le, Will Y Zou, Serena Y Yeung, and Andrew Y Ng. Learning hierarchical invariant spatio-temporal features for action recognition

with independent subspace analysis. In *CVPR 2011*, pages 3361–3368. IEEE, 2011.

[10] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler. MOT16: A benchmark for multi-object tracking. *arXiv:1603.00831 [cs]*, March 2016. arXiv: 1603.00831.

[11] L. Minh Dang, Kyungbok Min, Hanxiang Wang, Md. Jalil Piran, Cheol Hee Lee, and Hyeonjoon Moon. Sensor-based and vision-based human activity recognition: A comprehensive survey. *Pattern Recognition*, 108:107561, 2020.

[12] Brendan Tran Morris and Mohan Manubhai Trivedi. Trajectory learning for activity understanding: Unsupervised, multilevel, and long-term adaptive approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(11):2287–2301, 2011.

[13] Antonios Oikonomopoulos, Maja Pantic, and Ioannis Patras. Sparse b-spline polynomial descriptors for human activity recognition. *Image and Vision Computing*, 27(12):1814–1825, 2009. Visual and multimodal analysis of human spontaneous behaviour:.

[14] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger, 2016.

[15] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[16] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015.

[17] Christian Schuldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: a local svm approach. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 3, pages 32–36. IEEE, 2004.

[18] Marc Stevens. Fast collision attack on md5, 2006. m.m.j.stevens@student.tue.nl 13224 received 17 Mar 2006.

[19] Anthony Tockar. Riding with the Stars: Passenger Privacy in the NYC Taxicab Dataset. `https://agkn.wordpress.com/2014/09/15/riding-with-the-stars-passenger-privacy-in-the-nyc-taxicab-dataset/`, 2014. [Online; accessed 01-May-2022].

[20] Michalis Vrigkas, Christophoros Nikou, and Ioannis A. Kakadiaris. A review of human activity recognition methods. *Frontiers in Robotics and AI*, 2:28, 2015.

[21] James Walsh, Oluwafunmilola Kesa, Andrew Wang, Mihai Ilas, Patrick O'Hara, Oscar Giles, Neil Dhir, and Theodoros Damoulas. Near real-time social distancing in london, 2021.

[22] Christopher Whong. Foiling nyc's taxi trip data. `https://chriswhong.com/open-data/foil_nyc_taxi/`, 2014. [Online; accessed 01-May-2022].

[23] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric, 2017.