Deliverable D 3.3

WP3 Report on experimentation, analysis, and discussion of results

Project acronym:	RAILS
Starting date:	01/12/2019
Duration (in months):	43
Call (part) identifier:	H2020-S2R-OC-IPX-01-2019
Grant agreement no:	881782
Due date of deliverable:	Month 37
Actual submission date:	July 25st 2023
Responsible/Author:	Francesco Flammini (LNU)
Dissemination level:	Public
Status:	Issued

Reviewed: no

Document history		
Revision	*Date*	*Description*
1	July 10th 2023	First issue for internal review
2	July 25th 2023	Second issue for EU review

Report contributors		
Name	Beneficiary Short Name	Details of contribution
Francesco Flammini	LNU	WP3 Leader
Lorenzo De Donato	CINI	Contributor
Valeria Vittorini	CINI	Contributor
Ruifan Tang	LEEDS	Contributor
Zhiyuan Lin	LEEDS	Contributor

Funding

This project has received funding from the Shift2Rail Joint Undertaking under the European Union's Horizon 2020 research and innovation programme under grant agreement n. 881782 Rails. The JU receives support from the European Union's Horizon 2020 research and innovation program and the Shift2Rail JU members other than the Union.

Disclaimer

The information and views set out in this document are those of the author(s) and do not necessarily reflect the official opinion of Shift2Rail Joint Undertaking. The JU does not guarantee the accuracy of the data included in this document. Neither the JU nor any person acting on the JU's behalf may be held responsible for the use which may be made of the information contained therein.

Contents

Executive Summary

This deliverable proposes AI approaches to address the problems and challenges that emerged from the methodological Proofs-of-Concept (PoCs) reported in Deliverable D3.2, in order to investigate the adoption of learning techniques and other AI methods to enhance and to optimize railway maintenance activities. On the basis of what partially analysed in previous deliverables, two specific experimental PoCs, namely, “Smart Maintenance at Level Crossings” and “Predictive Maintenance for Rolling Stock”, are provided to explore the technical feasibility of specific railway functionalities through the use of AI approaches.

To this aim, this document addresses the following themes for each case PoC: $i)$ a technical description of the proposed methods, highlighting the problem statement to solve and the learning techniques which are going to be leveraged; $ii)$ the definition of the training phase for the learning approaches, which could be conducted exploiting generated or selected datasets; $iii)$ the description of the validation procedure to show the effectiveness of the proposed strategies in concrete operational scenarios; $iv)$ a preliminary discussion of the results to highlight possible benefits and drawbacks of the innovative approaches.

The findings from these explorations indicate that AI could be a valuable asset in enhancing the efficiency and effectiveness of railway maintenance procedures. However, it is worth mentioning that these PoCs were not conceived to develop the most suitable solution for their specific tasks, rather, they were developed with the aim of highlighting important aspects and recommendations (that will be collected in the next Deliverable D3.4) that could be useful for future research in the context of smarter railway maintenance.

Public Benchmark from WP3. Some of the data collected to develop the “Smart Maintenance at Level Crossings” PoC (discussed in Chapter 4) are publicly available on Zenodo[1]. These data are a product of the research activity conducted in RAILS and created for scientific purposes only to study the potentials of Deep Learning (DL) approaches to detect level crossing warning bells. The RAILS Consortium and the authors **do not assume** any responsibility for the use that other researchers or users will make of these data.

[1] https://zenodo.org/record/7945412

Abbreviations and acronyms

Abbreviations / Acronyms	Description
AI	Artificial Intelligence
BAM	Barrier Analysis Module
BD	Barrier Dataset
BR	Bright
CDC	Clip from Day Clear video
CNC	Clip from Night Clear video
CNN	Convolutional Neural Network
DA	Dark
DAR	Day After Rain
DC	Day Clear
DL	Deep Learning
DNN	Deep Neural Network
DR	Day Rain
DS	Day Snow
FC	Fully-Connected
FF	Fitness Function
FN	False Negative
FO	Fog
FP	False Positive
FPS	Frames per Second
FVS	Frame-Level Validation Set
GE	Generic Alarm
GR	Gravel
GTA	Grand Theft Audio
HO	Hole
IoT	Internet of Things
IoU	Intersection over Union
IT	Italian
KPI	Key Performance Indicator
LbF	Loss-based Fitness
LC	Level Crossing
mAP	Mean Average Precision
NA	No Alarm
NAR	Night After Rain
NN	Night Clear
NR	Night Rain
NS	Night Snow
PoC	Proof-of-Concept
RMS	Root Mean Squared
RN	Rain
RQ	Research Question

SF	Sun Flare
SGD	Stochastic Gradient Descend
SH	Shadow
SN	Snow
SNR	Signal to Noise Ratio
TDC	CDC with Transformed frames
TJS	Trimmed Joint Set
TL	Transfer Learning
TNC	CNC with Transformed frames
TP	True Positive
TTS	Trimmed Training Set
UK	British
WB	Warning Bell
WBD	Warning Bell Dataset
WBDM	Warning Bell Detection Module
WL	Warning Light
WLD	Warning Light Dataset
WLDM	Warning Light Detection Module
RL	Reinforcement Learning
TSP	Travelling Salesman Problem
RSRP	Rolling Stock Rostering Problem
IRSR	Intelligent Rolling Stock Rostering
DQN	Deep Q-Network

1. Background

The present document constitutes Deliverable D3.3 “WP3 Report on experimentation, analysis, and discussion of results” of the Shift2Rail JU project “Roadmaps for AI integration in the Rail Sector” (RAILS). The project is in the framework of Shift2Rail’s Innovation Programme IPX. As such, RAILS does not focus on a specific domain, nor does it directly contribute to specific Technical Demonstrators but contributes to Disruptive Innovation and Exploratory Research in the field of Artificial Intelligence within the Shift2Rail Innovation Programme. The successor of the Shift2Rail Joint Undertaking is currently Europe’s Rail Joint Undertaking (EU-Rail) established by Council Regulation (EU) 2021/2085 of 19 November 2021.

The RAILS WP3 investigates the adoption of learning techniques and other AI methods to support smart maintenance in railways. The present deliverable is consequent to the results reported in Deliverable D3.2, in which methodological PoCs have been provided for two selected case studies: “Smart Maintenance at Level Crossings” and “Predictive Maintenance for Rolling Stock”.

As for **“Smart Maintenance at Level Crossings”**, this PoC aims at understanding to what extent non-intrusive sensors (cameras and microphones to be specific) could be exploited together with Deep Learning approaches to shift from scheduled inspection and corrective maintenance to continuous monitoring and predictive maintenance of Level Crossings. Some research questions have been identified in Deliverable D3.2 to drive this analysis; we also tried to understand if some of the AI approaches already assessed in other domains for other purposes could be actually used to improve railway practices. Furthermore, specific key performance indicators have been identified to evaluate the effectiveness of the proposed approaches. After an examination of different AI methods, some learning approaches have been assumed to be leveraged, namely, audio classification and AI-aided computer vision. A description of the corresponding methodologies have been provided. Finally, the expected results and possible criticalities of the proposed methodologies have arisen from this preliminary analysis. To summarise, this deliverable aims at supporting the theoretical Level Crossing PoC proposed in Deliverable D3.2 with experimental results. This could answer to the research questions and the expected results that emerged in the previous deliverable and could represent a further step towards the definition of a benchmark for future research inspiration.

The same objectives and methodological goals also hold for the **“Predictive Maintenance for Rolling Stock”** PoC, that intends to explore the capability of exploiting Reinforcement Learning techniques (e.g., Deep Q-Networks) into rolling stock circulation and maintenance scheduling tasks, in order to optimise and enhance the current non-flexible maintenance policy and paradigms. Notably, the topic “Railway Stations using Digital Twins (DT)” proposed at the stage of Deliverable D3.1 was based on the transferability analysis conducted in the same deliverable, which was suggested by the possible research directions from other transport sectors, and it was designed for further exploration with regard to the rolling stock maintenance part from Deliverable D3.2 to Deliverable D3.4. However, as the research proceeded and stepped into Deliverable D3.2, it was realised that no realistic data was available for building up such a DT system. The direction of the PoC has been adjusted to explore the feasibility of “historical trajectory and train movement records used as a solution to participate in the predictive maintenance of chassis of locomotives" in Deliverable D3.2. While with further investigation it turns out the actual implementation costs such as installing sensors on the bogie components are already way more beyond the research scope than we expected. As an alternative, a new but closely relevant direction "AI-assisted rolling stock rostering and maintenance" has been proposed in Deliverable D3.3 under the original topic "Predictive Maintenance for Rolling Stock". The data used to develop this PoC were collected and provided by the Railway Operator Transpennine Express [1] with the non-business usage only, its timetable/operational information are not publicly disclosed. The results obtained by these datasets as part of the research carried out in RAILS, with the sole purpose of optimizing the process of maintenance activities during rolling stock rostering. However, it's essential to note that either the RAILS Consortium or the authors do not have the right to disclose, misuse, or transfer the datasets for any other unauthorized research or commercial activities outside of this PoC.

[1] https://www.tpexpress.co.uk/

2. Objective

This document, in line with the previous deliverables, deals with the following objectives of the RAILS project:

- *Objective 4: Development of methodological and experimental proofs-of-concept*;
- *Objective 5: Development of Benchmarks, Models and Simulations*.

In particular, in the previous deliverables, two pilot case studies have been identified, namely, “Smart Maintenance at Level Crossings” and “Predictive Maintenance for Rolling Stock”; for each of them, experimental PoCs have been addressed to study the feasibility of AI methods in the railway field, and some learning approaches have been identified as a potential solution to develop their respective railway functionalities.

On the basis of the considerations made in Deliverable D3.2, the main goal of this deliverable is to investigate innovative AI models for the considered case studies to be evaluated via test and validation activities, in order to understand how and if the AI approaches identified during the previous tasks can support and enhance rail safety and automation.
Indeed, the ultimate goal of the PoCs is not to propose full solutions to a specific problem but to derive lessons learned and recommendations for future research that will be collected in the next Deliverable D3.4. To this aim, the present document focuses on the following objectives:

- the definition of detailed AI models based on the selected learning approaches;
- the description of the learning process of the proposed methods through a training phase, which can be carried out exploiting specific generated or selected datasets;
- the validation of the proposed models to show their effectiveness;
- a preliminary analysis of the results to highlight the possible benefits and drawbacks of the proposed techniques.

Hence, this study is meant to be a step towards the acquisition of the necessary knowledge to understand the potentiality of AI in railways with a specific focus on maintenance activities. In this direction, the main object of the next deliverable will be an in-depth analysis to identify gaps and opportunities, weaknesses and strengths that emerged from each case study, with the final aim of defining technology roadmaps towards the effective adoption of AI in the rail sector.

3. Introduction

Deliverable D3.3 reports the validation activities of the solutions and approaches identified in Deliverable D3.2. This deliverable provides insights and information on the validity of the research results and the feasibility of the AI approaches investigated in the context of the two pilot case studies preliminary identified in previous deliverables. However, as already specified in Chapter 1, considered the complexity and feasibility of data collection regarding the status of rolling stock components, we have slightly refined the scope of the "Predictive Maintenance for Rolling Stock" PoC.

The heart of this document consists of two main chapters, addressing the aforementioned issues for the two selected pilot case studies: Chapter 4 is devoted to "Smart Maintenance at Level Crossings" and Chapter 5 deals with "Predictive Maintenance for Rolling Stock". The two chapters are organised with similar structure: a brief introduction constitutes Sections 4.1 and 5.1; a detailed description of the proposed AI models is provided in Sections 4.2 and 5.2; Section 4.3 discusses how datasets to develop the "Smart Maintenance at Level Crossings" PoC were generated, while Section 5.3 introduces the details of how the proposed framework is implemented in line with the actual settings and datasets; The training phase together with the validation of the proposed approach in concrete operational scenarios are deeply analysed in Sections 4.4 and 5.4; the results of the validation phase are shown in Sections 4.5 and 5.5; finally, a preliminary discussion of the potential advantages of the proposed approaches is addressed in Sections 4.6 and 5.6.

A critical examination of the work and of the results obtained in this Deliverable, also against the current state-of-the-art in railways, is the object of Deliverable D3.4 ("Report on identification of future innovation needs and recommendations for improvements"). Specifically, the latter document reports lessons learned, weaknesses and strengths shown by each exploited technology, technical and implementation recommendations, unaddressed issues, and innovation needs, with the aim of identifying technology roadmaps for AI integration in the rail sector.

4. Smart Maintenance at Level Crossings

4.1. Introduction

In this chapter, we focus on the implementation of an AI system to continuously monitor Level Crossings (LCs) and analyse their health status while leveraging non-intrusive sensors (i.e., cameras and microphones) [1, 2].

Currently, inspections oriented at evaluating the correct functioning of LCs are typically conducted on the field by following fixed scheduling (see, for example, [3, 4]); however, this approach does not allow timely detection of failures that may occur between two consecutive inspections. This is one of the motivations that, in the last years, contributed to increasing the interest in continuous monitoring systems to collect data related to the health status of LC systems[1,2] in real-time to promptly identify malfunctions.

One of the main issues when it comes to building systems of these kinds is related to the installation of IoT sensors which represent one of the main enablers for real-time monitoring. Intrusive IoT sensors, i.e., applied directly on the asset to be monitored, may lead to two main complications:

- The newest LCs may already be equipped with the latest and adequate sensors, however, there are thousands of "old" LCs, deployed decades ago, which may be not; therefore, this would lead to a massive, expensive, and time-consuming sensorisation.
- Being safety-critical assets, level crossings are subject to strict standards and regulations including, for example, CEN-CENELEC standards for the assessment of the Safety Integrity Level (SIL) [5]. Hence, the introduction of intrusive IoT sensors (if not contemplated by design) could lead to expensive and time-consuming re-approval processes.

In this document, *we intend to provide a contribution towards the continuous monitoring of LCs by means of cost-effective and non-intrusive sensors*. Particularly, we will focus on **fixed** camera(s) and microphone(s) as: i) they are non-intrusive, i.e., they can be installed without compromising the functioning of the LCs; ii) they may be already installed at LCs for surveillance purposes, therefore, we could have some advantages related to technology reuse. Actually, the same applies in case they will be installed for maintenance purposes and leveraged, in the future, for safety-related aspects (e.g., obstacle detection).

To summarise the scope of this case study, **we are intended to understand whether DL approaches can be exploited in combination with data from fixed cameras and microphones to identify the health status of any kind (old and new) of LCs**. This can also be formalised into three Research Questions (RQs) [2]:

RQ1: How can we automatically detect anomalies in LC devices, such as Barriers, Warning Lights and Warning Bells, for maintenance purposes, by using AI applied to possibly existing LC surveillance cameras and microphones?

RQ2: How can we generate relevant datasets for Warning Lights, Barrier movement, and/or

[1] https://www.smartmotors.org/level-crossing-monitoring

[2] https://www.voestalpine.com/railway-systems/en/products/rxm-rail-crossing-monitoring/

Warning Bell anomaly detection based on Deep Learning?

RQ3: Can we demonstrate possible answers to RQ1 and RQ2 through a simple proof-of-concept demonstrator in order to inspire future developments and a technology roadmap?

Notably, the fact that the camera is fixed, i.e., its Field of View (FoV) does not change in time, facilitates the implementation of the AI system as discussed in the following of this chapter. Hence, it may be seen as a requisite, not so distant from reality, necessary for the physical implementability of the proposed approaches. Nevertheless, we will also discuss a few alternatives in case the camera could not be fixed or the FoV accidentally changes due to, for example, atmospheric phenomena.

To conclude, in Deliverable [2], we also highlighted some Key Performance Indicators (KPIs) we will consider to provide a high-level evaluation of the monitoring system we discuss in this chapter. These KPIs are reported in Table 4.1.

Table 4.1: KPIs for the evaluation of LC continuous monitoring systems (reported from [2]).

KPI	Description
KPI1	**Implementability Costs**. Includes costs of sensors, installation, possible re-approval processes, and consumption.
KPI2	**Computation Time**. Indicates the time required by the system to detect possible malfunctions.
KPI3	**Effectiveness**. Indicates how the monitoring system would perform in terms of accuracy and detectable malfunctions.

4.2. Model Description

From a high-level perspective, Level Crossings can be subdivided into two macro-categories: *Passive LCs*, which involve neither warning signals (warning bells or warning lights) nor barriers, and *Active LCs*, which can involve both warning signals and barriers. In addition, *Active LCs* can be *Automatic*, i.e., the barriers are automatically triggered by the approaching train, or *Manual*, i.e., the barriers are manually activated by human operators. In this document, we focus on the realisation of an AI system capable of working with any of these types of LCs (especially if unattended). To that aim, we considered the most comprehensive case, i.e., that involving all the three macro components mentioned above: warning bells (WBs), warning lights (WLs), and barriers. All these components must work properly together respecting particular constraints (e.g., [6]) to ensure safe operability.

Our practical goal was to build a *multi modular architecture*, schematised in Fig. 4.1, where each module focuses on the monitoring of a specific LC macro component leveraging audio and video data. Worth highlighting, **this framework has the great advantage of allowing us to consider the whole Level Crossing System as a black-box, focusing only on its behaviour, i.e., just evaluating its visible and audible output and estimating its deviations from the expected/nominal functioning; therefore, in our view, it could be easily adapted to different kinds level crossings.**

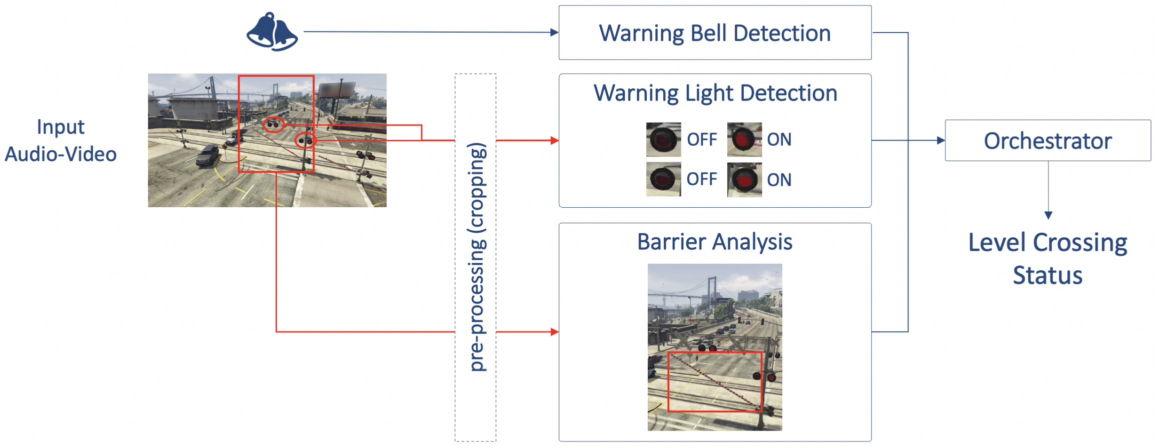


Fig. 4.1. High-Level Architecture for Level Crossing Intelligent Monitoring. Image extracted from Grand Theft Auto V (GTA V)[3].

4.2.1. Warning Bell Detection Module (WBDM)

This module will be oriented at detecting the sound activity of WBs. From a theoretical perspective, a ML-based audio detection system can be implemented as a binary classification problem by taking into account two main classes: one representing the "sound activity" and the other representing "no sound activity". The main problem with this kind of approach is that it could lack robustness. Indeed, when it comes to LCs, there are some kinds of sounds/alarms (e.g., car alarms, car horns and police/ambulance sirens) that could be "confused" with WBs. Therefore, we decided to address the WB detection as a three-class classification problem by building a dataset consisting of three classes: Level Crossing *Warning Bell*, which represents the situation when the WB is active; *No Alarm*, which represents the situation when the WB is silent; and *Generic Alarm*, which includes sounds that could be confused with WBs. As better discussed in Section 4.3.1, the datasets for our experiments has been built we built by leveraging AudioSet[4] [7].

In the same way, to implement the classification approach, we exploited VGGish[5] [7], a Convolutional Neural Network (CNN) originally conceived to extract 128-dimensional features from AudioSet. We choose this network as our baseline for two main reasons: i) it has already been used for audio analysis; and ii) the authors made available online a set of pre-trained weights obtained by training VGGish on a preliminary version of YouTube-8M[5,6] which allowed us to analyse the potential of Transfer Learning in this context. More details are given in Sections 4.4.1 and 4.5.1.

4.2.2. Warning Light Detection Module (WLDM)

For the implementation of this module, which would be oriented at detection and/or classifying the WLs, we identified two main alternatives.

The first one does not take into account the main peculiarity of our scenario, i.e., the fact that the *camera is fixed*. In this case, it would be possible to exploit object detection approaches to both locate the WLs within the video frames and classify them into "turned on" (active)

[3]https://www.rockstargames.com/it/gta-v
[4]https://research.google.com/audioset/download.html
[5]https://github.com/tensorflow/models/tree/master/research/audioset/vggish/
[6]https://research.google.com/youtube8m/index.html

or “turned off” (idle). In the literature, especially in the automotive field, there have been proposed various solutions for traffic lights detection (i.e., localisation and classification) based on DL object detectors like region-based CNNs (R-CNN) [8] and YOLO [9]. Given that the LC WLs are nothing more than a specific (simplified) kind of traffic light, it is not so visionary to assume that such approaches could work properly also in this scenario.

On the other hand, the second alternative would be to leverage the fact that the camera is fixed. In this case, given that the WLs occupy always the same position within all the video frames, it would be possible to first crop them out, and then apply a CNN to classify them as active or idle. To that aim, it would be possible to leverage potentially any kind of CNN.

Despite the fact that, when it comes to AI, it is not easy to select the best approach a-priori; in this case, a classification approach would be preferable for three main reasons:

- First, they would be much easier to implement as CNNs oriented at image classification are typically less complex (architecturally speaking) than object detectors. Indeed, the latter usually involves an additional mechanism to dynamically extract the region of interest (i.e., locate the object within the image), hence, it introduces a complexity that could be easily solved without applying any DL approach if the condition of the fixed camera holds. In the latter case, it would be possible to apply an extremely trivial pre-processing step to statically crop out the WLs from the frames.
- Second, it would be easier to build a dataset for image classification rather than for object detection. Indeed, to train an object detector, we would need a labelled dataset that, for each of the WLs, indicates the position within each frame (i.e., the bounding box) and the class (i.e., active or idle). On the other hand, the dataset for the classification approach would simply be composed of two collections of images: active WLs and idle WLs. The first process would be much more time-consuming than the second one.
- Third, some object detectors may suffer from complex backgrounds and the fact that WLs typically occupy a small portion of the image (small-scale object detection problem [10]). Differently, by statically cropping the WLs and then simply classifying them through an image classification approach, we would overcome these potential issues.

Given the large attention that image classification has received since 2012, and given the fact that the binary image classification object of this task could be addressed by simply applying one of the state-of-the-art CNNs [11, 12], in the following of this chapter we will mainly focus on the WBDM and the Barrier Analysis module (discussed below) as it seems that limited work has been performed in these directions [1, 2].

To conclude, it is worth noting that it would be possible to leverage some datasets available online to pre-train and/or build classifiers to discern between active and idle WLs. Some of them are oriented to object detection, hence, they also encompass bounding box annotations; anyhow, it would be quite trivial to extract patches from these data to build a suitable dataset for the classification approach. Some examples are the LISA Traffic Light Dataset[7], the Bosch Small Traffic Lights Dataset[8], and the SJTU Small Traffic Light Dataset[9], but also more generic and well-known datasets as COCO[10].

[7] https://www.kaggle.com/datasets/mbornoe/lisa-traffic-light-dataset
[8] https://hci.iwr.uni-heidelberg.de/content/bosch-small-traffic-lights-dataset
[9] https://github.com/Thinklab-SJTU/S2TLD
[10] https://cocodataset.org/#explore

4.2.3. Barrier Analysis Module (BAM)

This module aims at detecting the barrier(s) within video frames and extracting its motion features, i.e., data representing the movement of the barrier over time. As far as we know, also basing on our previous analyses [1, 2], limited or no work has been carried out in the literature in this direction.

To implement this module, we decided to adopt an object detector, a YOLOv5[11] architecture to be precise, to detect the barrier within adjacent frames. As we will see in the following, after a pre-processing step which leverages the fact that the camera is fixed, each frame contains a single barrier, therefore, for each frame, the Deep Neural Network (DNN) produces in output a single bounding box containing the barrier. By plotting the heights of the bounding boxes extracted from the different frames, we could eventually obtain a representation of the movement that the barrier traces over time. Then, by comparing the nominal movement (i.e., the correct movement of the barrier) with that detected at run-time, we could understand if the barrier is behaving properly or it is defective. These analyses are detailed in Section 4.4.2.

4.2.4. Orchestrator

The last module, the Orchestrator, will gather and analyse the outcomes of the three modules discussed above in order to establish the health status of the whole LC system. In addition, this module could also integrate signals and information coming from other monitoring devices. Briefly, we can assume that railway network managers know the position of the train or, at least, if the LC should have been triggered given the position of the train. This information is extremely valuable if integrated within the proposed system as it could help to time the activation of LC components and filter out possible false positives or false negatives in output to AI algorithms. Detailed examples are given in Section 4.6.

4.3. Dataset Generation

As highlighted in the previous sections, the aim is to delineate the health status of a LC by considering only audio and video data. Since we did not find pre-existing *suitable* datasets, we leveraged some online repositories and real-life simulators to build our datasets to train and test the DNNs.

4.3.1. Audio Data Collection - Warning Bell Dataset (WBD)

As introduced in Section 4.2.1, we decided to address the WB detection task as a three-class classification problem by building a dataset consisting of three classes: Level Crossing *Warning Bell*, which represents the situation when the WB is active; *No Alarm*, which represents the situation when the WB is silent; and *Generic Alarm, which includes sounds that could be confused with WBs.*

To generate this dataset, we leveraged the AudioSet dataset [7], a collection of about 2 million hand-labelled audio clips, most of which have a duration of 10 seconds, extracted from YouTube[12] videos and organised according to a specific ontology in more than 500 classes [13]. Then, we also exploited the rationale behind AudioSet to collect additional samples directly from the YouTube platform.

[11] https://github.com/ultralytics/yolov5
[12] https://www.youtube.com

On these bases, the samples for the three aforementioned classes have been gathered as follows:

- **Warning Bell (WB)** samples have been collected from YouTube by following AudioSet's rationale. We manually checked several LC videos and extracted one or more 10-second audio clips from each of them; we took into account WBs ringing at different intensities and videos published after 2019.

  Provided that LCs from different railway infrastructure operators could have different WB sounds, building a global classifier working for each of them would be quite a hard task as hundreds (or even thousands) of samples must be collected for each kind of sound (i.e., for each type of WB). In addition, ad-hoc classifiers may also perform better than a unique general-purpose classifier as they would be specialised to detect a specific WB sound; it would most likely be counterproductive to "disturb" a classifier with features related to LCs of type Y if it would be mainly used to detect sound activities in the context of LC of type X.

  Hence, for the sake of feasibility, and to study the performance of the selected CNN in, at least, two possible scenarios, we focused only on Italian (IT) and British (UK) LCs. Therefore, we manually selected a total of 330 10-second audio clips subdivided into 165 IT WBs and 165 UK WBs.
- **No Alarm (NA)** samples were gathered by following the same procedure discussed above. In this case, we collected 350 10-second audio clips related to background noises both in the city and the countryside (e.g., traffic noise at different intensities, wind, birds), excluding any kind of alarm (e.g., car horns);
- **Generic Alarm (GE)** samples have bee obtained by leveraging the AudioSet dataset. We selected audios related to 19 classes among the available ones including *sirens* (e.g., ambulance, emergency vehicle), *alarms* (e.g., car alarms, smoke detectors), and *bells* (e.g., church bells, bicycle bells). These classes have been chosen as UK WBs are similar to sirens or alarms; differently, IT WBs are similar to some kinds of bells. Hence, given that those sounds are recordable in the vicinity of LC, they could be confused with LC WBs. Relevant samples were manually processed to filter out duplicates and to remove low-quality ones. After this process, 500 samples were kept in the GE class.

Downstream the data collection process, we obtained a dataset composed of 1180 audio clips distributed as shown in Fig. 4.2. These data compose the Warning Bell Dataset (WBD) which has been used to train, validate, and test the CNN we selected to implement the WBDM as discussed in Section 4.4.1.

4.3.2. Video Data Generation

As for the previous case, we were not able to find suitable video datasets to train and test the DDNs composing the WLDM and the BAM. By *suitable*, we mean videos collected by using a *fixed camera* under *different light and weather conditions*.

We have already discussed the advantages introduced by the fixed camera in Section 4.2.2 when it comes to WL detection. Actually, it could help to improve performances also in the context of Barrier Analysis. Indeed, it would be possible to apply a pre-processing also in this case to crop out, from the video frames, the area that the barrier could occupy when moving. Hence, the YOLOv5 network, instead of "looking for" the barrier within the whole

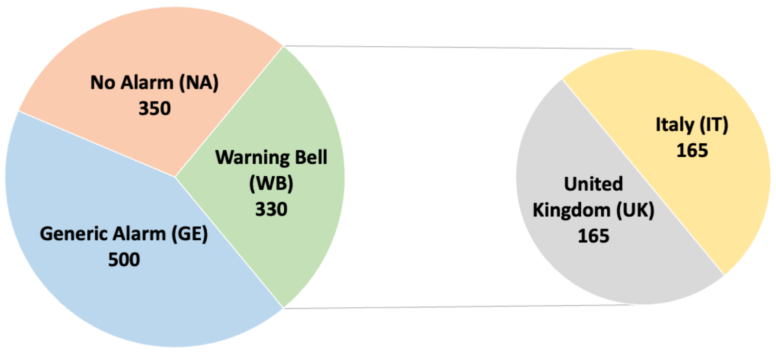


Fig. 4.2. Distribution of the Collected Audio Samples.

frame, will focus on a specific area only. In addition to that, another action that could be taken in advance to theoretically improve detection performances is to collect all the data from the same LC with the same camera angle, and, obviously, under different light and weather conditions. Indeed, although it would be possible to build[13] a generic AI model working for any LC, it would be advisable to build an AI model for each LC by considering data coming from that specific LC only. In this way, by training the network with several video frames (properly labelled) depicting the barrier in all possible positions and angles it can take (with various light and weather conditions), it is not visionary to assume that, at run-time, the model will analyse frames which would be extremely similar (yet not identical) to those it has seen during the raining phase; a fact that would most likely benefit the detection performance.

To build a dataset encompassing these characteristics, it would be quite tricky to leverage videos extracted from, for example, the YouTube platform since users usually collect them through hand-held cameras and each video would be related to a different LC. Hence, we decided to exploit a **Real-life Simulator**, Grand Theft Auto V (GTA V)[14], to build a suitable dataset for our case study.

By considering always the same LC in GTA V, we collected 8 videos by varying weather and light conditions. Fig. 4.3 shows a frame for each of the videos we extracted which can be classified into 8 types (referred to as "GTA Types" in the following): i) Day Clear (DC); ii) Day Rain (DR); iii) Day After Rain (DAR); iv) Day Snow (DS); v) Night Clear (NC); vi) Night Rain (NR); vii) Night After Rain (NAR); viii) Night Snow (NS).

Then, as depicted in Fig. 4.4, by exploiting the fact that the camera is fixed, it would be possible to crop out the WLs and the barriers to respectively build the Warning Light Dataset (WLD) and the Barrier Dataset (BD). Notably, we extracted the region of one of the barriers closer to the "camera"; given that the barriers on the other side of the LC would be too far from the camera to be properly recorded, it would be advisable, in some cases, to use two cameras to monitor the LC, one from each side.

As introduced in Section 4.2.2, given the large attention image classification has received during the last decade, we decided to focus mainly on the BAM, instead of the WLDM.

Most of the frames extracted from GTA videos depict *open* or *closed* barriers; frames con-

[13]By "build" we mean training an AI model on a specific dataset.
[14]https://www.rockstargames.com/it/gta-v

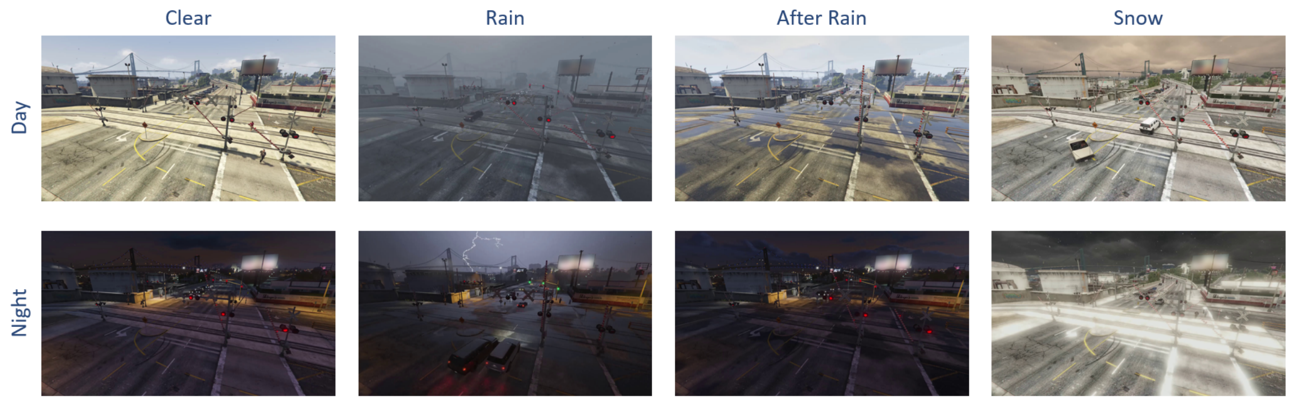


Fig. 4.3. Frames extracted from GTA Videos.

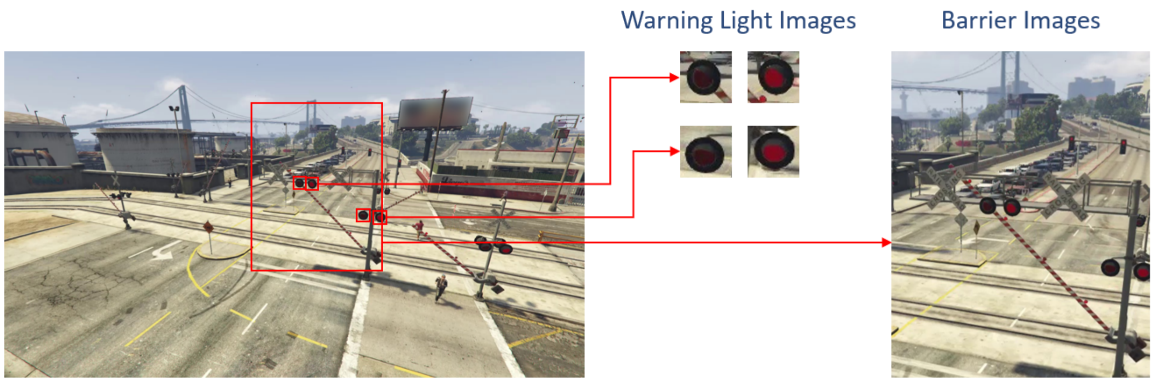


Fig. 4.4. Pre-processing (Cropping).

taining *moving* barriers are quite rare compared to the others. Hence, we manually selected only 240 frames for each GTA Type trying to obtain a balanced distribution between open, closed, and moving barriers not to have an unbalanced dataset. Then, the initial BD contained 1920 frames (240 $\times$ 8 videos) that we manually labelled by leveraging the LabelImg tool[15]. The labels, in this case, contain information related to the position of the rectangle (bounding box) containing the barrier. Then, to increase this dataset in terms of the number of samples, and introduce some more light and weather aleatory, we applied some **Data Augmentation** by leveraging the Automold python library[16].

Worth highlighting, the Automold library offers different transformations from which we selected only those that could be suitable for our case study (also taking into account that the GTA Types already include some kind of diversity in terms of whether and light conditions); Table 4.2 summarises the Automold transformations we adopted. Notably, most of these transformations introduce some randomness, for example: the “Hole” spawns an ellipse of a random colour whose centre is located in a random position; the “Rain” produces drops with random intensity and angle; the “Bright” increases the brightness of the pixels according to a random factor; and so on. Hence, it is quite unlikely that a given transformation could produce identical frames.

So, to build our final BD, we applied the aforementioned Data Augmentation transformations

[15]https://github.com/heartexlabs/labelImg
[16]https://github.com/UjjwalSaxena/Automold--Road-Augmentation-Library

Table 4.2: Types of Data Augmentation Transformations

Transformation	Acronym	Description
Bright	BR	Randomly increase the brightness of the image
Dark	DA	Randomly decrease the brightness of the pixel
Rain	RN	Simulate rain by randomly adding drops with different intensities and angulation
Snow	SN	Simulate snow by randomly increasing the brightness of specific pixels
Fog	FO	Simulate fog by randomly blurring parts of the image
Shadow	SH	Simulate shadows by adding random semitransparent masks on the image
Sun Flare	SF	Simulate sun flares by adding concentric and random circles in the image
Hole	HO	Introduce noise by randomly introducing coloured ellipses within the image
Gravel	GR	Introduce noise by simulating a random gravel pattern in the image

on each frame contained in the initial BD. Table 4.3 shows the mapping between the GTA Type and the kind of transformation we applied for each specific Type.

Table 4.3: Mapping between GTA Types and Transformation for BA-DS

		Transformation								
		BR	*DA*	*RA*	*SN*	*FO*	*SH*	*SF*	*HO*	*GR*
GTA Type	*DC*	x	x	x		x	x	x	x	x
	DR	x	x	x		x	x	x	x	x
	DAR	x	x	x		x	x	x	x	x
	DS	x	x	x	x	x	x	x	x	x
	NC	x	x	x	x	x	x	x	x	x
	NR	x	x	x	x	x	x	x	x	x
	NAR	x	x	x	x	x	x	x	x	x
	NS		x	x			x	x	x	x

Notably, there are some missing combinations. This is because the “Snow”, both as GTA Type (especially in NS) and Transformation (i.e., SN), makes the images so bright that the barrier disappears in the frame. Also, some of the frames generated through the SN transformation seem to be distant from what would happen in reality. However, we kept some of these “noisy” frames in the dataset not to obtain a too ideal representation of reality; indeed, sub-classes like DS$\times$BR (i.e., DS frames augmented through the BR transformation) or the NS itself do contain some complex frames. Fig. 4.5 shows some examples.

Lastly, for the sake of completeness, Fig. 4.6 visually shows the effects of the transformations by taking DC frames as reference frames.

Downstream this process, we obtained 74 sub-classes (8 GTA Types and 66 combinations between GTA Types and Data Augmentation transformations). We got 240 frames for each of these sub-classes, hence, the dataset is composed of 17760 images in total. These images were then subdivided into training (12432 images, 70%), validation (2664 images, 15%), and test (2664 images, 15%) sets. Important to mention:

- The labels generated manually for, e.g., the DC GTA Type, have also been used for all the DC$\times$XX sub-classes obtained by combining the DC with the XX transformations

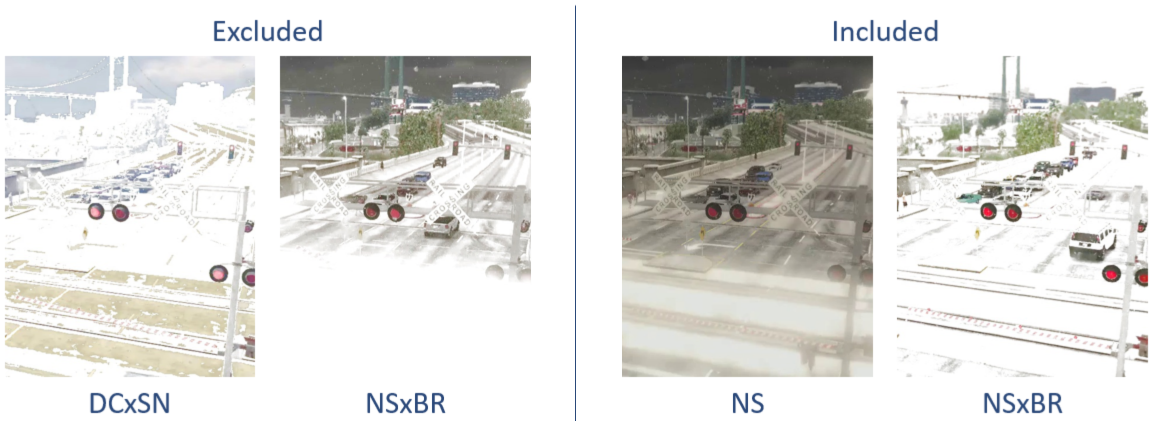


Fig. 4.5. Excluded vs Included Transformations.

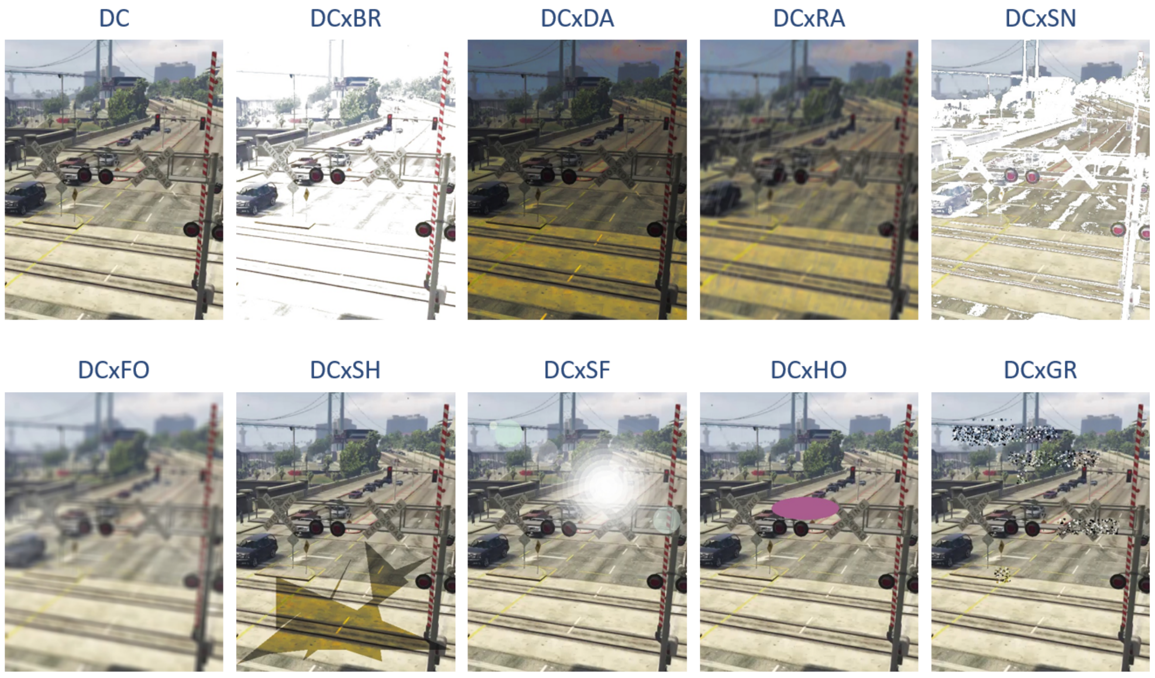


Fig. 4.6. Examples of Augmented Frames.

reported in Table 4.3.

- The ratio 70-15-15% has been also kept at the sub-class level to make the dataset as balanced as possible. In other words, for each sub-class, 168 over 240 frames belong to the training set, 36 frames belong to the validation set and the remaining 36 to the test set.

4.4. Training and Validation

This section discusses the training and validation phases of the DNNs we adopted to implement the WBDM and the BAM.

4.4.1. Warning Bell Detection Module

Recalling what was discussed in Section 4.2.1, to implement this module, we adapted the VGGish [7], a CNN originally conceived to extract 128-dimensional features from AudioSet data, starting from some TensrFlow/Keras (Python) implementations available on GitHub[17,18]. Important to mention, the authors of VGGish also made available online a set of weights obtained by training the network on a preliminary version of YouTube-8M that we used to pre-train the DNN and apply Transfer Learning.

We will show the performance of the network according to both these approaches, however, before proceeding with the training and the application of TL, we needed to pre-process the collected audios to make them compliant with the input layer of the VGGish's Feature Extractor.

4.4.1.1. Dataset Preparation

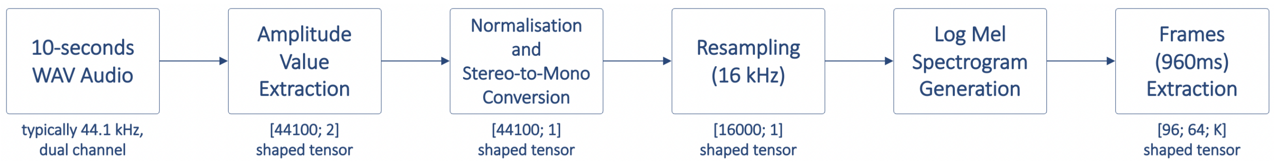

10-seconds WAV Audio
typically 44.1 kHz, dual channel
Amplitude Value Extraction
[44100; 2] shaped tensor
Normalisation and Stereo-to-Mono Conversion
[44100; 1] shaped tensor
Resampling (16 kHz)
[16000; 1] shaped tensor
Log Mel Spectrogram Generation
Frames (960ms) Extraction
[96; 64; K] shaped tensor

Fig. 4.7. Audio pre-processing.

VGGish takes in input images with a specific size ([96; 64; 1]) representing the Logarithmic (Log) Mel spectrogram of a portion of the audio. To transform the audios into images of this kind, we leveraged the same pre-process performed by the VGGish's authors (shown in Fig. 4.7) which goes through the following steps:

- Extract the amplitude values from the 10-second WAV audios collected as discussed in Section 4.3.1; these amplitudes are saved as [44100, 2] shaped tensors given the sampling frequency of 44.1 kHz and the stereo (dual channel) acquisition.
- Normalise these values between [-1;1] and then apply the conversion from stereo to mono by averaging all the values along the channel dimension.
- Re-sample the obtained data according to a sample rate of 16000Hz to obtain [16000; 1] shaped tensors.
- The tensors are then transformed into Log Mel Spectrogram as described in [7].
- Lastly, the obtained spectrograms are subdivided into K non-overlapping frames of shape [96; 64; 1], where K, with some approximations, is equal to the length of the audios in seconds (typically 10 seconds). Important to mention, the subdivision could not be perfect, i.e., there might be some remaining data at the end of the Log Mel spectrogram which do not fit the [96; 64; 1] shape; these data are dropped and not used.

Downstream this process we obtained, for each audio clip, K frames each of which represents 960ms (i.e., roughly 1 second) of the original audio. We considered these 1-second frames as the base unit in our training, validation, and test phases, by associating them with

[17] https://github.com/tensorflow/models/tree/master/research/audioset/vggish/
[18] https://github.com/beasteers/VGGish

the same label of the source audio they were extracted from. It is worth noting that the first and the last 1-second frames may not be representative of the class because of possible noises or delays in starting or ending the recording. Therefore, we decided to trim these frames out from each audio sample in the Training set. Fig. 4.8 shows an example of Log Mel spectrogram framing and trimming.

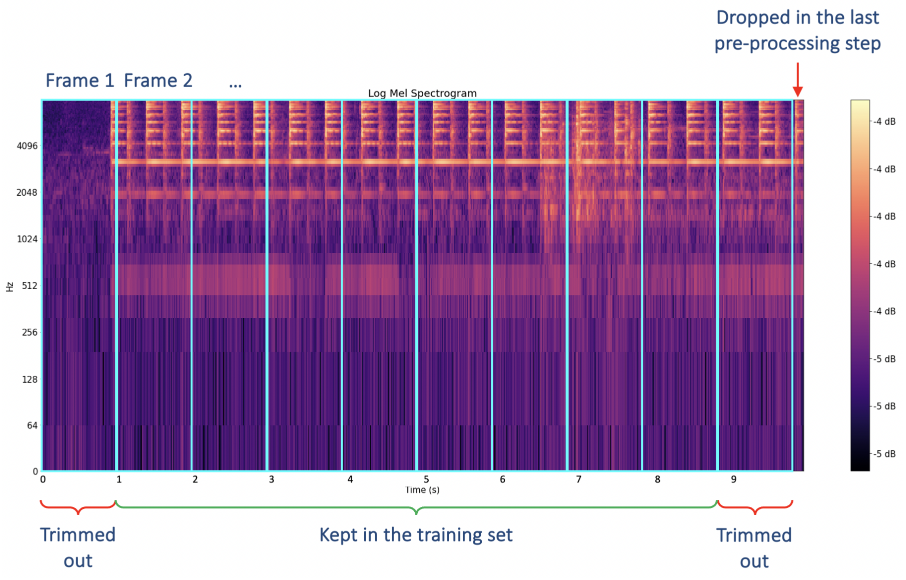


Fig. 4.8. Log Mel Spectrogram Framing and Trimming.

To summarise, we first collected the WBD described in Section 4.3.1, which from now on we will refer to as **Audio-level** dataset, and subdivided it into training (876 audio clips, ∼74%), validation (152 audio clips, ∼13%), and test (152 audio clips, ∼13%) sets. Notably, IT and UK WBs have been subdivided by considering the same distribution, i.e.: 123 IT WBs (∼74%) in the training set, 21 IT WBs (∼13%) in the validation set, and 21 IT WBs (∼13%) in the training set; exactly the same goes for UK WBs. Then, we applied the pre-processing described above to obtain the **Frame-level** training, validation, and test sets. To be specific, the frames in the Frame-Level training set have been obtained by pre-processing all the audios contained in the Audio-Level training set; the same goes for the Frame-Level validation and test sets. Therefore, there is no data duplication and each audio/frame has been considered in only one of the aforementioned Audio-Level or Frame-Level sets.

Table 4.4 shows the distribution of the samples for training, validation, and test sets at both the audio and the frame levels. The table also reports two additional Frame-Level sets: the *Trimmed Training Set (TTS)* and the *Trimmed Joint Set (TJS)*. The former represents the Frame-Level Training Set after the aforementioned trimming operation; similarly, the latter represents the set obtained by merging the Frame-Level Training Set and the Frame-Level Validation Set (FVS) after the trimming operation.

4.4.1.2. Training and Validation

Starting from the original VGGish[19] architecture, which high-level structure is shown in Fig. 4.9, we tested different configurations by varying some parameters and hyperparameters.

[19] https://github.com/tensorflow/models/tree/master/research/audioset/vggish

Table 4.4: Audio-Level and Frame-Level Sets.

	Audio-Level Sets			**Frame-Level Sets**				
	Training	*Validation*	*Test*	*Training*	*Trimmed Training (TTS)*	*Validation (FVS)*	*Test*	*Trimmed Joint (TJS)*
WB	246*	42*	42*	2460*	1968*	420*	420*	2304*
NA	260	45	45	2600	2080	450	450	2440
GE	370	65	65	3628	2889	641	620	3400
Total	876	152	152	8688	6937	1511	1490	8144

* Exactly half of these are related to IT WB audio clips, while the other half are related to UK WB audio clips.

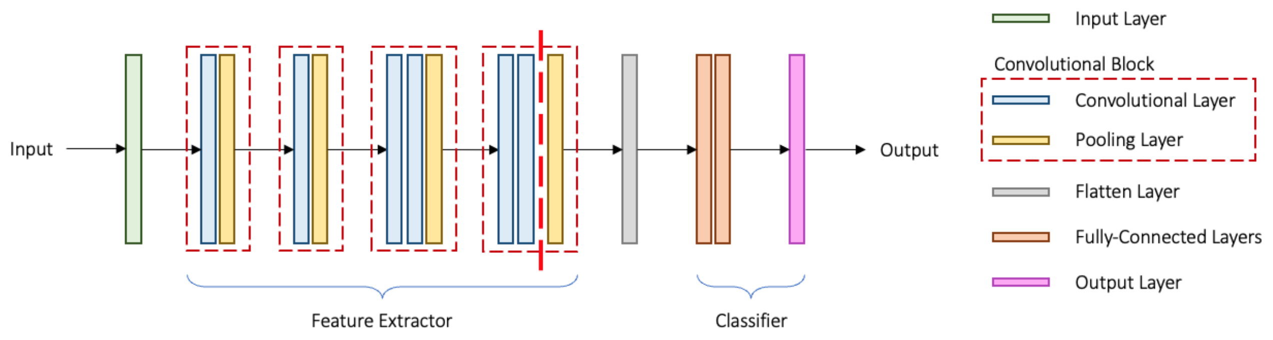


Fig. 4.9. VGGish High-Level Architecture.

The original VGGish consists of: four **Convolutional Blocks** composing the *Feature Extractor*, a **Flatten Layer**[20]and three **Fully-Connected (FC) Layers** (the last of which is named *Output Layer*) composing the *Classifier*. Important to mention, VGGish does not produces class scores in output, instead, it extracts a one-dimensional array with 128 elements which is an encoded representation of the sample in input; these layers were substituted with other FC Layers to create our classifier.

Therefore, to build our CNN, we discarded all the layers to the right of the vertical dashed red line in Fig. 5. Notably, besides the FC Layers (usually dropped when reusing CNNs according to the Transfer Learning paradigm, see Fig. **??**), we also dropped the *Pooling Layer* of the fourth Convolutional Block. This is because, in our tests, we also exploited some specific pooling layers that directly apply the flattening operation; the selection of a suitable Pooling/Flatten Layer was then considered as a hyperparameter of the network.

Taking into account what discussed so far, to define a suitable VGGish-based CNN for our task, we trained and validated different configurations of the original VGGish by leaving unchanged the composition of the Layers to the left of the dashed red line in Fig. 5, but varying the parameters and hyperparameters reported in Table 4.5. The different types of Pooling/Flatten Layer have the following characteristics:

- *global_pooling*: as proposed by the VGGish authors, it is a combination of a Max-Pooling Layer (with 2×2 pool size and 2×2 stride), which produces a [6; 4; 512] shaped

[20]From a theoretical perspective, this is not properly a layer as it does not involve any parameter. It simply takes in input the feature extracted from the Convolutional Blocks and transforms them into a one-dimensional array so that they can be processed by the Fully-Connected Layers.

tensor, and a GlobalAveragePooling2D[21] Layer, which performs an avg-pooling with 6×4 pool size and no stride producing a $[1; 512]$ shaped tensor (the flatten operation is "implicit" as the pooling filter has the same height and width of the tensor in input);

- *max2_flatten*: a max-pooling layer with 2×2 pool size and 2×2 stride combined with a Flatten Layer, producing a [1; 12288] shaped tensor;
- *max4_flatten*: a max-pooling layer with 4×4 pool size and 4×4 stride in combination with a Flatten Layer, producing a $[1; 3072]$ shaped tensor;
- *avg4_flatten*: an avg-pooling layer with 4×4 pool size and 4×4 stride in combination with a Flatten Layer, producing a [1; 3072] shaped tensor.

Table 4.5: Network's Hyperparameters and Parameters.

Hyperparameters	**Values**
Number of FC Layers	1, 2
Neurons per FC Layer	1024, 512, 256, 128 (and related combinations in case of 2 FC layers)
Last Pooling/Flatten Layer(s)	global_pooling, max2_flatten, max4_flatten, avg4_flatten

Parameters	**Values**
Learning Rate	10^{-4}, 10^{-5}
Batch Size	20, 60, 100

Then, tests to select the best configuration were performed in two steps: first, we kept the parameters fixed to the underlined values in Table 4.5 and varied the hyperparameters; second, we chose the best-performing configurations obtained at the precedent step and, by keeping the hyperparameters fixed, we proceeded towards the tuning of the parameters. Important to underline, all these tests were performed by *loading and freezing the pre-trained weights* of the VGGish to avoid fluctuations as better detailed in Section 4.5.1.1. Also, the following parameters and hyperparameters were left unchanged for all tests:

- Adam Optimizer[22] with fixed *epsilon* equals to 10^{-8};
- ReLU as the activation function for all the neurons of the FC layers (output layer excluded);
- An Output Layer composed of three neurons with softmax as activation function. We considered three neurons as we adopted the one-hot coding for the classes: [1,0,0] for the WB class, [0,1,0] for the NA class, and [0,0,1] for the GE class.

Finally, we used the Categorical Cross-Entropy[23] as loss function and the Categorical Accuracy[24] as evaluation metrics. For the sake of simplicity, in the following, we will trivially refer to them as "loss" and "accuracy" respectively.

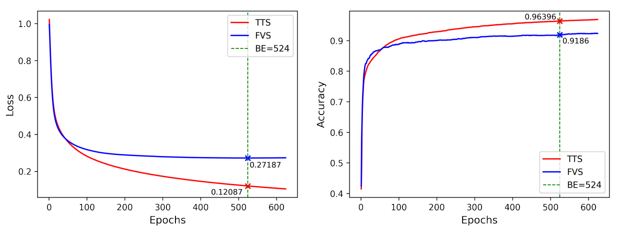
[21] https://keras.io/api/layers/pooling_layers/global_average_pooling2d/
[22] www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam
[23] www.tensorflow.org/api_docs/python/tf/keras/losses/CategoricalCrossentropy
[24] www.tensorflow.org/api_docs/python/tf/keras/metrics/CategoricalAccuracy

Step 1: Tuning of Hyperparameters. We performed experiments by training 56 different configurations (combinations of hyperparameters) on the TTS (fifth column in Table 4.4) and evaluating them on the FVS (sixth column in Table 4.4). In this phase, each configuration was trained for a maximum of 2000 epochs and an Early Stopping criterion was adopted so that if the loss computed on the FVS (FVS loss) did not improve for 100 consecutive epochs, the training would have stopped.

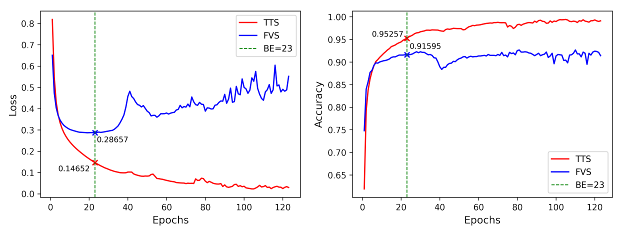
Among all the 56 configurations, we selected those that i) did not evidently diverge and ii) did not present heavy oscillations. All configurations with "max2_flatten" or "max4_flatten" as the Pooling/Flatten Layer were not compliant with these characteristics; indeed, they either diverged evidently or presented accentuated oscillations indicating that the network was not *stable* enough during the training phase. The same goes for the configurations with a "avg4_flatten" Pooling/Flatten Layer and two FC Layers.

Important to underline, with "stable" we mean that, starting from a specific epoch which varies according to the configuration, the variation of the FVS loss is not significant over time. For the sake of knowledge, Fig. 4.10 reports an example of "stable" configuration and an example of "unstable" configuration; the second one is also characterised by heavy oscillations.

All the configurations that survived this screening are reported in Table 4.6 sorted in ascending order in relation to the FVS loss. The majority of these configurations reached a sort of plateau in terms of FVS loss; their divergence is not that significant and they can be considered quite equivalent in terms of stability. Hence, we selected about half of the "survived" configurations by setting a threshold in terms of FVS loss. All configurations that, at the Best Epoch (BE), had a FVS loss lower than 0.28 (highlighted in yellow in Table 4.6) were taken into account for a deeper analysis within the following step.



(a) Stable Configuration: avg4_flatten, one FC Layer with 128 neurons.



(b) Unstable Configuration (with evident oscillations): max2_flatten, two FC Layers (512 and 128 neurons).

Fig. 4.10. Examples of stable and unstable configurations.

Step 2: Tuning of Parameters We then optimised the parameters of the survived configurations by performing 40 additional training by varying the learning rate $\{10^{-4}, 10^{-5}\}$ and the batch size $\{20, 60, 100\}$. In this case, we set the maximum number of epochs to 20000 while the patience for the Early Stopping criterion was set to 300 epochs. Table 4.7 reports the performance of the configurations we analysed in this step; all of them resulted to be very stable and can be considered almost equivalent. Given that, to select the best configuration, we took into account those that achieved the lowest FVS loss (highlighted in yellow in the Table) and, among them, we considered the one that had the lowest FVS loss (0.26866) in combination with the highest FVS accuracy (92.22%). The model that achieved

Table 4.6: Results of the Best Performing Configurations at Step 1. Fixed parameters: Batch Size = 20 and Learning Rate = 10^{-4}.

Pooling/Flatten Layer	Number of FC Layers	Number of Neurons per FC Layer	Lowest FVS loss	Best Epoch	FVS Accuracy	TTS Accuracy	TTS Loss
avg4_flatten	1	128	0,271869212	524	0,918596983	0,963961363	0,120873004
global_pooling	1	1024	0,276927322	640	0,915287912	0,942194045	0,171911165
global_pooling	2	1024, 256	0,277950883	266	0,917935133	0,941761553	0,165159345
global_pooling	1	256	0,278084040	1241	0,915287912	0,942049861	0,169504464
global_pooling	1	512	0,278849244	833	0,914626062	0,940320015	0,176325157
avg4_flatten	1	256	0,279265702	328	0,917935133	0,961366594	0,131476372
global_pooling	2	256, 128	0,279277980	317	0,914626062	0,938734353	0,180421114
avg4_flatten	1	1024	0,279867083	144	0,910655200	0,949834228	0,158503562
avg4_flatten	1	512	0,280445457	256	0,916611493	0,958627641	0,133938462
global_pooling	2	512, 256	0,280517995	244	0,913964272	0,940608323	0,175798401
global_pooling	2	512, 128	0,280727476	340	0,919258773	0,943203092	0,166194081
global_pooling	2	1024, 128	0,281716824	323	0,919258773	0,945365429	0,163153335
global_pooling	2	1024, 512	0,283945471	154	0,916611493	0,931814909	0,195582137
global_pooling	1	128	0,286066711	1944	0,907346129	0,941473246	0,172205478
global_pooling	2	256, 256	0,287442833	252	0,908669770	0,932679832	0,195510745
global_pooling	2	512, 512	0,288879961	228	0,913964272	0,939166784	0,178854913
global_pooling	2	1024, 1024	0,288992524	143	0,914626062	0,932968140	0,188183978
global_pooling	2	128, 128	0,297417551	325	0,907346129	0,929652572	0,204037294

these performances had a single FC Layer with 1024 neurons and the avg4_flatten as the Pooling/Flatten Layer and was trained for 1663 by using a fixed Learning Rate of 10^{-5} and a Batch Size of 20. Fig. 4.11 depicts its training curves.

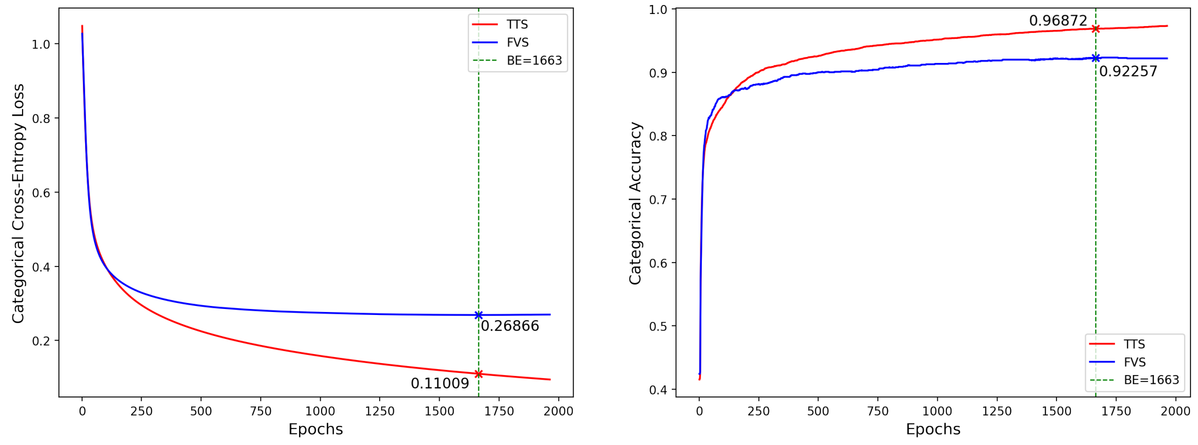


Fig. 4.11. Accuracy and loss trends of the Best Configuration.

Additional experiments, together with the evaluation of the network on the Test Set at both the Audio and Frame levels, are discussed in Section 4.5.1.

Table 4.7: Performance of the configurations obtained by varying the Parameters.

Pooling/Flatten Layer	Number of FC Layers	Number of Neurons per FC Layer	Learning Rate	Batch Size	Lowest FVS Loss	Best Epoch	FVS Accuracy	TTS Accuracy	TTS Loss
avg4_flatten	1	1024	1e-4	20	0,279867083	144	0,910655200	0,949834228	0,158503562
				60	0,271329820	317	0,915287912	0,959348440	0,127505466
				100	0,288176149	282	0,904037058	0,950266659	0,158361688
			1e-5	20	0,268660605	1663	0,922567844	0,968718469	0,110086784
				60	0,269418597	2692	0,919258773	0,966267824	0,114772394
				100	0,269779086	3163	0,922567844	0,964826286	0,119249158
		256	1e-4	20	0,279265702	328	0,917935133	0,961366594	0,131476372
				60	0,274091303	628	0,917935133	0,962808132	0,123999164
				100	0,293375343	1010	0,913964272	0,968430161	0,103010766
			1e-5	20	0,271797478	3303	0,921244204	0,967997670	0,111510925
				60	0,276702315	5350	0,919920564	0,966267824	0,115577891
				100	0,275079280	5991	0,921905994	0,962808132	0,125444233
		128	1e-4	20	0,271869212	524	0,918596983	0,963961363	0,120873004
				60	0,275764376	801	0,919258773	0,960357487	0,130715087
				100	0,288322300	876	0,910655200	0,956032872	0,145293161
			1e-5	20	0,275312990	3983	0,917273343	0,963817239	0,124627739
				60	0,277921289	7281	0,917935133	0,964393854	0,123896554
				100	0,280142844	8452	0,917935133	0,961510718	0,129528791
global_pooling	2	256, 128	1e-4	20	0,279277980	317	0,914626062	0,938734353	0,180421114
				60	0,297080070	451	0,900066197	0,929075956	0,201850489
				100	0,303139478	502	0,898742557	0,928211033	0,206832826
			1e-5	20	0,282508075	2360	0,912640631	0,940320015	0,175041765
				60	0,283704191	4300	0,909993410	0,940752506	0,177360266
				100	0,290422708	4692	0,906022489	0,937436938	0,185912594
		1024, 256	1e-4	20	0,277950883	266	0,917935133	0,941761553	0,165159345
				60	0,320999265	285	0,895433486	0,930085063	0,196818113
				100	0,295450836	429	0,910655200	0,943347275	0,164581433
			1e-5	20	0,278453618	1208	0,913302422	0,940896630	0,173380539
				60	0,272538871	2184	0,917273343	0,943923891	0,171140224
				100	0,278371185	2816	0,915949702	0,943635583	0,163533062
	1	1024	1e-4	20	0,276927322	640	0,915287912	0,942194045	0,171911165
				60	0,285098135	901	0,909993410	0,938301861	0,184194610
				100	0,305332392	1334	0,900727987	0,941761553	0,172729805
			1e-5	20	0,281987160	4798	0,909993410	0,941617429	0,174826384
				60	0,279797912	8241	0,911978841	0,941905737	0,175435111
				100	0,281194836	11337	0,911316991	0,943491399	0,170351982
		512	1e-4	20	0,278849244	833	0,914626062	0,940320015	0,176325157
				60	0,281500429	1208	0,913302422	0,936283708	0,188010082
				100	0,294665486	1711	0,906022489	0,939310968	0,177498668
			1e-5	20	0,281848341	7873	0,908007920	0,943347275	0,170167983
				60	0,284449369	10635	0,910655200	0,942194045	0,176432386
				100	0,282323211	14104	0,910655200	0,941905737	0,175223783
		256	1e-4	20	0,278084040	1241	0,915287912	0,942049861	0,169504464
				60	0,280244231	1625	0,913302422	0,938445985	0,186244294
				100	0,288093358	2689	0,908669770	0,944068015	0,168941036
			1e-5	20	0,285397410	8180	0,908007920	0,939887583	0,181509435
				60	0,283887297	17939	0,909993410	0,940464199	0,178032622
				100	0,287049115	15277	0,907346129	0,935851216	0,191615477

4.4.2. Barrier Analysis Module

As described in Section 4.2.3, this module aims at detecting the barrier within video frames to extract relevant features related to its movement; to that aim, we used a YOLOv5 whose PyTorch implementation is available on GitHub[25].

Selection of YOLOv5 model. YOLOv5 is proposed in five sizes namely nano (n), small (s), medium (m), large (l), and Xlarge (x); as the authors report, *"larger models like YOLOv5x* [...]

[25] https://github.com/ultralytics/yolov5

will produce better results in nearly all cases, but have more parameters, require more CUDA memory [i.e., more GPU RAM] *to train, and are slower to run. For mobile deployments we recommend YOLOv5s/m, for cloud deployments we recommend YOLOv5l/x"*[26]. It is important to underline that they tested the performance of these configuration (in terms of mean Average Precision (mAP) and computation time) on the COCO 2017[27] dataset, which contains data for multi object detection (about 80 different objects to be detected). Our task is way less complex from this perspective given two main reasons: i) our is a single-class detection task (i.e., we aim at detecting only one object - the barrier); and ii) with our pre-processing (see Section 4.3.2), we further simplified the detection task as we cropped out, from the original frame, the area within which the object to be detected will appear, hence, we would not have problems introduced by small-scale objects (typically the hardest to be detected) as the barrier occupy a significant portion of the patch we cropped out. Therefore, we opted for the YOLOv5s to understand whether a relatively small architecture could be effective for our task while preserving advantages in terms of computation times.

Training YOLOv5s. As the first test, we trained the YOLOv5s on our dataset by leveraging the parameters that the authors found to be the best in their tests on the COCO dataset. In the same way, we used the pre-trained weights that the authors obtained by training the YOLOv5s on the COCO dataset, i.e., we applied Transfer Learning. Additionally, we also adopted augmentation that the authors of YOLOv5 propose; basically, it is oriented at increasing the Training Set by performing some specific transformations (e.g., colourspace augmentations, mosaic augmentation)[28] that usually improve training performances according to the authors.

For the training, we set a maximum number of epochs equal to 300 and, to meet time and memory constraints, a batch size equal to 128 and an image size of 320 pixels. Actually, to make a first comparison, we trained the network twice: once with the Stochastic Gradient Descend (SGD) optimizer and a second time with the Adam optimizer. In addition, we adopted an Early Stopping criterion to stop the training if the performance of the network did not improve for a certain amount of consecutive epochs. The Early Stopping is applied in combination with a specific function, that YOLOv5's authors named *fitness* function, which is a linear combination of the two metrics mAP@.5 and mAP@.5:.95 weighted with 0.1 and 0.9 respectively. The objective of the training is to find the adequate set of weights that maximise this Fitness Function (FF), while the objective of the Early Stopping is to stop the training if the fitness does not improve for more than, in our case, 60 consecutive epochs. Fig 4.12 shows the comparison between the two training we performed in this phase; the performances have been evaluated on the Validation Set.

The results show that in the case of Adam Optimizer the performance of the network is characterised by heavy oscillations, at least for the first epochs; also, even though they stabilise during the last epochs, overall, they are slightly worse than that obtained by using the SGD. Hence, the analysis performed in the following has been made by taking into account the SGD Optimizer.

Nevertheless, before discussing other tests we performed, for the sake of knowledge, we think that is important to provide a theoretical overview of the FF, metrics, and the loss

[26]https://github.com/ultralytics/yolov5/wiki/Tips-for-Best-Training-Results
[27]https://cocodataset.org/#home
[28]https://github.com/ultralytics/yolov5/issues/1423

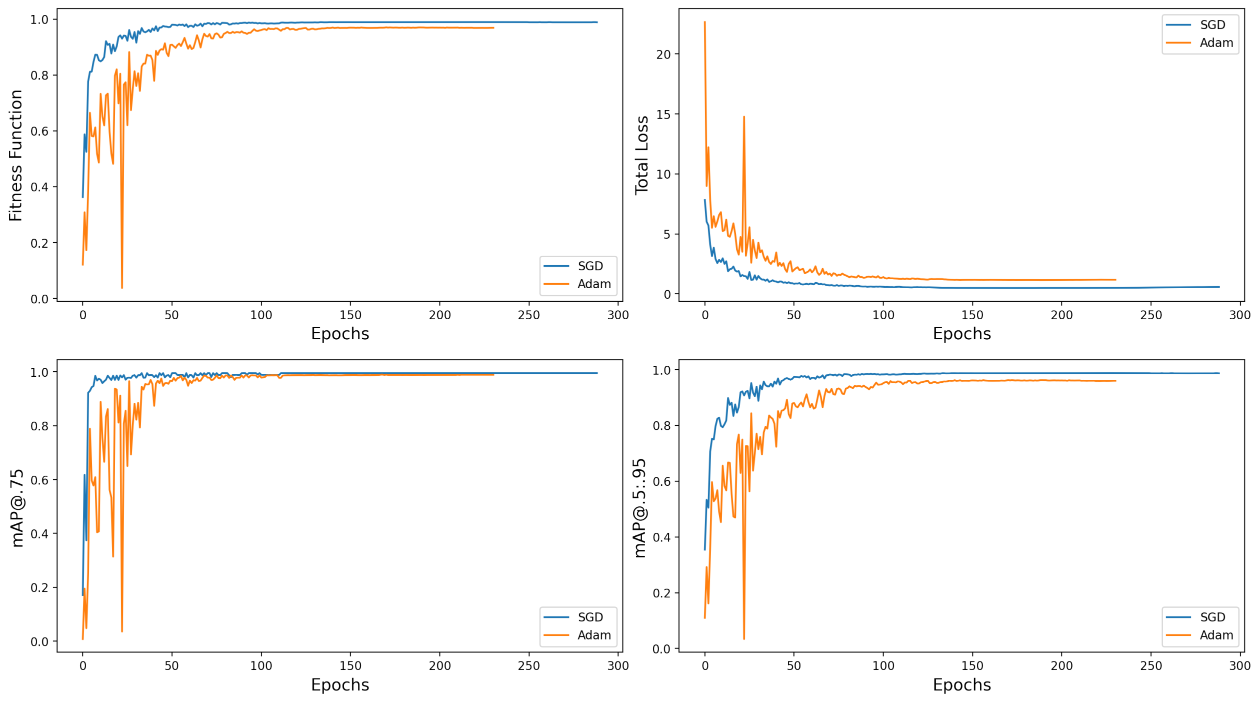


Fig. 4.12. YOLOv5s Trained with SGD Optimizer vs Trained with Adam Optimizer: Trends have been obtained by considering the Validation Set.

function that YOLOv5s involves.

Theoretical Overview on YOLOv5's Fitness Function, Metrics, and Loss Function. Typically, as we also did for the CNN discussed in Section 4.4.1, the best weights are selected by monitoring the loss function and selecting the configuration related to the lowest loss on the Validation Set. However, in this case, the best weights are selected by taking into account the highest value of the FF computed on the Validation Set. For the training discussed above, in the case of SGD Optimizer, the maximum fitness value on the Validation Set (0,989664) has been achieved at the 229th epoch; hence, the best weights that should be considered are those related to that epoch. Nevertheless, we tried to understand what would happen in case the FF had a different shape. In doing so, we also added the mAP@.75 to the FF; therefore, the FF can be seen as a function of three parameters (w1, w2, w3):

$$FF(w1, w2, w3) = w1 * mAP@.5 + w2 * mAP@.75 + w3 * mAP@.5 : .95$$

By varying the parameters we can obtain different fitness functions. We decided to add the mAP@.75 for the following reason. From a theoretical perspective, the mAP is nothing more than the mean of the different Average Precision (AP) computed for each class (i.e., each object that is intended to be detected). Important to mention, as we have only one class (i.e., the barrier), the mAP and AP are equivalent in our case: nevertheless, we will keep using the mAP notation for consistency. The AP is computed as the area under the Precision-Recall curve where the Precision (P) is computed as $\frac{TPs}{TPs+FPs}$ while the Recall (R) is equal to $\frac{TPs}{TPs+FNs}$. Here, TPs, FPs, and FNs indicate True Positives, False Positives, and False Negatives respectively. Hence, P indicates the percentage of predictions that are actually correct (i.e., in our case, how many of the detected bounding boxes really contain barriers), while R indicates how good is the model in detecting the object (i.e., how many barriers have been correctly detected among all the barriers contained in the dataset). When it comes to object detection, TPs, FPs, and FNs are computed by taking into account a given Intersection over Union (IoU) threshold as a reference. For example, if we consider an IoU threshold of 0.5 (IoU = 0.5), if the predicted bounding box overlaps with the true bounding box for at least 50%, the prediction is counted as TP; if it does not overlap at all, or overlaps for less then 50%, it is considered as a FP; lastly, if no bounding box is predicted but an actual bounding box exists, we have a FN. So, basically, mAP@.5 (which, in our case, is equal to the AP computed with IoU = 0.5) is the area under the P-R curve computed by considering an IoU = 0.5. Typically, the higher the IoU threshold, the lower the value of the relative metric. This is quite intuitive: if a predicted bounding box overlaps for 60% with the actual one, it will be considered as TP in the case of IoU = 0.5, while, in the case of IoU = 0.75, it will be considered as a FP. Hence, the mAP@.75 is more restrictive as, especially for single-object tasks, it is always lower (or at most equal) than the mAP@.5. Therefore, it may be more representative of the real performances since our aim is to adopt the object detector to obtain the barrier's bounding box from which to extract the height to understand the movement of the barrier over time; a high mAP@.5 may be representative of a good object detector, but its predictions may be not precise enough for our task. On the other hand, in order to avoid *overfitting*, we cannot rely on the mAP@.75 only to select the best weights. Indeed, conceptually speaking, if we select the best weights on the basis of the mAP@.75 only (or even mAP@.95 to find an extremely precise model, if possible), we could potentially obtain a model that is specified in finding the barrier within the frames composing the Validation Set that could not perform properly on the Test Set (or at run-time). Hence, to obtain a model that is capable of generalising (i.e., performing as properly as possible in different scenarios) while being sufficiently precise in predicting the bounding boxes, we decided to keep within the computation of the FF at least two between mAP@.5, mAP.75, and mAP@.5:.95; where the latter is the average of 10 APs computed with different IoU thresholds from 0.5 to 0.95 with a step of 0.05 (i.e., 0.5, 0.55, 0.6, ..., 0.75, ..., 0.95).

Selecting the Best Weights by Varying the Fitness Function. Based on this theoretical reasoning, we studied what would have changed in the weights selection process if we varied the FF. For the sake of comparison, we also considered a completely different FF that, instead of taking into account the metrics, considers the loss values. Basically, such a FF, named *Loss-based Fitness (LbF)*, allows for the selection of the weight related to the lowest total loss on the Validation Set. Extremely important to underline: *by varying the FF, according to our tests, we do not affect the training in any manner*. The tests we performed show that YOLOv5s is very stable, i.e., by repeating the training at different times without changing the parameters, hyperparameters, and the Training and the Validation Sets, we obtain always the same results (or results that are extremely close) at each epoch. Hence, *the modification of the FF, being a simple linear combination of metrics, only helps to select a specific configuration of weights instead of another; e.g., those related to the training epoch X instead of those related to epoch Z*. Table 4.8 shows the results of some of these tests by indicating, for each FF, which is the best-related epoch and the values of the metrics evaluated on the Validation Set. Obviously, the maximum value of the LbF is extremely different from the others because it is computed by taking into account the loss functions instead of the metrics.

Table 4.8: Performance of YOLOv5s on the Validation Set by varying the Fitness Function.

Fitness Function	Maximum Value	Best Epoch	mAP@.75	mAP@.5:.95	Total Loss
FF(0.1, 0.0, 0.9)	0.98814	229	0.99499	0.98738	0.50670
FF(0.1, 0.2, 0.7)	0.98966				
FF(0.1, 0.4, 0.5)	0.99118				
FF(0.0, 0.3, 0.7)	0.98966				
FF(0.0, 0.5, 0.5)	0.99118	223	0.99500	0.98737	0.50599
LbF	490.450	172	0.99497	0.98692	0.49185

All FFs reported in the first macro-row of Table 4.8 lead to the same results, this is because they “point” to the same set of weights (i.e., those related to the 229th epoch). In the other cases, we have slightly different results, anyhow, from a conceptual perspective, FF(0.0,0.5,0.5) does not contribute too much in terms of either metrics (the mAP@.75 practically equals that of the other FFs) or total loss. As for the other sets of weights, i.e., those obtained by considering the other FFs (BestMetricsWeights) and those obtained by taking into account the LbF (BestLossWeights), we performed another test by considering the Test Set to understand which configuration produces the best results in terms of generalisation. We noticed that also on the Test Set the two sets of weights produced about the same performances; both of them achieved a mAP@.75 of 0.99499, while, in terms of mAP@.5:.95, the BestMetricsWeights allowed the network to perform slightly (almost imperceptible) better than the BestLossWeights (0.992064 against 0.992029). Given these results, we can say that the BestMetricsWeights help the network to achieve slightly “better” generalisation performances, hence, we have considered these for the tests performed in Section 4.5.3.

4.5. Evaluation of Results

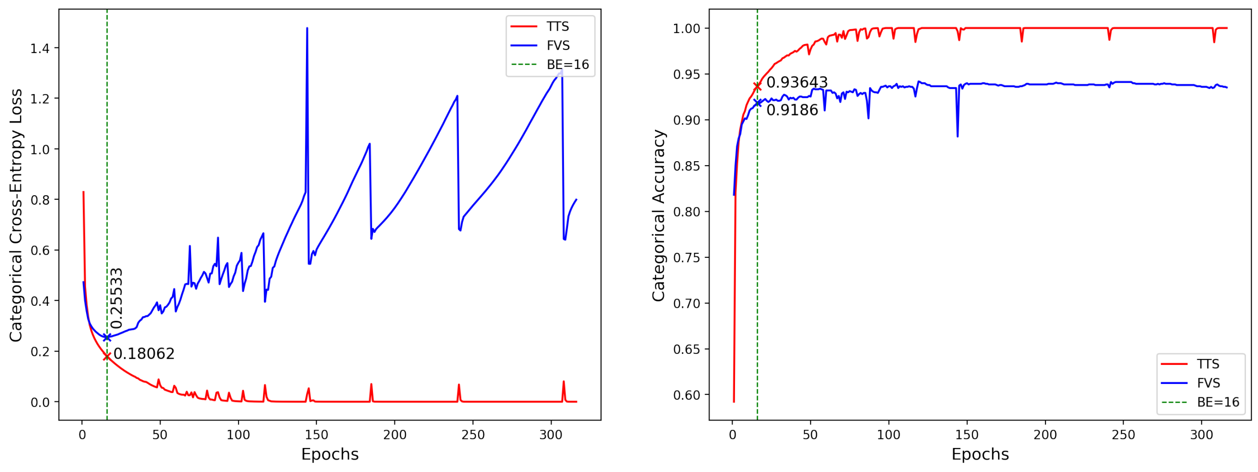
In this section, we perform some additional tests to further investigate the effectiveness of the DNNs trained as discussed above.

4.5.1. Warning Bell Detection Module

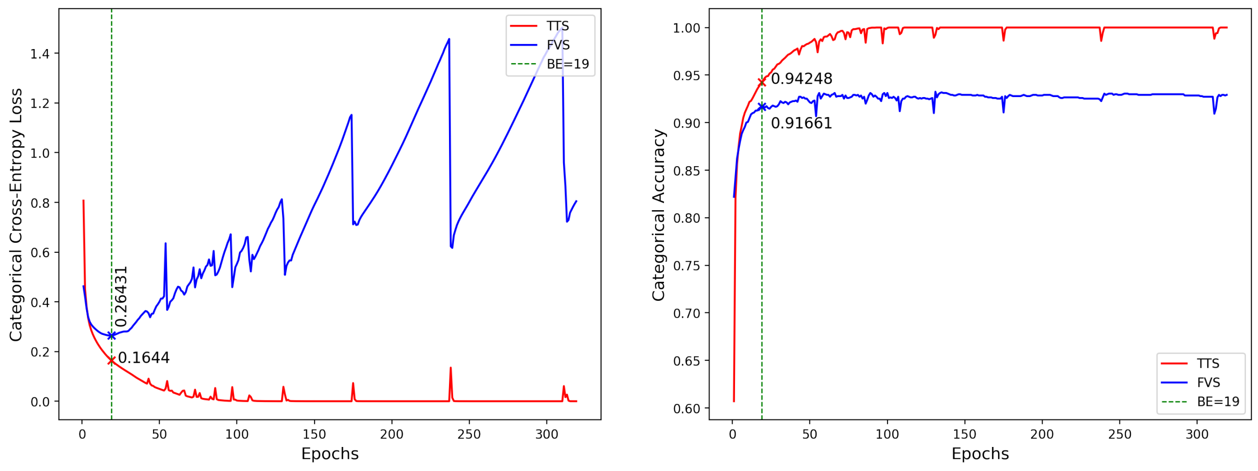
Starting from the best configuration identified in Section 4.4.1.2, we performed some additional experiments to: i) assess the advantages introduced by Transfer Learning when a limited amount of data is available; ii) study how the implemented network performs on the test set at both the Audio and the Frame levels; iii) evaluate the performance of the network by varying the dimension of the Frame-Level Trimmed Training Set (i.e., the TTS); iv) assess the cross-country generalisation performances of the implemented CNN (i.e., understanding how the network behaves on UK WB samples if trained with IT WB samples only, and vice-versa); and v) study the robustness of the network to the noise.

4.5.1.1. Efficiency of Transfer Learning

As discussed in Section 4.4.1.2, Fig. 4.11 shows the performance of the best configuration on the Frame-Level Training and Validation Sets when the pre-trained weights are loaded and frozen. To better understand the advantages of TL, we performed additional test experiments without loading and/or freezing the weights. Fig. 4.13(a) and Fig. 4.13(b) show the performance of the same network under these circumstances.



(a) Weights not loaded



(b) Weights loaded but not frozen

Fig. 4.13. Training performances of the best configuration with weights loaded but not frozen (b), or not loaded (a).

Table 4.9: Frame-Level metrics by varying the weight status.

Weights Status	Lowest FVS Loss	Best Epoch	FVS Accuracy	TTS Loss	TTS Accuracy
Loaded and frozen	0.2687	1663	0.9226	0.1101	0.9687
Loaded but not frozen	0.2643	19	0.9166	0.1644	0.9425
Not loaded	0.2553	16	0.9186	0.1806	0.9364

When training a DNN, it typically goes through an initial phase where its performances are characterised by some oscillations; then, they tend to be smoother with the advancement of the epochs. However, this happens only if the dataset used to train the DNN is sufficiently large and qualitatively optimal to properly characterise such a DNN. In our case, we collected a relatively "small" dataset, therefore, the oscillation phase continues over time until it diverges in case we do not load the weights (i.e., we perform the training from scratch - Fig. 4.13(a)), or we load but do not freeze them (i.e., we try to fine-tune the weights - Fig. 4.13(b)).

Table 4.9 reports the FVS and TTS loss and accuracy values the network achieves at the best epoch for each weight status (loaded and frozen, loaded but not frozen, not loaded). Despite the small gain in terms of FVS loss when not loading or not freezing the weights, if TL is not applied, the training curves are characterised by heavy oscillations (see Fig. 4.13). These results seem to validate the theoretical hypothesis that TL offers a suitable trade-off between good performance and network stability even if the dataset in not optimal or large enough to properly characterise the architecture.

4.5.1.2. Testing the CNN on Audio- and Frame-Level Test Sets

So far we have discussed tests considering only the TTS and the FVS. However, to understand whether the implemented CNN could be somehow effective, we have to test its accuracy on data it has not seen during the training phase, i.e., on the test sets.

First, we re-trained the best configuration for 1663 epochs on the TJS (last column in Table 4.4). The accuracy and the loss trends evaluated on the test sets are shown in Fig. 4.14, which reports the accuracy at both the Frame and Audio levels. In addition, Table 4.10 shows the confusion matrices at both the Audio and Frame levels, while Table 4.11 reports the precision, recall, and F1-score metrics evaluated on the test sets for each class at each level.

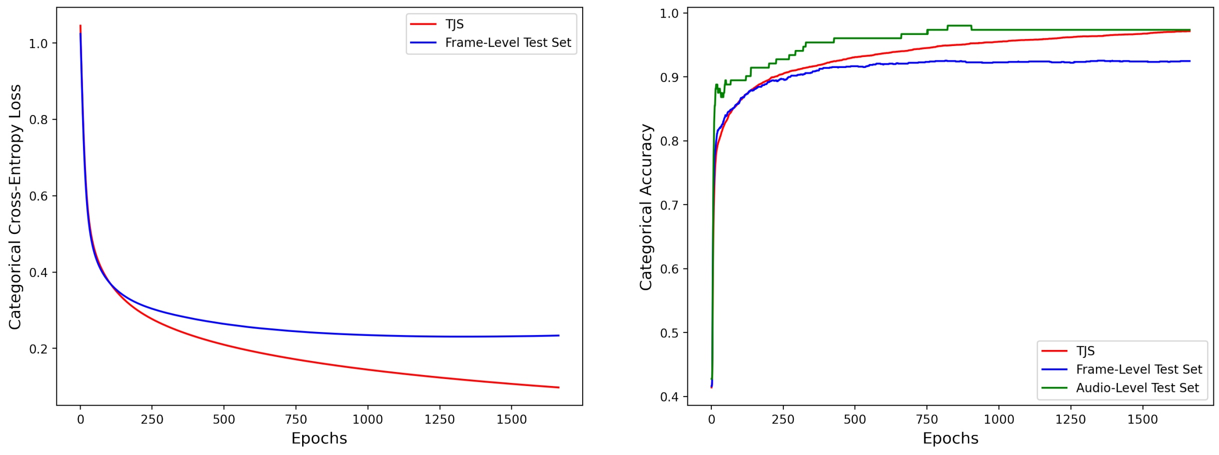


Fig. 4.14. Accuracy and loss performance of the best configuration on the Test Set(s).

Eventually, at the 1663rd epoch, we obtained a Frame-Level Accuracy of 92.48% and an Audio-Level Accuracy of 97.37%. Recalling what was introduced in Section 4.3.1, the Frame-Level Test Set has been extracted from the Audio-Level Test Set by framing its samples; hence, they practically contain about the same data. Taking that into account, the Accuracy at the Audio-Level is computed by averaging the K predictions at the Frame-Level (where K is the number of frames extracted from that audio). Hence, if an *audio* is composed of 10 *frames* and 7 of them are correctly classified, the Accuracy at the Frame-Level is 70%; however, by averaging them, we will have only a single prediction at the Audio-Level and, in

Table 4.10: Confusion matrices computed on Frame-Level and Audio-Level Test Sets.

Frame Level

Class \ Prediction	WB	NA	GE
WB	388	10	22
NA	11	417	22
GE	23	24	573

Audio Level

Class \ Prediction	WB	NA	GE
WB	42	0	0
NA	0	43	2
GE	0	2	63

Table 4.11: Precision, Recall, and F1-score per class.

Class	*Frame Level*			*Audio Level*		
	Precision	*Recall*	*F1-score*	*Precision*	*Recall*	*F1-score*
WB	0.9194	0.9238	0.9216	1	1	1
NA	0.9246	0.9267	0.9256	0.9556	0.9556	0.9556
GE	0.9287	0.9242	0.9264	0.9692	0.9692	0.9692

case this prediction is correct, we have an accuracy of 100% on that audio (and not 70%). Therefore, the Accuracy at the Audio-Level is higher because errors made by the CNN at the Frame-Level are somehow absorbed when computing the averaging the predictions.

As for computation time performances, the CNN takes, on average, $\sim$1.13 ms to process a single frame. To obtain the average processing time at the Audio-Level, we considered the weighted computation time as a few audio clips in the Audio-Level Test Set had a duration lower than 10 seconds. Hence, we multiplied the computation time required for each audio with the corresponding number of frames; then, we summed all the contributions and divided the sum by the total number of frames (i.e., the dimension of the Frame-Level Test Set). According to this calculus, the CNN takes, on average, $\sim$10.99 ms to process an audio clip.

4.5.1.3. Testing the CNN with Smaller Datasets

We studied the behaviour of the network by varying the dimensions of the TJS (Trimmed Joint Set) while always applying TL to understand the correlation holding between the dimension on the training set and the accuracy of the CNN. To that aim, we iteratively reduced the number of samples of the TJS (by 15%, 30%, 45%, and 60%); then, we retrained the network on each of these datasets and tested its performance on the Frame-Level Test Set, which remains unchanged to have a common reference to compare results.

The reduction of the TJS has been performed class-wise and subclass-wise to obtain balanced sets. For example, to build the "85% TJS", we separately decreased by 15% the number of IT WB frames, UK WB frames, NA frames, and GE frames. Table 4.12 summarises the structure of the configurations of the TJS and the evaluated accuracy of the network at the Frame and Audio levels. Notably, there is a correlation between the dimension of the TTS and the accuracy of the CNN; practically, the more the training data the higher the accuracy. This result indicates that there may be still room for improvement if additional data are collected.

Table 4.12: Reduced Datasets: Configurations and Accuracy.

Frames	Original TJS	85% TJS	70% TJS	55% TJS	40% TJS
IT WB	1152	979	806	633	460
UK WB	1152	979	806	633	460
NA	2440	2074	1707	1341	975
GE	3400	2890	2379	1869	1359
Total	8144	6922	5698	4476	3254
Frame Level Accuracy	92.48%	92.08%	91.61%	90.74%	90.07%
Audio Level Accuracy	97.37%	96.71%	96.05%	94.74%	93.42%

4.5.1.4. Testing Cross-Country Generalisation Performances

In this section, we discuss the experiments we performed to study the cross-country generalisation performance of the best configuration. The main concept is to train the network on samples related to a specific country (IT or UK) and then evaluate its performance on the test sets with a focus on the frames related to the other country. As before, the network was trained by relying on samples of the TJS and evaluated on the test sets.

The TJS includes 2304 WB frames, 1152 IT WB frames and 1152 UK WB frames (see the “Original TJS” column of Table 4.12). To generate dates as balanced as possible, we randomly selected 50% of NA samples and 50% of GE samples so that the data within “IT only TJS” and “UK only TJS” had the same distribution (in terms of percentages) of the Original TJS. Table 4.13 shows the structure of the two training sets we used for these analyses and the performance of the network at the Audio and Frame levels evaluated on the test sets.

Table 4.13: One-country TJS: Datasets Configurations and Accuracy on Test Sets.

Frames	IT only TJS	UK only TJS
IT WB	1152	0
UK WB	0	1152
NA	1220	1220
GE	1700	1700
Total	4072	4072
Frame Level Accuracy	82.42%	81.95%
Audio Level Accuracy	85.53%	86.18%

Besides the overall accuracy of the CNN (reported in Table 4.13), we are interested in understanding what happens when it comes to specifically detecting WBs:

- By training the CNN on the *Original TJS* (see tests in the previous sections), it was able to correctly classify 197 IT WB frames over 210 (0.9381 recall) and 191 UK WB frames over 220 (0.9095 recall).
- By training the CNN on the *IT only TJS*, it was able to correctly classify 193 IT WB frames over 210 (0.919 recall), but only 29 UK WB frames over 210.
- Lastly, by training the CNN on the *UK only TJS*, it was able to correctly classify 195 UK WBs over 210 (0.9285 recall), but only 26 IT WBs over 210.

As expected, the network has very low generalisation performances which can be due to the fact that the two kinds of WBs (IT and UK) are quite dissimilar; therefore, the feature learnt by the network when trained on IT frames are not adequate to correctly detect UK WBs (and vice versa). When trained on the original TJS, the CNN is capable of properly detecting both kinds of WBs, achieving a reasonably high recall. On the other hand, in the other two cases, although the amount of training data (4072 frames) was half the amount of data in the Original TJS (8144 frames), the CNN achieved comparable (or even better) performances in terms of recall; obviously, it had worse performance in relation to the WB sub-class that was not included in training data. Therefore, given also the results we obtained in Section 4.5.1.3, it would not be so visionary to think that if the IT only and UK only datasets will be increased, the classification accuracies of the "specialised" CNNs will consequently increase.

4.5.2. Tenting Robustness to Noise

The last experiment was oriented at evaluating the robustness of the CNN to different types of noises at different intensities. These experiments were performed by taking into account the best configuration and the weights learned during the re-training on the TJS. Then, performances were evaluated on test sets once the different noises were introduced.

We considered three different noises: a Gaussian Noise, that was synthetically generated, and two "real" noises which were representative of the NA class (Motorcycle Noise) and the GE class (Ambulance Noise); important to mention, the real audios selected as noises *were not included* in the training, validation, or test sets. Noises were taken into account one by one and added to each audio in the test set; so, basically, we obtained Gaussian-augmented test sets by adding the Gaussian noise to each audio in the Audio-Level Test Set, which clearly led to noisy frames in the Frame-Level Test Set. We did the same with Motorcycle and Ambulance Noises. Then, to study the performance of the network at different noise levels, we considered the Signal to Noise Ratio (SNR) and established seven SNR values representing seven noise intensities. The SNR can be computed as:

$$SNR = \frac{RMS^2_{Audio}}{RMS^2_{Noise}}$$

where RMS stands for Root Mean Squared value. By setting the SNR and knowing the RMS of each audio in the test set, we could retrieve the required RMS_{Noise} and generate a Gaussian Noise or model the Ambulance and Motorcycle Noises accordingly.

We performed a total of 21 tests by combining noise type and noise intensity level. Each test was carried out by considering the following steps:

- We fixed the type of noise: Gaussian, Ambulance, or Motorcycle.
- We set the required SNR: 0.1, 1, 2.5, 5, 10, 25, or 50.

  Important to mention, the lower the SNR the higher the contribution of the noise.

- For each audio in the test set, we added the noise so that the SNR of each Noisy Audio (i.e., the audio obtained by summing the original audio with the noise) matched the required SNR.

Table 4.14 reports the performance of the network for each test in terms of accuracy, and precision and recall of the WB class at both the frame and the audio levels.

Table 4.14: Network classification performances at different noise levels and types.

		SNR Level →	*50*	*25*	*10*	*5*	*2.5*	*1*	*0.1*
Gaussian Noise	***Frame Level***	*Accuracy*	88.39%	85.97%	80.20%	73.36%	66.17%	57.58%	43.56%
		WB Precision	0.9735	0.9870	0.9915	1.0000	1.0000	1.0000	0.0000
		WB Recall	0.7881	0.7238	0.5548	0.3738	0.1667	0.0286	0.0000
	Audio Level	*Accuracy*	95.39%	93.42%	84.87%	78.29%	69.08%	60.53%	40.79%
		WB Precision	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.0000
		WB Recall	0.8810	0.8333	0.6190	0.4286	0.1429	0.0238	0.0000
Motorcycle Noise	***Frame Level***	*Accuracy*	87.18%	83.96%	77.85%	71.21%	65.17%	57.05%	43.22%
		WB Precision	0.8606	0.8340	0.8057	0.7912	0.7700	0.7602	0.7750
		WB Recall	0.8500	0.8238	0.7452	0.6405	0.5310	0.3810	0.0548
	Audio Level	*Accuracy*	94.08%	90.13%	80.26%	73.68%	69.08%	56.58%	42.76%
		WB Precision	0.9767	0.8936	0.8571	0.8077	0.7778	0.8000	0.8621
		WB Recall	0.9524	0.9048	0.8095	0.6429	0.5476	0.3571	0.0000
Ambulance Noise	***Frame Level***	*Accuracy*	89.87%	88.46%	86.64%	85.57%	84.03%	83.29%	68.86%
		WB Precision	0.8541	0.8219	0.7964	0.7642	0.7611	0.7477	0.5111
		WB Recall	0.9405	0.9452	0.9476	0.9381	0.9167	0.8833	0.5905
	Audio Level	*Accuracy*	96.71%	94.74%	92.76%	90.13%	88.16%	87.50%	71.05%
		WB Precision	1.0000	0.9744	0.9189	0.9000	0.9583	0.9375	0.0000
		WB Recall	1.0000	1.0000	1.0000	1.0000	1.0000	0.9524	0.5952

As a general role, the performances of the network decrease as long as the contribution of the noise increases. However, in the case of Ambulance Noise, the WB Recall remains higher than 0.9 at the Frame-Level and equal to 1 at the Audio-Level up to an SNR value of 2.5, indicating that, despite the noise, most of (or all) the WB samples have been correctly detected. Similarly, the WB Precision remains around 0.8 at the Frame-Level and greater or equal to 0.9 at the Audio-Level indicating a low misclassification rate from the WB class perspective (i.e., among the samples classified as WB only a few of them belongs to the NA or GE classes). Given these results, we can assume that the network is quite robust to the Ambulance Noise we considered in our tests. Conversely, in the case of Gaussian on Motorcycle Noise, despite the WB Precision being quite high up to a SNR of 1, the performances in terms of accuracy and WB Recall degrades quickly. This basically indicates that the higher the noise the more the network struggles to correctly detect WB samples.

4.5.3. Barrier Analysis Module

As we did for the WBDM, we performed some additional tests to further investigate some aspects related to YOLOv5s and its effectiveness in relation to our goal. First, we tested the contribution that pre-trained weights and the augmentation methods proposed by YOLOv5's authors bring in terms of detection performances. Then, we applied the YOLOv5s trained on our dataset on some video clips showing the barrier moving over time to study the effectiveness of the approach to extract the motion path the barrier traces.

4.5.3.1. Efficiency of Transfer Learning and Augmentations.

We performed two experiments to test the effectiveness of Transfer Learning and the augmentation methods proposed by YOLOv5's authors. First, we kept the augmentation method but performed training from scratch, i.e., we did not consider the pre-trained weights; second, we adopted Transfer Learning but not the augmentation method. Fig. 4.15 reports the performances evaluated on the Validation Set. These tests have been done by using the SGD Optimizer, so, basically, the trends labelled as SGD in these charts are the same as those shown in Fig. 4.12.

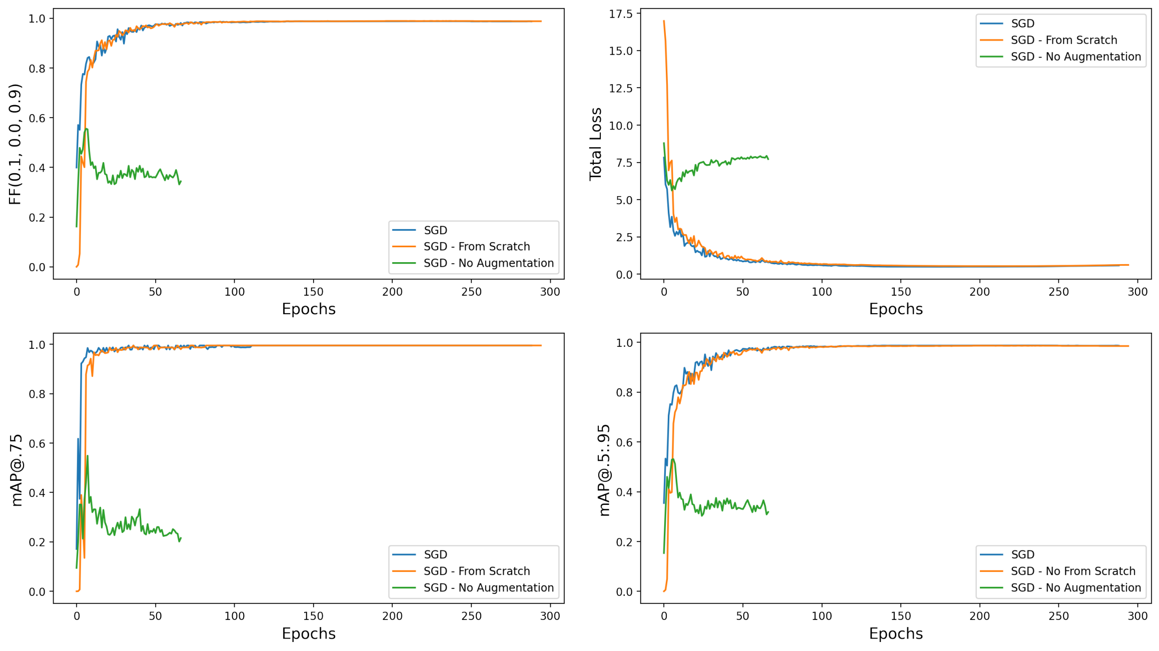


Fig. 4.15. Efficiency of Transfer Learning and Augmentations in YOLOv5s

It is quite evident that the augmentations proposed by YOLOv5's authors bring enormous benefits also in our case; on the other hand, pre-trained weights do not contribute in an evident way (differently to what happened for the WBDM). Perhaps, this is due to the fact that our task is quite simple compared to those the YOLOv5 has been created for (e.g., multi-object detection) or to the fact that the pre-processing we performed on our data (to "theoretically" help the network to perform properly) actually bring benefits in the training phase. Anyhow, whether we use or not Transfer Learning, the performances of YOLOv5s do not vary significantly. To understand which is the best way to train YOLOv5s, Table 4.15 reports the performances computed at the best epochs on the Validation Set; the best epochs for each test have been selected by considering the original fitness function as, also in case of training from scratch, our experiments showed that all the FFs (see Section 4.4.2 and Table 4.8 for additional insights) "pointed" to the same set of weights.

Based on the results obtained so far, we can affirm that YOLOv5s, trained with basic parameters and suggestions provided by the authors (i.e., pre-trained weights and augmentations), is extremely effective for our task. As already shown at the bottom of Section 4.4.2, the mAP@.75 and the mAP@.5:.95 evaluated on our test set are equal to 0.99499 and 0.992064 respectively, showing very high accuracy in detecting (and thus locating) the barrier.

Table 4.15: Performance on the Validation Set with and without Pre-Trained Weights.

Pre-Trained Weights	Best Epoch	mAP@.75	mAP@.5:.95	Total Loss
Yes	229	0.99499	0.98738	0.50670
No	235	0.99496	0.98671	0.53968

4.5.3.2. Studying YOLOv5s on Test Videos

To evaluate the extraction of the movement of the barrier, which is the final goal of the BAM, we applied YOLOv5s on four video clips extracted from the original Day Clear (DC) and Night Clear (NC) videos introduced in Section 4.3.2. First, we extracted a video clip from the DC video (CDC) and another video clip from the NC video (CNC); these have a duration of about 28 seconds (at 30 FPS) and are respectively composed of 842 and 858 frames. Since both CDC and CNC i) do not contain any noise in terms of weather conditions or augmentations and ii) contain some frames (specifically 240 frames) that have been already used in the training, validation, and test phases, these two videos have been considered only to have a reference of how the network would perform in "ideal" condition during daylight (i.e., CDC) and nighttime (i.e., CNC). Then, we applied to each CDC and CNC frame a random transformation chosen among those introduced in Table 4.2. Given the random selection and the fact that also the augmentations themselves are based on parameters selected randomly from some specific intervals, we can assume that it is extremely unlikely that we obtained frames perfectly identical to those we had in the Training, Validation, and Test Sets. In this way, we obtained two other videos named Transformed DC (TDC) and Transformed NC (TNC); they have the same number of frames as their non-transformed counterpart.

Detection Pipeline and Computation Times. To analyse the computation times, we referred to the whole detection pipeline of the BAM shown in Fig. 4.16.

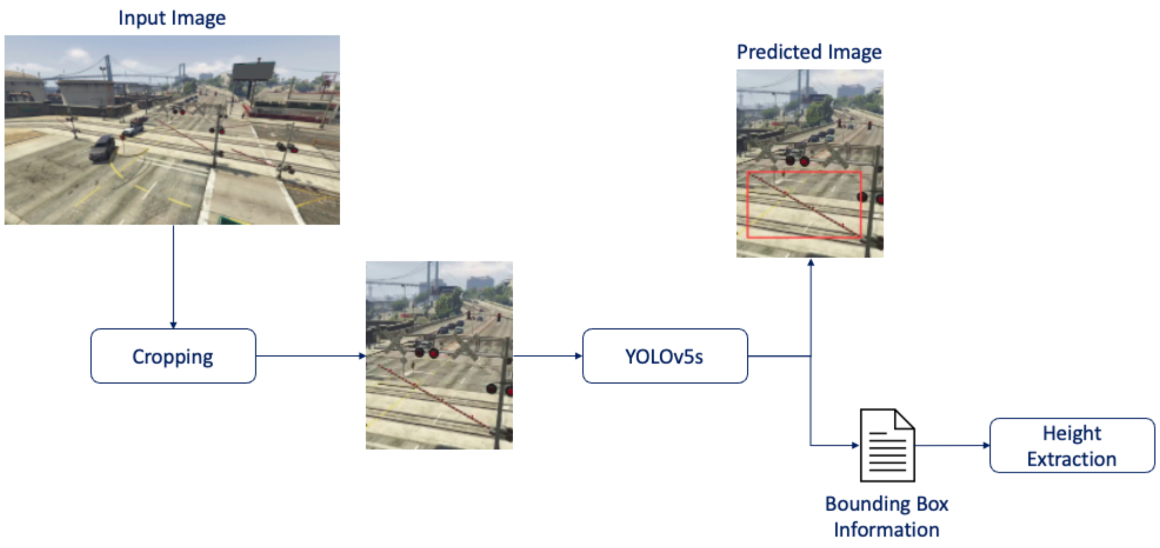


Fig. 4.16. Detection Pipeline: Extracting the Barrier's Height through YOLOv5s.

By taking into account a hypothetical real-time execution, for each frame, the BAM:

1. Extracts the region of interest, i.e., the area that contains the barrier, from the frame. The "Cropping" algorithm does not introduce any AI and is based on some pre-specified coordinates that do not vary as the videos were collected with a simulated camera that have always the same field of view.
2. Predicts the bounding box containing the barrier by executing YOLOv5s.
3. Extracts the height of the bounding box.

Important to mention, YOLOv5 already implements an internal function to directly extracts the heights and the widths of the predicted bounding boxes, however, we designed a "Height Extraction" algorithm from scratch to also implement a corrective behaviour to cope with False Negatives, i.e., barriers that are not detected, and False Positives, i.e., bounding boxes that not contain the barrier; further details are given below. In terms of computation times, Table 4.16 reports the times, on average, required by the detection pipeline to process the frames of each video we considered. Given the low computation times, we can assume that this approach is suitable for real-time applications (at least for this detection phase).

Table 4.16: Pipeline's Computation Times in milliseconds (ms)

	Avg. Time per Frame (in ms)			
	Cropping	*YOLOv5s*	*Height Extraction*	***Tot.***
CDC	3,4493	6,9784	0,0018	10,4295
TDC	3,4493*	6,8656	0,0021	10,3170
CNC	3,3114	7,0391	0,0020	10,3525
TNC	3,3114*	7,0193	0,0021	10,3328
Avg.	3,3804	6,9756	0,0020	10,3580

* frames were first cropped and then augmented, so cropping times are the same.
(TDC = CDC and TNC = CNC)

Motion Detection Performances. As mentioned above, we tested YOLOv5s on CDC, TDC, CNC, and TNC. Figured 4.17 and 4.18 report the movement of the barrier obtained by processing the videos through the pipeline presented above; the former compares the results obtained on CDC and TDC, while the latter compares the results obtained on CNC and TNC. Notably, besides some spikes, the same motion trend is identified for both the original and transformed videos. The two horizontal lines, *Top Threshold* and *Bottom Threshold*, can be used as references to identify the status of the barrier; e.g., if the height is above the *Top Threshold*, the barrier is open; if it is in between *Top Threshold* and *Bottom Threshold*, the barrier is moving (closing/opening); while if it is lower than *Bottom Threshold*, the barrier is closed.

Important to mention, there are a few cases in which the barrier is not detected by YOLOv5s (False Negatives). For CDC and TDC, the barrier is not detected in frames 34, 35, 36, 37, and 819 (5 miss-detection over 842 frames - 0.6%). On the other hand, for CNC, YOLOv5s does not produce any prediction for frames 35, 36, 37, 38, 39, and 820 (6 miss-detection over 858 frames - 0.7%); almost the same goes for TNC but, in this case, the barrier in frame 39 is detected correctly. Since predictions are not produced for both the original and augmented videos for (almost) exactly the same frames, we can assume that the augmentation (i.e., the noise) does not affect the detection performance, instead, the miss-detections could be

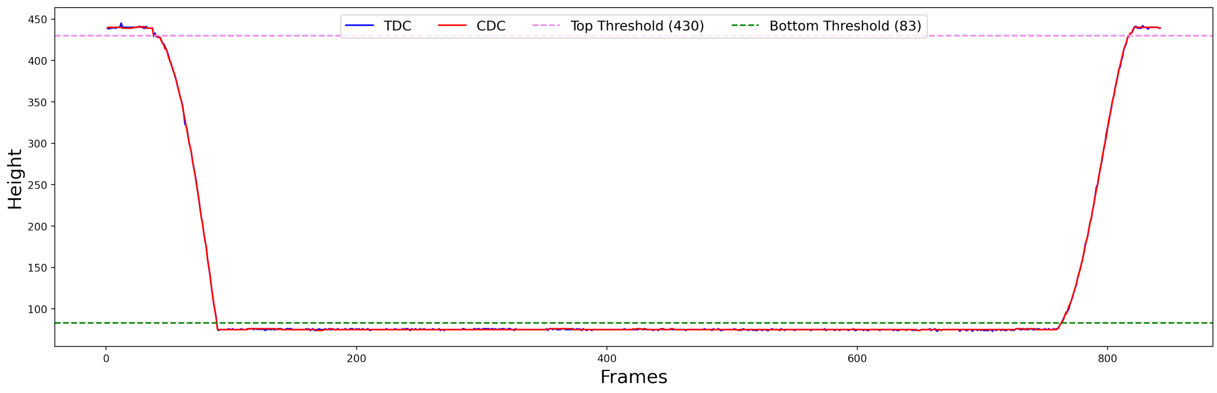


Fig. 4.17. Barrier Movement (Heights) Extracted from CDC and TDC.

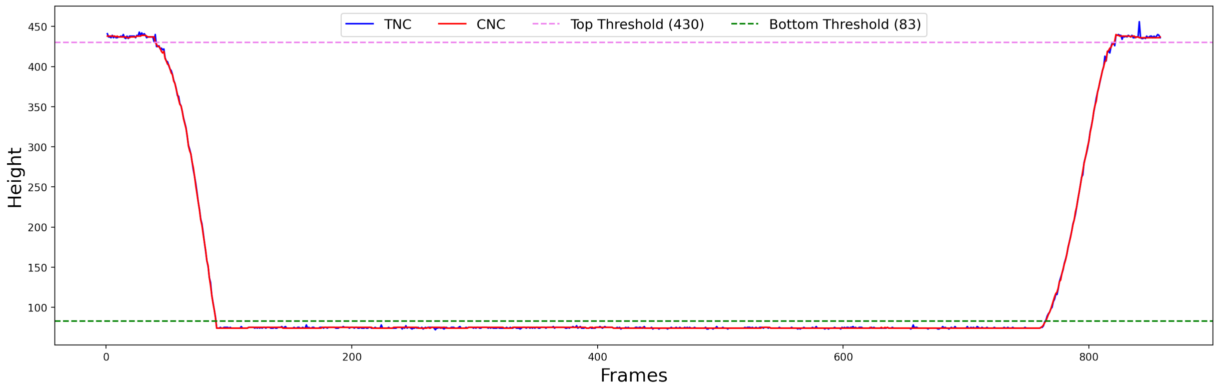


Fig. 4.18. Barrier Movement (Heights) Extracted from CNC and TNC.

mainly due to the position of the barrier in those frames. Indeed, these frames (shown in Figures 4.19 and 4.20) share the characteristic that the barrier is almost exactly vertical.

Besides the fact that it could be possible to improve the performances by involving larger datasets with more samples of any possible position the barrier can assume, this issue can also be mitigated by implementing a sort of corrective behaviour in the “Height Extraction” algorithm. Assuming that YOLOv5s correctly detects the barrier in the i-th frame and its height is h_i, we can have three possible cases for the (i+1)-th frame:

1. YOLOv5s correctly detects a single bounding box; hence, no corrective actions are needed.
2. YOLOv5s detects more than one bounding box; in this case, only one is the True Positive (we only have one barrier for each frame), the others are False Positives. To mitigate this issue, we considered as the correct prediction the bounding box whose height was closer to h_i.
3. YOLOv5s does not detect any bounding box; in this case, we have a False Negative. To mitigate this issue, we simply assigned to $h_{(i+1)}$ the same value of h_i. Considering the low miss-detection rates we got (0.6 and 0.7%) and the fact that we only got at most 5 consecutive miss-detections, we think that this alternative is viable; however, in case we had too many consecutive miss-detections, then, a re-training with a more comprehensive dataset would have been required.

Besides the height extraction, another important aspect to consider is the confidence with

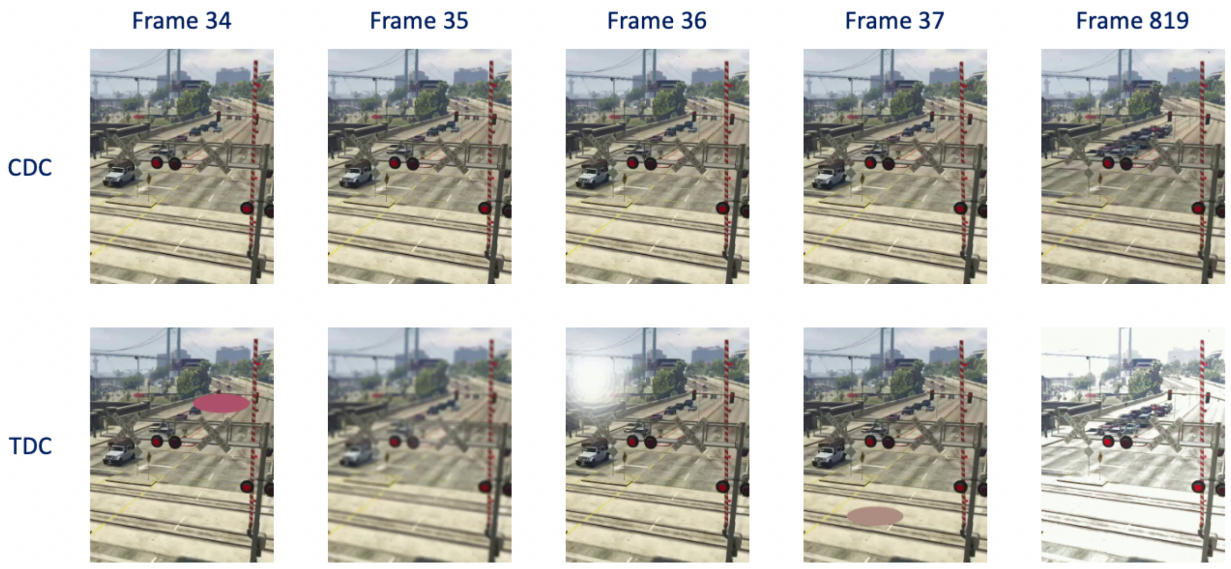


Fig. 4.19. CDC and TDC Frames with undetected barriers.

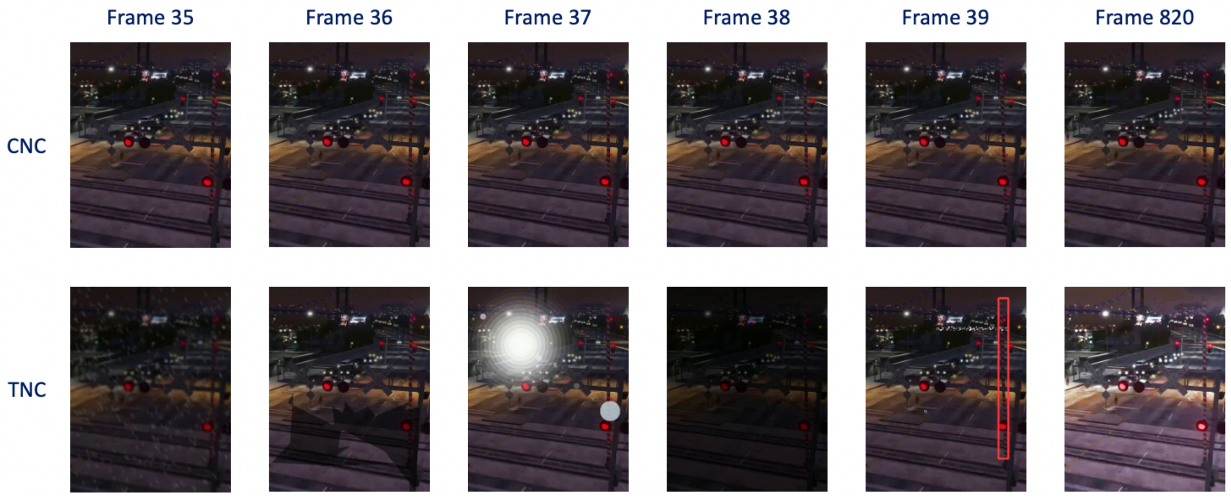


Fig. 4.20. CNC and TNC Frames with undetected barriers.

which YOLOv5 produces a specific output. Figures 4.21 and 4.22 report the confidence that the network associate with each prediction; the former compares the results obtained on CDC and TDC, while the latter compares the results obtained on CNC and TNC. Differently from what we did with the heights, in this case, we did not associate any confidence value to the barriers that were not detected.

In about all cases, confidence values follow similar trends. In the case of daylight (CDC and TDC[29]), YOLOv5s has about the same performances, whether the frames are augmented or not. On the other hand, during nighttime (CNC and TNC), the noise introduced by the transformations visibly affects the confidence the network has when predicting the bounding boxes; in four cases (red lower peaks in Fig. 4.22), the confidence is significantly lower for TNC frames than for their CNC counterparts. These four cases are shown in Fig. 4.23. Even though the sun flares cover the barrier in TNC Frames 1 and 841, YOLOv5s is still able

[29] Important to mention, the transformations include “brightness reduction”, hence, some TDC frames may be more related to night light than daylight.

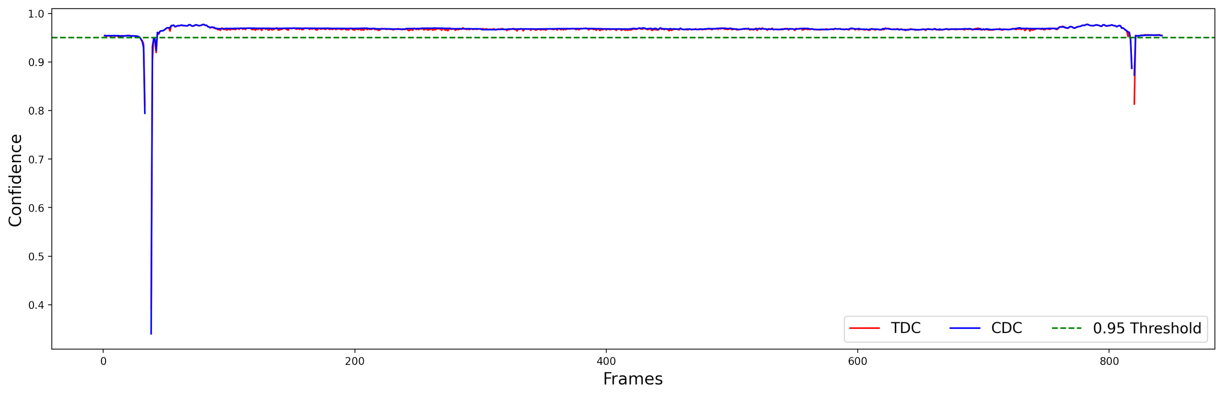


Fig. 4.21. Prediction Confidences for CDC and TDC Frames.

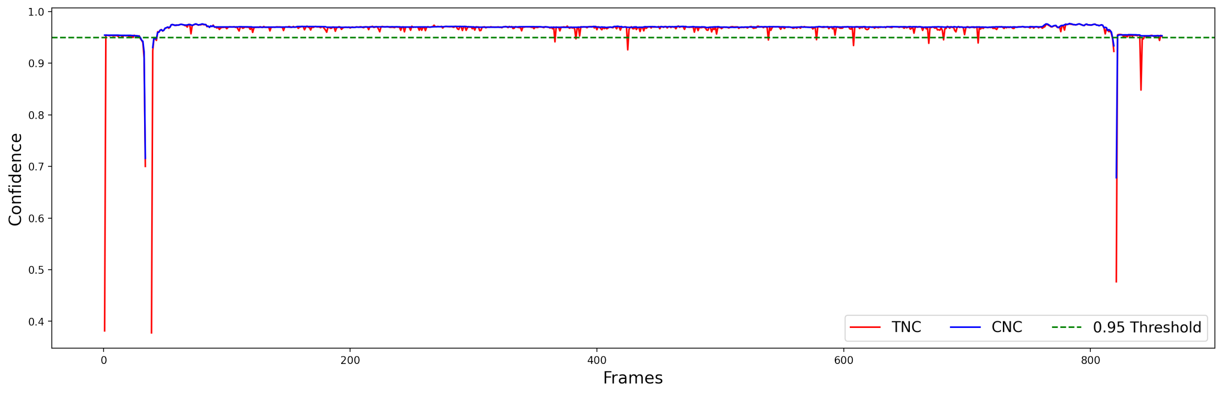
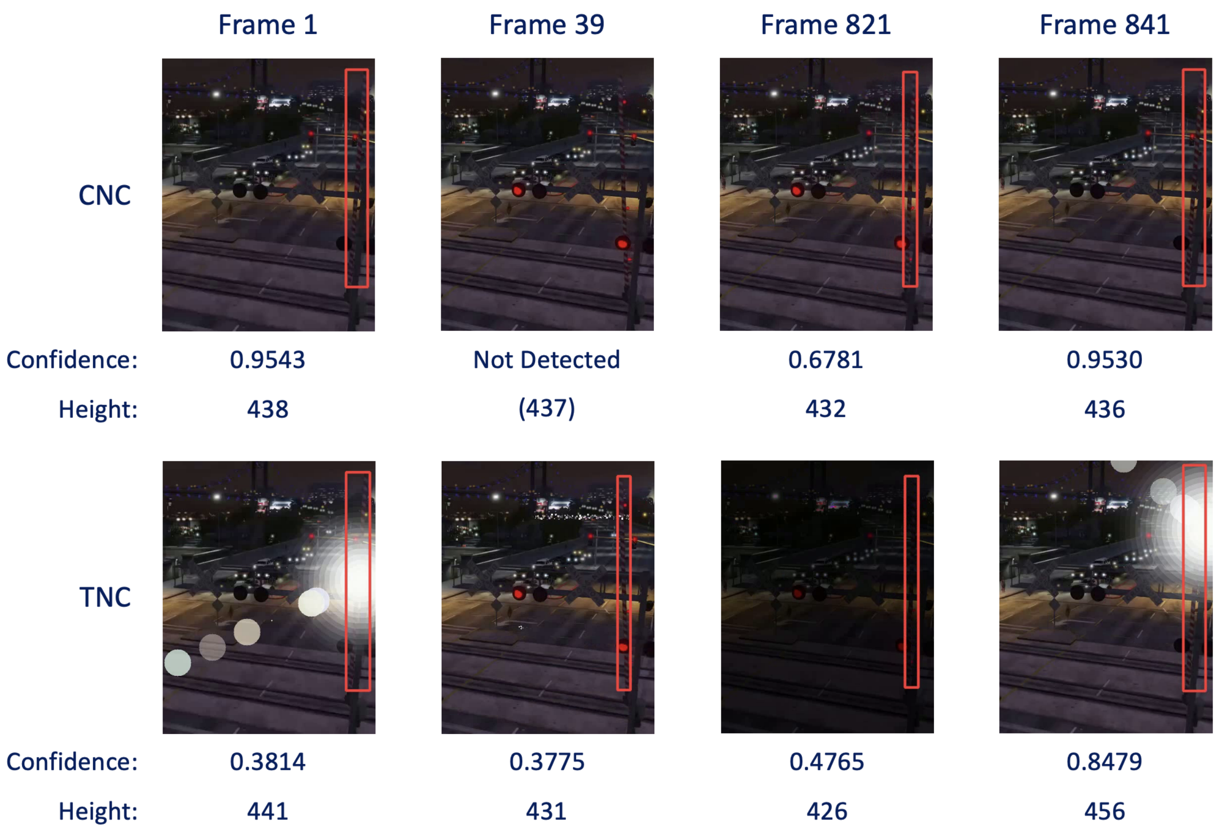


Fig. 4.22. Prediction Confidences for CNC and TNC Frames.

to detect it but with lower confidence; interestingly, for TNC Frames 821, the height of the produced bounding box is 456 pixels which is 20 pixels greater than the actual height (436 pixels, if we consider the detection for CNC frames as “ground truths” when possible). This phenomenon can also be seen in Fig. 4.18 where we have a blue peak towards the end of the line. Lastly, the barrier in CNC Frame 39 is not detected and its height has been set to that of the latest detected bounding box (CNC Frame 34).

Overall, we can notice that when the barrier is vertical, YOLOv5s either detect it with lower confidence compared to the other positions (i.e., oblique and horizontal) or do not detect it at all. This is quite evident from Figures 4.21 and 4.22, and also from Table 4.17 that reports the average confidence values for each video and for each position of the barrier (open, moving, and closed). To identify the positions, we took into account the Top and Bottom Thresholds as shown in Figures 4.17 and 4.18. Important to mention, frames intervals for CNC and TNC should have been the same (as it happened for CDC and TDC) as the position of the barrier is exactly the same in these videos (the position in TNC Frame ‘i’ is exactly the same as that in CNC Frame ‘i‘); however, it seems that the transformations disturb the detection a bit leading YOLOv5s to detect bounding box with slightly different heights, hence, the turning points (from open to moving, and from moving to closed) are shifted a bit. Nevertheless, these shifts are not that significant; we are talking about a couple of frames.

Downstream all these analyses, we can conclude that YOLOv5s succeed, with some adjustments a-posteriori, in detecting the barrier in almost all the considered conditions, then,



Frame 1
Frame 39
Frame 821
Frame 841
CNC
Confidence: 0.9543 Not Detected 0.6781 0.9530
Height: 438 (437) 432 436
TNC
Confidence: 0.3814 0.3775 0.4765 0.8479
Height: 441 431 426 456

Fig. 4.23. TNC Frames with Low Confidence Values.

Table 4.17: Mean Confidence Values per Barrier Position.

	Open		**Moving**		**Closed**		***Overall***
	Frames	*Avg. Conf.*	*Frames*	*Avg. Conf.*	*Frames*	*Avg. Conf.*	***Avg. Conf.***
CDC	1:39 — 818:842	0.9373	40:88 — 764:817	0.9714	89:763	0.9681	0.9606
TDC	// — //	0.9370	// — //	0.9711	// — //	0.9673	0.9599
CNC	1:40 — 821:858	0.9458	41:89 — 766:820	0.9711	90:765	0.9702	0.9614
TNC	// — 819:858	0.9247	// — 765:818	0.9707	90:764	0.9687	0.9594

The annotation “X:Y” means “from frame X to frame Y”; “–” stands for “and”; while, “//” means that the values are equal to those indicated in the line above.

the proposed approach can be considered viable to extract the movement of the barrier over time. Perhaps, as a possible improvement, it would be possible to enlarge the dataset to include more samples of “vertical” barriers and possibly overcome the problems they currently introduce.

4.6. Discussion of Results

In this section, we provide a discussion of all the results obtained so far while implementing two of the four modules composing the LC Intelligent Monitoring System we investigated (Fig. 4.1). Lastly, we tried to formally answer to the research questions reported in Section 4.1 and evaluate the analysed system from the perspective of the KPIs in Table 4.1.

4.6.1. Discussing Practical Results and Future Opportunities.

Warning Bell Detection Module. Concerning the WBDM, we build a three-class dataset from scratch (by leveraging the AudioSet dataset [7] and the YouTube platform) to train a VGGish-inspired [7] CNN to recognise WBs while being robust to sound that may be easily confused with WBs. The CNN was trained on 1-second frames achieving an accuracy of 92.26% at the Frame-Level, while the accuracy at the Audio-Level was 96.87%. The errors made by the network when classifying sounds may have different effects: first, False Negatives (FNs), i.e., not-detected WBs, may lead to unnecessary maintenance intervention; second, False Positives (FPs), i.e., GE or NA sounds detected as WBs, may lead to missed maintenance activities when they are actually required. Nevertheless, if we consider the WBDM as a system oriented to support human operators (a scenario that is closer to reality), we can actually think that repair intervention would be activated only after that a human operator has double-checked the outcome of the WBDM by, for example, manually re-analysing the recorded audio in case the WBDM classify it as defective; hence, the problems introduced by the FNs would be mitigated. Similarly, FPs do not represent real criticisms. Indeed:

- If a GE/NA sound is classified as a WB, but neither the barrier nor the WLs are working, it could actually mean two things: i) two modules have failed and only the WBs are working properly (an extremely rare event); and ii) the WDBM has produced a FP. Hence, provided that a network manager could actually know where a train is located on the rail network and, hence, it could potentially also know, by means of other monitoring systems, if the LC should activate or not (basing on the distance between the train and the LC), then, the second alternative is the only plausible one, therefore, FPs could be simply ignored.
- Differently, if the LC is active (because the train is approaching) and the WB does not work but is detected by the WBDM as active (FP), then, we could have an issue but, also in this case, it could be mitigated by the aleatory of environment itself. It is unlikely that, each time the LC is active, there would be a sound next to it that could mislead the classification performed by the WBDM. Hence, it is not so visionary to think that, if at the i-th activation of the LC we have a FP, we will not have a FP at the (i+1)-th or (i+2)-th activation and, usually, we have multiple activations within the same day, therefore, it is quite unlikely that the malfunction remains undetected for a long time.

So, although the level of safety decreases if warning signals do not operate correctly, compared with the current situation where those malfunctions can remain undetected for a long time, possibly until the next planned maintenance inspection, the intelligent detection system generally represents an improvement in terms of safety, also considering a certain – reasonably limited – amount of FNs and FPs. Actually, if we consider WB as the positive class (while NA and GE as the “joint” negative class), our model did not produce any FP or FN at the Audio-Level; misclassifications only happened between NA and GE classes (i.e., two NA audios were classified as GE, and two GE audios were classified as NA). This means that, from a conceptual perspective, our network is capable of correctly identifying WB audios and does not classify as WB any non-WB audio.

We also performed some additional experiments to test the behaviour and the performance of the CNN:

- First, we analysed the advantages introduced by Transfer Learning (Section 4.5.1.1). The results showed that TL offers a trade-off between performance and network stability, and hence suggests the viability of the approach, subject to optimisation for specific real-world LC diagnostic applications.

- Second, we evaluated the classification accuracy by varying the dimension of the TJS (Section 4.5.1.3). The results show a correlation between the number of samples in the training set and the model's accuracy; in particular, the higher the number of samples, the better the results. Notably, there may be room for further improvement if additional samples are collected.
- Third, we evaluated the cross-country generalisation performance (Section 4.5.1.4). As expected, the network is not capable of correctly classifying UK WB samples if trained on IT WB samples only (and vice-versa). Also, according to our tests, "specialised" CNNs (i.e., trained with WB samples coming from a specific country) seem to be more promising than the "general-purpose" CNN (i.e., trained with all the WB samples regardless of their origin) when evaluated on the class subject of the specialisation. To further investigate this aspect, we collected 60 additional audios: 30 from French (FR) and 30 from German (DE) LCs. Then, we trained the CNN on the entire Frame-Level WBD (see Sections 4.3.1 and 4.4.1.1) and tried to predict the class for FR and DE WB frames. As expected, the network has very poor classification performances: as for DE LCs, the network achieved 6.33% accuracy at the Frame-Level and was able to correctly classify only a single audio (3.33% accuracy at the Audio-Level); similarly, for FR LCs, even though it performed a little bit better, the network achieved only 33% accuracy at the Frame-Level and was able to correctly classify only 9 audios over 30 (30% accuracy at the Audio-Level). These results seem to validate the hypothesis that, most likely, it would be better to build ad-hoc CNNs for each kind of WB, instead of training a general-purpose CNN to work with all the possible WBs.
- Lastly, we studied the robustness of the classifier to three kinds of noises at different intensity levels. As a general role, the higher the contribution of the noise, the lower the classification accuracy. From a technical point of view, if we focus on the maintenance task, there may be two possible solutions to improve the robustness to noise: i) it would be possible to augment training data by adding different kinds of noisy samples so that the network would potentially learn how to correctly classify them; alternatively, ii) it would be possible to build a sub-module (whether based on AI or not) oriented at detecting noises so that the outcome of the classifier can be weighted with the score in output to the detector (basically, if the processed audio is too noisy, the classification may not be reliable so it should not be taken into account). However, there are two other aspects that, in our view, should be taken into account. First, as also mentioned above, the aleatory of the environment makes the repetition of these noisy events quite rare. Therefore, despite some classifications may not be so reliable when noises disturb audio recordings, there would be cases (potentially occurring with more frequency than current weekly/monthly inspections) where the network operates correctly given the absence of noise. Second, the results we obtain seem to validate the hypothesis that the network would be visionary able to approximate, to some extent, human hearing; a fact that could contribute, if further investigated, to also increase safety at LCs. Indeed, even though the WB is correctly functioning, it is not said to be clearly audible (e.g., give a disturbing noise) from approaching road users leading, in conjunction with other possible malfunctions or poor visibility conditions, to potential threats. Therefore, the fact that a given DL system is not able to clearly "hear" (and thus classify) the WB

could be a symptom of poor signal audibility also by road users.

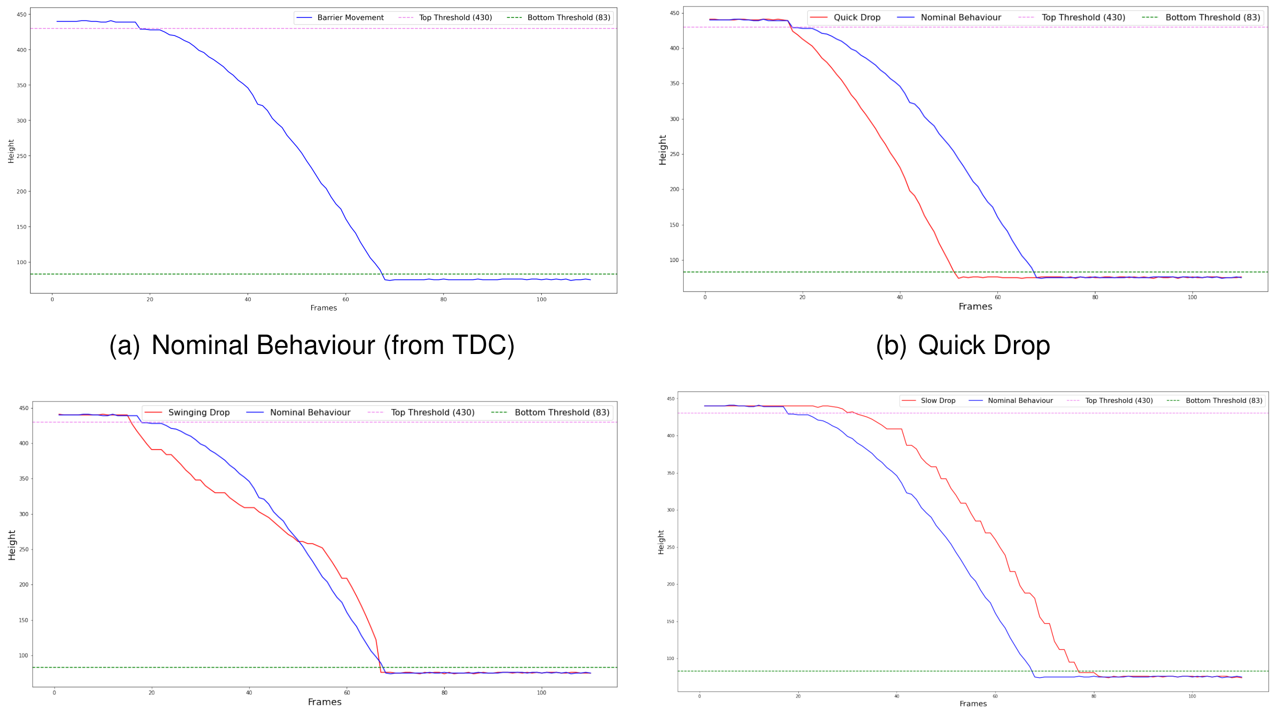
Barrier Analysis Module. As for the BAM, we adopted YOLOv5 (model 's') to detect the barrier within video frames. Data to train and test the DNN were collected by through GTA V, then, by taking advantage of the fact that the camera is fixed (task-specific characteristic), we pre-processed all the frames by cropping and keeping only the portion that could contain the barrier. Lastly, we applied some data augmentation transformations to increase the dataset dimension and aleatory in terms of light and weather conditions. Eventually, we obtained a network capable of detecting the barrier with an overall mAP@.75 of 0.99499 and a confidence of 96.05 % (on average). Nevertheless, as happened for the WBDM, the BAM is not exempt from producing FPs and FNs under some specific conditions. However, also in this case, they do not represent significant threats as they can be easily mitigated by applying some corrective algorithms as discussed in Section 4.5.3 and summarised herein:

- A FP means that the DNN produces a bounding box that does not contain any barrier. Here, we can mainly have two scenarios: i) YOLOv5s produces more than one bounding box for a given frame, hence, only one is correct; ii) YOLOv5s produces a single bounding box which is a FP. In both cases, we either select the bounding box that is closer (in terms of height) to the bounding box predicted for the previous frame or discard the prediction and set the height of the barrier in the current frame to the height of the barrier predicted for the previous frame.
- A FN means that the barrier is not detected. As for the previous case, we can simply set the height of the barrier in the current frame to the height of the barrier predicted for the previous frame.

These corrections, together with the very low miss-detection rate (0.6-0.7%), make the proposed approach viable for this task.

Barrier Anomaly Detection and the Orchestrator. Once the motion features have been extracted through YOLOv5s, these data should be processed to understand whether the barrier is functioning correctly or not. This functionality, in our architecture, would be demanded to the **Orchestrator** which, besides gathering the output of the other modules, could also include information coming from other sources when available; for example, it can consider the signal that triggers the LC when the train is approaching. In this scenario, it would be possible to keep track of the delay between the trigger and the activation of the LC macro-components to estimate whether there is a sort of significant and anomalous delay or not. Then, once this has been defined, we could adopt some metrics to estimate the degree of malfunction of the LC.

To better explain this aspect, as an example, we simulated possible malfunctions in the barrier's movement. Fig. 4.24 reports the anomalies we generated by modifying the behaviour of the barrier extracted from the TDC video (that we considered as the nominal behaviour for these tests); for this demonstration, we only took into account the phase during which the barrier is closing. The quick drop simulates the situation where the barrier closes too quickly (or starts closing earlier) compared to the nominal behaviour; the slow drop simulates the complementary event, i.e., the barrier closes too slowly or it starts closing too late; lastly, the swinging drop simulates an anomalous behaviour where the barrier does not close smoothly but it takes about the same time of the nominal behaviour. In these examples, we took into account the instant when the nominal/anomalous behaviour crosses the Top Threshold as the reference point to identify the event that the barrier is closing; in case the trigger signal can be exploited, it would be possible to start evaluating the movement of the barrier from that point. Similarly, when the nominal/anomalous behaviour crosses the Bottom Threshold, we can consider the barrier as closed. Hence, by taking into account these events, it would be possible to obtain information like those reported in Table 4.18. Numbers under the 'Time' column are reported in terms of frames; hence, considering that the videos have been collected at 30 FPS, for example, the barrier takes 50/30 (∼1,66) seconds in the nominal case. Then, metrics like the 'Time Taken (%)' and the 'MSE' (Mean Squared Error) can be used to understand the health status of the barrier.



(a) Nominal Behaviour (from TDC)

(b) Quick Drop

(c) Swinging Drop

(d) Slow Drop

Fig. 4.24. Generated Barrier's Malfunctions

Table 4.18: Metrics for Anomaly Evaluation in Barrier Movement.

Barrier Movement	Start Frame	End Frame	Time	Start Delay	End Delay	Time Taken (%)	MSE
Nominal Behaviour	*17*	*67*	*50*	/	/	/	/
Quick Drop	17	51	34	No	- 16	- 32.0 %	4623.0
Swinging Drop	15	66	51	- 2	- 1	+ 1.9 %	648.9
Slow Drop	31	76	59	+ 14	+ 9	+ 15.2 %	2272.9

Warning Light Detection Module. The same reasoning introduced above for Barrier Anomaly Detection could hold for anomaly detection in Warning Lights. As mentioned in Section 4.2.2, we did not address the WLDM as classification approaches have already received much attention in the literature. In our view, it would be possible to use any CNN to implement a simple classifier capable of detecting red (i.e., turned on) or black/brown (i.e., turned off) WLs. In addition to that, some data seem to be already available online to build a suitable dataset for this classification problem.

From a conceptual perspective, similar to the BAM, the WLDM will focus on the detection of the WLs. Then, it would be possible to associate with the "turned on" a value of 1 and with the "turned off" state a value of 0. These values can then be plotted over time to obtain the behaviour of the various WLs. Then, the Orchestrator, as already described for the barrier, could implement an algorithm to understand whether the WLs are blinking by following the nominal pattern or not.

4.6.2. Answering to Research Questions and Evaluating KPIs.

In this chapter, we discussed how AI, mainly DL, can be adopted to move towards LC Intelligent Monitoring Systems. These analyses were mainly guided by three research questions (which were identified in our previous Deliverable [2] and reported, for the sake of simplicity, in Section 4.1), which we tried to formally answer herein:

RQ1: *How can we automatically detect anomalies in LC devices, such as Barriers, Warning Lights and Warning Bells, for maintenance purposes, by using AI applied to possibly existing LC surveillance cameras and microphones?*

The DL models we investigated resulted to be quite promising to detect and extract the behaviour of LC components starting from audio and video data. Instead of using traditional and intrusive sensors, cameras and microphones (in combination with AI approaches like CNN classifiers or object detectors) can be adopted to extract motion and temporal features which can then be leveraged to evaluate the deviation of the components from their nominal behaviour. Then, in our view, it is not visionary to think that cameras that are already used at LCs for surveillance purposes can be also adopted for maintenance purposes. There is only one requirement that these cameras must meet, i.e., their field of view must include all the WLs and the whole barrier in all the positions it could assume.

RQ2: *How can we generate relevant datasets for Warning Lights, Barrier movement, and/or Warning Bell anomaly detection based on Deep Learning?*

As from our previous analyses, data availability is a sensitive issue in railways. Most of AI approaches require a lot of data to be properly trained and perform efficiently. Within this chapter, we tried to highlight some directions on how datasets can be collected at least to address the problem from the perspective of a proof-of-concept. In addition to that, we also discussed the characteristics data should have (especially for the barrier analysis module) in order to build suitable datasets. In our view, these investigations can serve as guidelines to understand how real data should be collected and which class of AI approaches should be used in combination to easily migrate the LC monitoring system to real scenarios.

RQ3: *Can we demonstrate possible answers to RQ1 and RQ2 through a simple proof-of-concept demonstrator in order to inspire future developments and a technology roadmap?*

This proof-of-concept allowed us to investigate different aspects related to AI and data collection. With our analyses, we basically laid the ground to facilitate future and more practical developments in the context of LC continuous monitoring. The main findings, recommendations, and directions towards technology roadmaps will be discussed in detail in our future deliverables.

In addition to that, in [2], we also defined some KPIs (see Table 4.1) that would have helped us to evaluate from a high-level perspective the suitability of the proposed monitoring system:

KPI1: Implementability Costs. *Includes costs of sensors, installation, possible re-approval processes, and consumption.*

Based on our perception, cameras and microphones would allow avoiding any kind of re-approval process. In addition to that, safety at LCs has recently become one of the most discussed aspects in railways and different systems have been proposed to monitor the LC area by means of cameras (e.g., to detect possible obstacles). This fact makes camera sensors even more appealing and more cost-effective given their dual usability. The same may go for microphones because, as discussed in Section 4.6.1, DL approaches could somehow approximate human capability of hearing WBs, therefore, also these sensors can be used for both maintenance and safety purposes. In our view, their multiple usability makes the new installation of these sensors more advantageous compared to other kinds of sensors that would be oriented to collect data for maintenance purposes only.

KPI2: Computation Time. *Indicates the time required by the system to detect possible malfunctions.*

The approaches we implemented are capable of detecting WBs frames in about 1.13 ms and the barrier within a single frame in about 10.36 ms. However, it is important to mention that whether computation times are appropriate or not also depends on the sensors. For example, if the installed camera works at 30 FPS, it means that it will capture a frame each 33.3 ms; therefore, even though the barrier detection took more than 10.36 ms to detect the barrier in a single frame, it would have been still suitable to work in real-time with 30 FPS cameras as long as the computation time was lower than 33.3 ms. For the sake of knowledge, in the case of 60 FPS cameras, the computation time per video frame should be lower than 16.6 ms to allow for real-time monitoring. From this perspective, the results we obtained are quite efficient as there is still room to increase the accuracy of the model, even in case this improvement should be paid in terms of a few milliseconds, and implement anomaly detection mechanisms while remaining suitable for real-time operations.

KPI3: Effectiveness. *Indicates how the monitoring system would perform in terms of accuracy and detectable malfunctions.*

At this stage, the WBDM and BAM are capable of detecting the WBs and the barrier with quite high accuracy. As for the barrier specifically, a sort of anomaly score can be deducted by means of the MSE which compares the captured behaviour with the nominal one. Future work will be oriented at understanding whether and how anomalies can be predicted to estimate the remaining useful life of the components.

5. Predictive Maintenance for Rolling Stock

5.1. Introduction

Similar to the smart maintenance on Level Crossings, we also introduced relevant methodology and expected results on the topic of predictive maintenance for rolling stock in the previous deliverables D3.1 [1] and D3.2 [2], which predictively scheduling an effective maintenance plan in accordance with the aim of minimizing maintenance costs for the whole Rolling Stock fleet, has arisen significant discussions in terms of its performing efficiency or economically acceptable way. This is inspired by the high-level idea we already proposed in D3.1 on railway station infrastructure/system or vehicle components for maintenance purposes. From the high-level perspective, rolling stock maintenance is a significant aspect of the maintenance activities occur in railway stations. The current maintenance type performed on rolling stock components are typically intrusive maintenance, and such maintenance has to be conducted in the depots or station hubs, in a static manner. With these considerations, we narrow down our scope to the AI-based predictive maintenance for rolling stock components, by borrowing ideas derived from reinforcement learning. Therefore, as also suggested by the Advisory Board, they are a relevant case study to propose benchmarks, proofs-of-concept, and roadmaps.

It is crucial to continually monitor the rolling stock health condition due to preventive maintenance of railroad equipment is also required by a number of laws. Usually, these maintenance activities are carried out in a fixed periodic manner. That is, a time-based maintenance policy or a distance-based maintenance cycle. On the one hand, It is economically reasonable that maintenance activities should be carried out at the latest possible date (only if when the maintenance task has to be done as soon as possible) due to their high financial cost, i.e., based on time periods. On the other hand, normally the vehicles undertake various cyclical tasks assigned by transport orders and specific transport segments/routes, which are characterised by average daily/monthly mileage (the assigned workload). They may cause earlier maintenance as unexpected workloads will increase actual running mileage and thus shorten associated distance-based policy. However, a maintenance schedule for each vehicle impacted by the volume of work can be limited within a certain range by their timetable. In order to reduce the overall cost of uniform rolling stock maintenance activities, we shall represent the discussed topic in this case study as the job scheduling problem. Based on this, we suggest a reinforcement learning-based algorithm that incorporates Travelling Salesman Problem (TSP) and Intelligent Rolling Stock Rostering (IRSR) to deliver satisfactory results from the standpoint of industrial practice.

According to the past studies, [14] presented a novel approach to address the challenge of preventive maintenance routing by developing a multicommodity flow model. Their focus was on improving practical solutions by introducing limited modifications to a macroscopic rolling stock plan, where rolling stock units operate on lines connecting aggregate stations. The main objective of their model was to minimize shunting plan deviations, which are disruptions to the planned movement of rolling stock units. In a similar [15] proposed a multicommodity flow model specifically designed for optimizing the circulation of rolling stock units on a single line within the Dutch railway network. Their objective centered around minimizing the total kilometers traveled by train units of different types. Notably, the consideration of maintenance requirements was absent from their formulation, highlighting a potential area for further enhancement in future research efforts.

Reinforcement learning (RL) has emerged as a powerful approach in the field of decision-making and optimization. Its ability to learn from interactions with the environment and derive optimal strategies makes it a promising tool for addressing complex scheduling and rostering problems in various domains. One such domain is rolling stock rostering, which involves the efficient allocation of maintenance activities for trains while ensuring smooth operations and minimizing disruptions. Rolling stock rostering encompasses the scheduling of maintenance tasks such as inspections, repairs, and preventive maintenance activities for a fleet of trains. It is a challenging task due to the dynamic nature of the maintenance requirements and the need to balance operational efficiency with maintenance needs. Traditional approaches to rolling stock rostering often rely on heuristic methods or mathematical programming techniques, which may not capture the complexity and adaptability required for real-world scenarios. In recent years, the application of reinforcement learning techniques to rolling stock rostering has gained attention as a potential solution to address these challenges. By formulating the rostering problem as a reinforcement learning task, we can leverage the ability of RL algorithms to learn optimal policies through interactions with the environment. This approach allows for more adaptive and dynamic decision-making, as the RL agent can continuously update its strategies based on feedback from the system.

Effectively incorporating reinforcement learning into rolling stock rostering involves several key considerations. First and foremost, it requires a well-defined state space representation that captures relevant information about the fleet, maintenance requirements, and operational constraints. The choice of state representation greatly impacts the learning process and the agent's ability to make informed decisions. Secondly, an appropriate action space needs to be defined to allow the RL agent to select maintenance activities and allocate resources efficiently. This includes decisions related to maintenance priorities, assignment of maintenance teams, and scheduling of tasks within specific time windows. Furthermore, the reward function plays a crucial role in reinforcement learning. Designing an appropriate reward function is essential to guide the agent towards desired outcomes, such as minimizing delays, reducing maintenance costs, and ensuring compliance with safety regulations. It is important to strike a balance between short-term and long-term rewards, as well as consider any trade-offs between conflicting objectives. Additionally, the integration of domain-specific knowledge and constraints is crucial for effective reinforcement learning in rolling stock rostering. Incorporating expert knowledge, historical data, and operational constraints can enhance the learning process and lead to more realistic and robust solutions.

In this paper, we aim to explore and propose methodologies for effectively incorporating reinforcement learning techniques into rolling stock rostering, specifically focusing on the optimization of maintenance activities. We will investigate various RL algorithms and architectures, along with different state representations, action spaces, and reward functions. Through extensive experimentation and evaluation, we aim to demonstrate the benefits and limitations of RL-based approaches and provide insights into their practical implementation in real-world rolling stock rostering scenarios. By leveraging the capabilities of reinforcement learning, we anticipate that our research will contribute to more efficient and effective rolling stock rostering, ultimately leading to improved operational performance, reduced maintenance costs, and enhanced overall system reliability.

5.2. Model Description

The proposed model for incorporating reinforcement learning into rolling stock rostering with maintenance activities consists of two fundamental modules: Rostering and Maintenance. After processing the train services data and railway network information in these modules, the model further includes a reinforcement learning module, where the Agents, Environment, States, Actions, and Reward to facilitate the learning process have been properly constructed and defined. From an overall perspective, it is advisable to construct a diagram below (see Fig 5.1 that illustrates data feeding, processing, modeling, and predicting more explicitly.

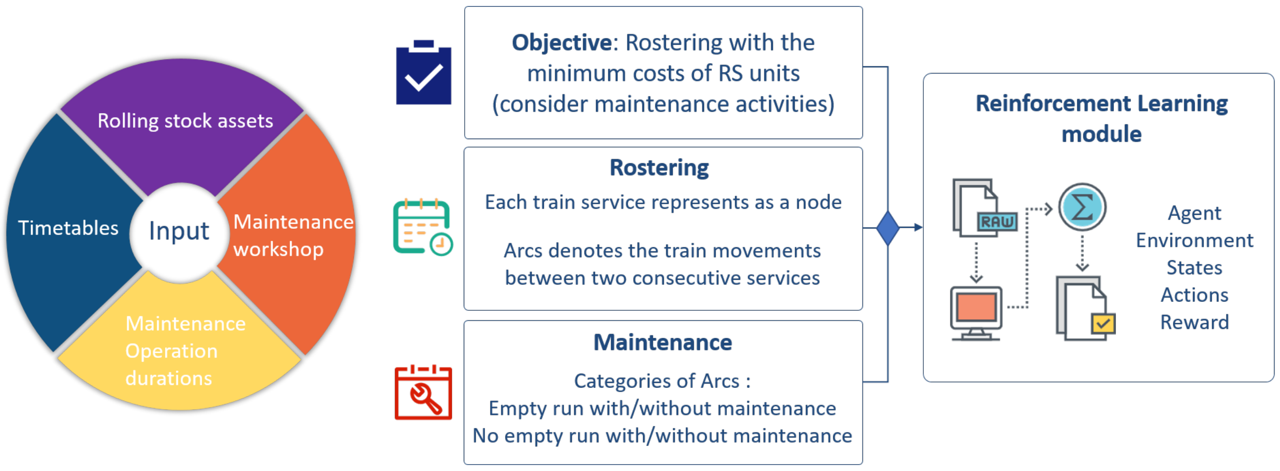
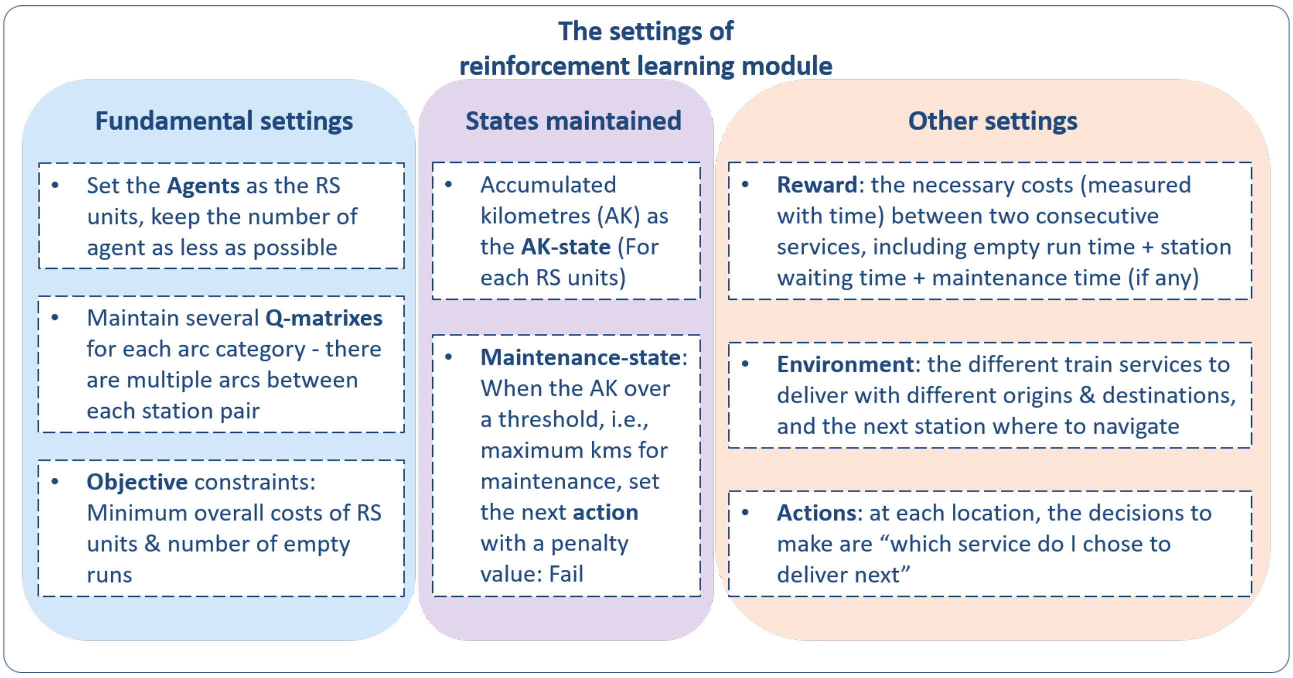

Rolling stock assets
Timetables
Input
Maintenance workshop
Maintenance Operation durations
Objective: Rostering with the minimum costs of RS units (consider maintenance activities)
Rostering
Each train service represents as a node
Arcs denotes the train movements between two consecutive services
Maintenance
Categories of Arcs :
Empty run with/without maintenance
No empty run with/without maintenance
Reinforcement Learning module
RAW
Agent
Environment
States
Actions
Reward

Fig. 5.1. High-Level Architecture for Rolling Stock Intelligent Rostering and Maintenance.

Rolling Stock Rostering module is the procedure of processing raw data into more suitable characterization inputs for subsequent modules. In this module, the train services data and railway network information are processed to optimize the allocation and scheduling of train services. Each train service is represented as a node or task, and arcs are introduced to denote the locomotive movements between consecutive train services. The goal is to efficiently assign rolling stock units to train services while considering the usage of locomotives to be minimized. At the same time, the punctuality and reliability of all scheduled passgenger services should be guaranteed.

Rolling Stock Maintenance module focuses on the allocation and scheduling of maintenance activities for all the 'active' rolling stock units, i.e., those units that have been introduced in the system for undertaking the actual train services. It incorporates four types of arcs: empty run with maintenance, empty run without maintenance, no empty run with maintenance, and no empty run without maintenance. The objective is to optimize the planning process of maintenance activities while minimizing the mileage of empty runs between maintenance workshop and the service delivery origin stations.

Reinforcement Learning module takes the data from the two modules described above. This module consists of the following components: 1) Agent: The RL agent is the decision-maker in the system. It receives observations from the environment, takes actions, and learns from the outcomes. The agent's role is to determine the optimal decisions regarding rolling stock rostering and maintenance activities. 2) Environment: The environment represents the rolling stock rostering and maintenance system. It provides the RL agent with observations and receives actions in return. The environment incorporates the processed train services data, railway network information, and relevant constraints. It simulates the dynamics of the system and generates rewards based on the agent's actions. 3) States: The states in the RL module represent the information that the agent observes from the environment. They capture relevant data such as the current status of rolling stock units, the availability of the remaining services, the progress of maintenance activities, and the railway network conditions. The state space is designed to provide the agent with sufficient information to make informed decisions. 4) Actions: The actions refer to the decisions made by the RL agent. In the context of rolling stock rostering and maintenance, actions can include assigning rolling stock units to train services, determining the movement of locomotives between tasks, scheduling maintenance activities, and allocating maintenance resources. The action space is defined to encompass all possible decisions required for effective system management. 5) Reward: The reward function in the RL module quantifies the desirability of the agent's actions. It provides feedback to the agent, guiding it towards learning optimal policies. The reward function takes into account various factors, such as minimizing delays, reducing maintenance costs, ensuring compliance with safety regulations, and improving system performance. It balances short-term objectives with long-term goals and accounts for any trade-offs between conflicting objectives. The relationships between each of RL components have been illustrated in the Fig 5.2 below.


The settings of reinforcement learning module

Fundamental settings

- Set the **Agents** as the RS units, keep the number of agent as less as possible
- Maintain several **Q-matrixes** for each arc category - there are multiple arcs between each station pair
- **Objective** constraints: Minimum overall costs of RS units & number of empty runs

States maintained

- Accumulated kilometres (AK) as the **AK-state** (For each RS units)
- **Maintenance-state**: When the AK over a threshold, i.e., maximum kms for maintenance, set the next **action** with a penalty value: Fail

Other settings

- **Reward**: the necessary costs (measured with time) between two consecutive services, including empty run time + station waiting time + maintenance time (if any)
- **Environment**: the different train services to deliver with different origins & destinations, and the next station where to navigate
- **Actions**: at each location, the decisions to make are "which service do I chose to deliver next"

Fig. 5.2. The settings of reinforcement learning module.

By combining the Rostering and Maintenance modules with the reinforcement learning module, the proposed model aims to find an integrated solution that optimizes rolling stock rostering, by combining the constraints of locomotive movements, and the minimum maintenance activities. The RL agent, trained using reinforcement learning techniques, learns to make decisions that maximize the overall system performance and reliability, taking into account the complex interactions between rolling stock operations and maintenance requirements.

5.3. Model Implementation

The implementation of the RL-based rolling stock rostering and maintenance model involves the integration of various components we described in last section to create a comprehensive decision-making framework. At its core, the model leverages Reinforcement Learning (RL) techniques, specifically the Deep Q-Network (DQN) algorithm, to enable intelligent decision-making by rolling stock units. The implementation encompasses the state representations of rolling stock unit as individual DQN agent, the definition of state and action spaces, the training of the agents through interactions with the environment, and the utilization of learned policies to optimize rostering and maintenance decisions. By combining these elements, the model provides a powerful tool for enhancing the operational efficiency and effectiveness of rolling stock management.

5.3.1. Rolling Stock Unit State & Service Representation

Let's decompose a set of typical vehicle movements as shown in the Fig 5.3: the time given by roster between two consecutive train services can be represented as

- waiting time at the passenger station before the train is brought to workshop
- time to provide empty run to workshop
- maintenance processing time window (including recovery time)
- time to provide empty run to passenger station
- waiting time at the passenger station after the train leaves the workshop

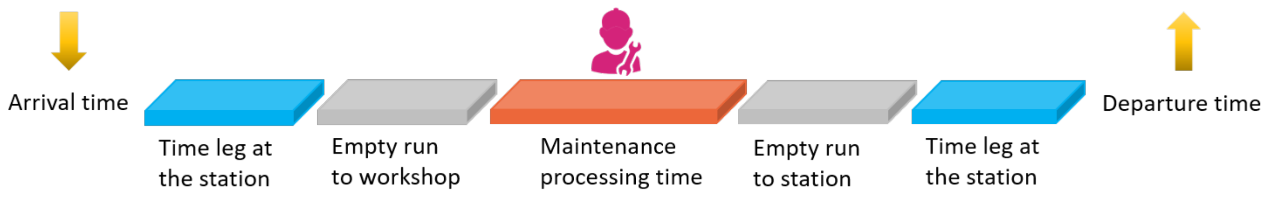


Fig. 5.3. Cycle of the RS rostering

Fig 5.4 shows a small graph to illustrate the problem formulation. For each train service, there is a node with labels indicating departure and arrival stations (A, B, C and D), plus the associated service travelling times. The solid green arcs indicate paired services, the solid purple arcs denote the empty rides without maintenance, the dotted purple arcs are the empty rides with maintenance tasks, the dotted green arcs represent the maintenance tasks without empty rides. The numerical labels show arc costs, Note there is only one kind of maintenance type is going to perform. A solution is an Hamiltonian path with one or more maintenance arcs to guarantee the maintenance expiry. In the solution the empty rides (purple arcs) are optional.

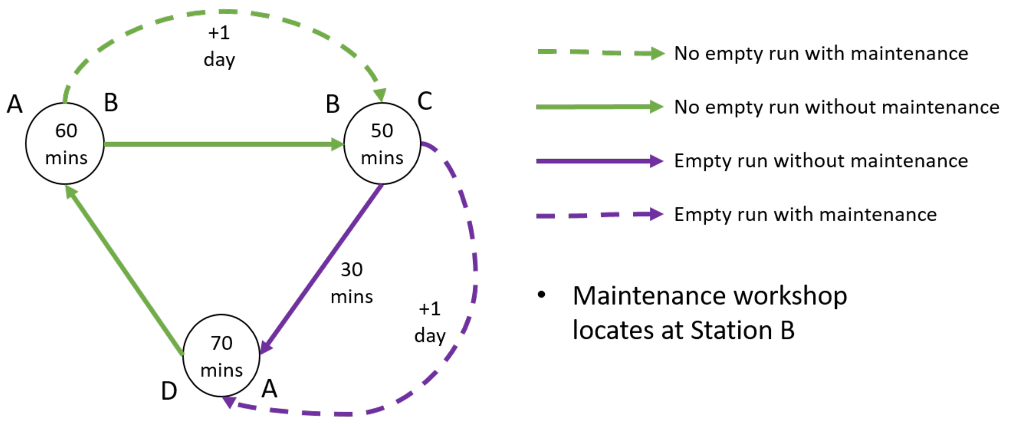

+1
day
A
B
60
mins
B
C
50
mins
30
mins
+1
day
70
mins
D
A
No empty run with maintenance
No empty run without maintenance
Empty run without maintenance
Empty run with maintenance
Maintenance workshop locates at Station B

Fig. 5.4. An illustrative example of four kinds of locomotive movements

In the proposed model, the state of a rolling stock unit is represented by a combination of four attributes: maintenance history, operational conditions, position, and mileage. These attributes capture important information about the unit's past maintenance, current operational state, physical location, and accumulated mileage.

Maintenance History: This attribute tracks the maintenance history of the rolling stock unit. It represents the number of maintenance events the unit has undergone since its initial state. The maintenance history provides a measure of the unit's reliability and maintenance requirements. This may not be an extensively utilised attribute in the current study paradigm but it would provide more insights about how to balance the usage between heavily maintained and less maintained units.

Operational Conditions: The operational conditions attribute describes the current state of the rolling stock unit. It categorises the unit into one of the six operational conditions:

State(0) - Available: The unit is in the shared pool, waiting to be assigned for the next round of tasks.

State(1) - Travel from the Workshop: The unit has completed its standby status and is en route to the origin of a service to deliver it.

State(2) - Operational Running: The unit is actively performing a service, with detailed information such as departure/arrival time, stations, and total travel distance/time provided by the assigned service.

State(3) - Travel for Operational Task: The unit has completed a service but has not reached the 'Accumulated Maximum Miles' threshold for maintenance. It remains in the system and is available to deliver the next service. There may be distances and travel time between the destination of the previous service and the origin of the next service.

State(4) - Travel to the Workshop: When the unit's mileage reaches the 'Accumulated Maximum Miles', its operational condition transitions to this state, indicating that it needs to travel to the maintenance workshop for the routine maintenance.

State(5) - During Maintenance: The unit is undergoing maintenance at the workshop. During this state, the 'mileage' and 'position' attributes remain unchanged, but additional time costs are incurred. Each maintenance site is dedicated to specific types of maintenance work, such as: interior or exterior cleaning, refuel (only for diesel units), regular inspection, repair (scheduled or not) and technical check-up. Each type of maintenance task must be performed regularly, i.e. within a maximum time limit or a maximum number of kilometres from the last maintenance of the same type. Since performing some maintenance task too often would cause an unnecessary cost for the company, each type of maintenance task should be performed in the proximity of its maximum limit. In fact, maintenance tasks impose severe constraints and their effect on the line capacity is difficult to analyze.

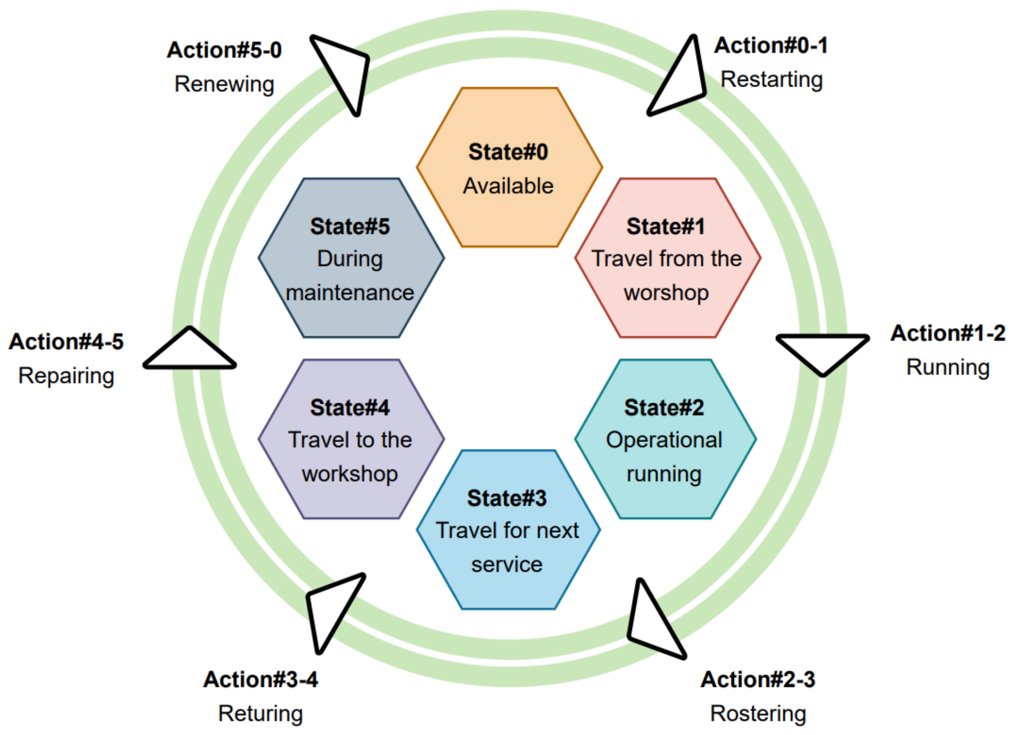


Fig. 5.5. Cycle of the States for four Rolling Stock Units

Shown as Fig 5.5, these six operational conditions form a closed loop, and the unit resets to state (0) after completing maintenance.

Position: The position attribute represents the current physical location of the rolling stock unit. It is defined using the station index, which maps each station name to a unique integer index. The position provides information about where the unit is located within the railway network.

Mileage: The mileage attribute indicates the accumulated mileage of the rolling stock unit. It represents the total distance travelled by the unit since its initial state. The mileage provides insights into the wear and tear of the unit and helps determine when maintenance is required.

5.3.2. Environment Dynamics

The Rolling Stock Environment operates in an episodic manner, where each episode represents a specific time period or planning horizon. The environment consists of a fixed number of rolling stock units and a set of services to be delivered. At the beginning of each episode, the environment is reset, and the rolling stock units are initialised with their respective initial states. Each of them is introduced into the rostering process interactively - individual unit as an agent will pick the first service (the most urgent one) to perform, unless there is no remaining service or this specific unit has to be brought back for maintenance. After finishing an episode all the services will be released back to the 'service pool' for next round picking-up. Four possible rostering cases are illustrated in the Fig 5.6 following.

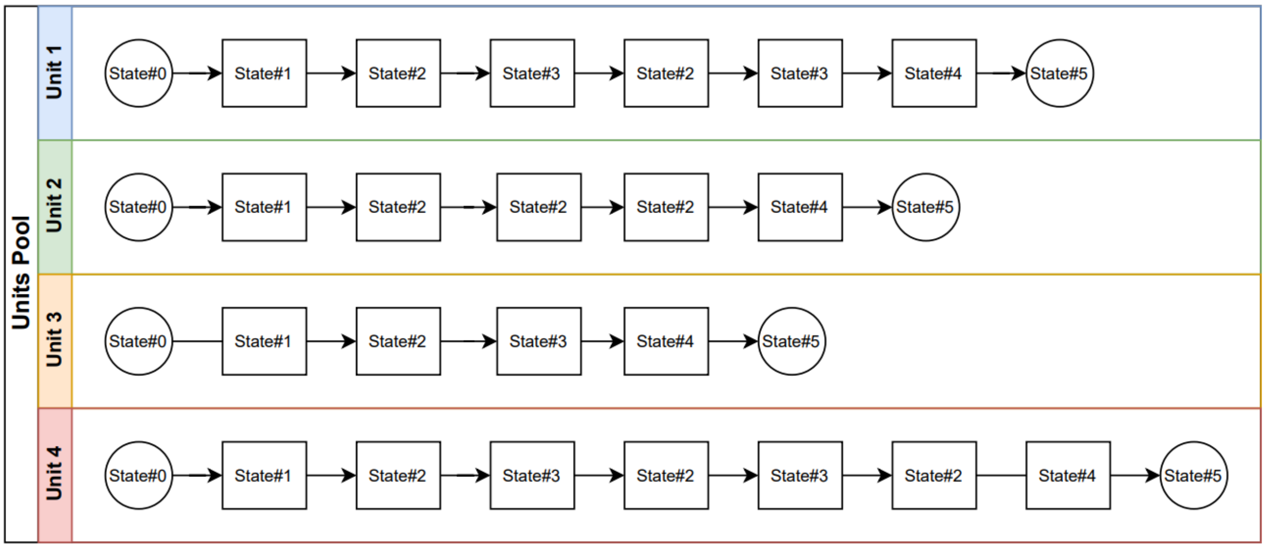


Fig. 5.6. 4 Examples of the RS Rostering

During each step of an episode, the agents (rolling stock units) select actions based on their current states. The actions determine the next state of the units and potentially trigger rewards. The actions available to the agents include:

- Action 0 - Perform a Service: The agent selects this action to deliver a service. The specific service to be delivered is determined based on the unit's current operational conditions, position, and available services. The unit may need to travel from its current position to the origin of the service and then deliver the service to the destination.
- Action 1 - Perform Maintenance: The agent selects this action when the unit's mileage reaches the 'Accumulated Maximum Miles' threshold, indicating the need for maintenance. The unit then transitions to the 'Travel to the Workshop' state and travels to the maintenance workshop for necessary maintenance activities.
- The environment dynamics are driven by the selected actions and the constraints defined by the model, such as maximum mileage thresholds and availability of services. The distance and time rewards are computed based on the unit's actions, the distances travelled, and the time taken to perform services or travel.

The episode terminates when the maximum number of steps is reached or when there are no more services available to be delivered.

5.4. Training and Validation

The proposed approach employs the Deep Q-Network (DQN) algorithm to generate schedules for rolling stock units and enable them to make optimal decisions between consecutive services and maintenance. Each rolling stock unit is represented by an individual DQN agent, which interacts with the environment to undertaking the passenger services and receiving maintenance callings. By observing the current state and selecting actions based on the learned Q-values, it chooses the least-cost (i.e., maximum positive reward received) action to perform. The Q-values in this study, are designed as the expected future rewards for taking specific actions in a given state, which are updated using experience replay. This process involves storing past experiences (consisting of the state, action, reward, and next state) and randomly sampling from this memory to train the neural network models of the agents.

During the training process, the model iteratively runs multiple episodes, allowing the agents to learn and improve their decision-making abilities over time. To achieve optimal learning and convergence, the hyperparameters of the model, including the learning rate, discount factor, exploration rate, and batch size, are carefully tuned. These hyperparameters play a crucial role in shaping the agents' learning dynamics and determining the balance between exploration and exploitation.

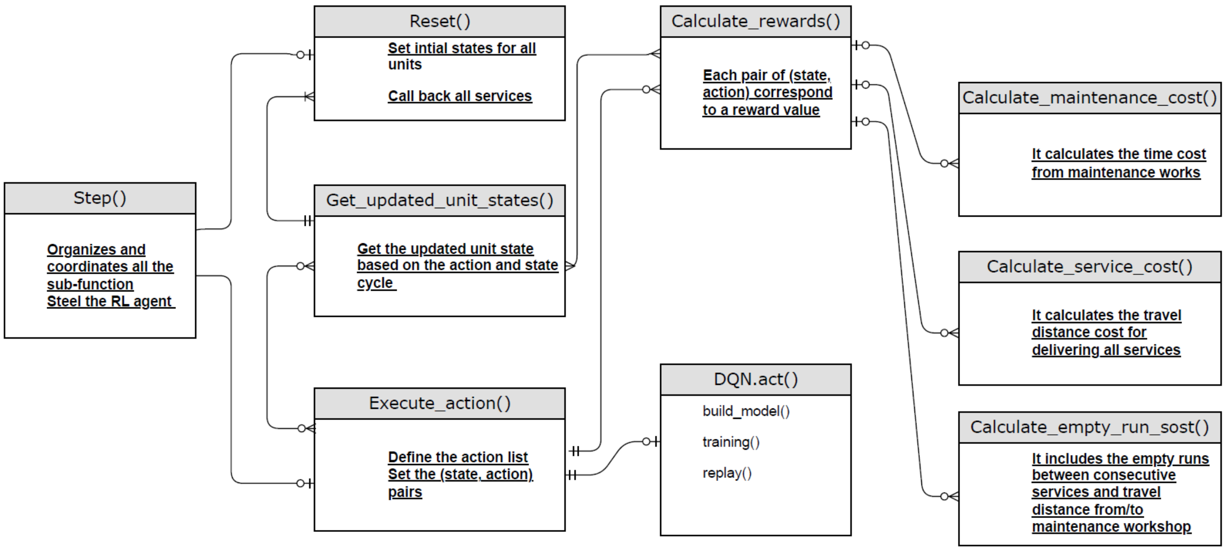

Reset()
Set intial states for all units
Call back all services
Calculate_rewards()
Each pair of (state, action) correspond to a reward value
Calculate_maintenance_cost()
It calculates the time cost from maintenance works
Step()
Organizes and coordinates all the sub-function
Steel the RL agent
Get_updated_unit_states()
Get the updated unit state based on the action and state cycle
Calculate_service_cost()
It calculates the travel distance cost for delivering all services
DQN.act()
build_model()
training()
replay()
Execute_action()
Define the action list
Set the (state, action) pairs
Calculate_empty_run_sost()
It includes the empty runs between consecutive services and travel distance from/to maintenance workshop

Fig. 5.7. The Software workflow for each episode of training

By training the model using the DQN algorithm, the rolling stock units gain the ability to make informed decisions about the optimal assignment of units to services. The trained model takes into account various factors, including the specific requirements of each service, the conditions of the rolling stock units, and the operational constraints of the system. By considering these factors, the model can effectively allocate units to services, optimizing the overall performance and efficiency of the rolling stock operation.

The trained model represents a valuable tool for decision support in the rolling stock domain. It empowers operators and decision-makers to make data-driven decisions, ensuring that the available rolling stock units are assigned to services in an optimal and effective manner. By leveraging the learned Q-values and the insights gained during training, the model provides a basis for enhancing operational efficiency, minimizing downtime, and improving overall service quality in the rolling stock industry.

In summary, the training process using the DQN algorithm equips the rolling stock units with the capability to make informed decisions, taking into account various factors and optimizing the assignment of units to services. The trained model serves as a valuable decision support tool, providing operators with the means to optimize the performance and efficiency of rolling stock operations. The fig 5.9 gives the information about how the implemented function interact and the hierarchical relations between them.

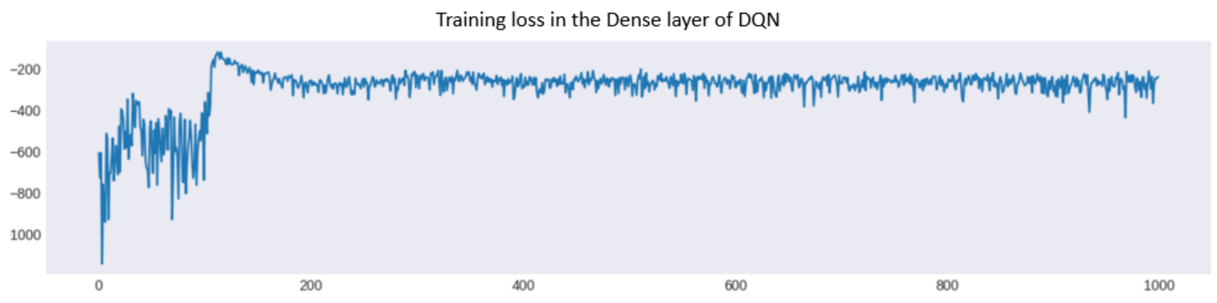


Fig. 5.8. The training loss curve of the RL based rolling stock rostering

The curve shown in Fig 5.8 denotes the overall training loss calculated with the training episode growing. As we can see, the training loss remains stable after 200 episodes.

5.5. Evaluation of Results

When we start training and experiencing the same problem overtime, the agent learns positive/negative rewards from the environment, this is shown by the episode rewards values for each experience replay.

The evaluation of the RL model results demonstrates the effectiveness of the exploration-exploitation trade-off strategy employed during the training process. In the initial phase of the training, spanning the first 400 episodes, the model actively explores different rostering schedules by employing an epsilon-greedy approach. The DQN.act() function selects actions based on the current state, calculating Q-values for all possible actions and choosing the action with the highest Q-value with a probability of (1 - epsilon). However, with a probability of epsilon, the model randomly selects an action from the available action list, facilitating exploration.

The epsilon value plays a crucial role in controlling the exploration-exploitation balance. As the training progresses, the epsilon value gradually decays based on the 'epsilon decay' rate (set to 0.999). This decay results in a lower probability of randomly selecting actions and a higher reliance on the learned Q-values. Consequently, in the second phase of the training, the model transitions from exploration to exploitation, leveraging the knowledge acquired from previous episodes. It gradually reduces the number of random actions taken and becomes increasingly driven by the Q-values, leading to more focused and optimal decision-making.

The evaluation of the RL model in these two phases showcases the model's ability to strike a balance between exploration and exploitation. The initial exploration phase allows the model to discover potentially optimal rostering schedules and gather valuable experience. Subsequently, as the model progresses and the epsilon value decreases, it leverages this experience to exploit the learned knowledge and make more informed decisions.

By carefully managing the exploration-exploitation trade-off and progressively shifting towards exploitation, the RL model demonstrates its capacity to learn and improve over time. This evaluation highlights the model's ability to adapt its decision-making strategy and converge towards optimal rostering schedules for rolling stock units.

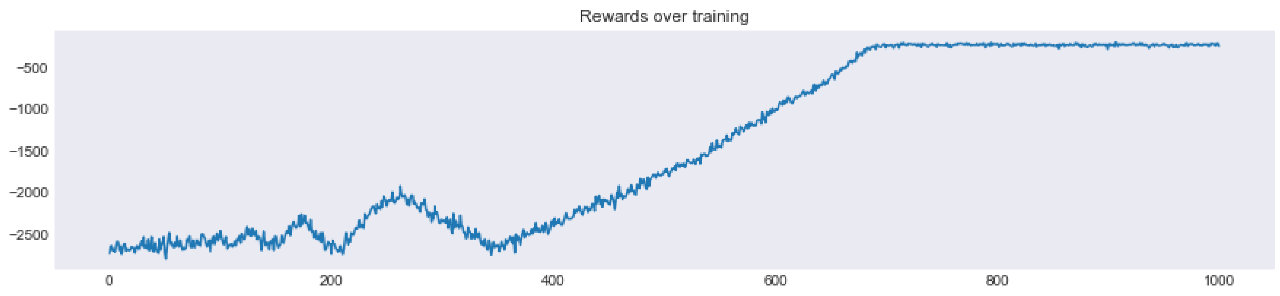


Fig. 5.9. The overall rewards for each episode of training

5.6. Discussion of Results

The Traveling Salesman Problem (TSP) and Rolling Stock Rostering Problem (RSRP) are typically optimized by a set of specific requirements and constraints according to the problem contexts, where one of the commonsense is it has to find the shortest (the most efficient) route to visit different cities/stations. Many different ways have been proposed to solve this problem using discrete optimization techniques. Like many optimization problems [16], RSRP is a NP-hard problem, which in short means that it's easy (in terms of speed) to solve for 5 or 50 services, already impossible to brute force for 500. And almost impossible for most algorithms for 5,000 services if we still consider maintenance activities to be inserted. Even for 15 services, it's already 1 trillion permutations to compute (i.e., 15!), there are optimization techniques that are more adequate : dynamic programming [17], branch and bound algorithms [18], nearest neighbors approximations [19], or ant colonies optimizations [20].

Furthermore, the problem described in this study is too simple to be delivered in a real-life rostering/maintenance situation. It does not take into account multiple vehicle fleet types, track capacity constraints, In-station dispatch constraints, aleatory perturbations, etc... Hence, each variant of the RSRP has its own optimization frameworks (in terms of variables and constriants). More complexity (e.g., rolling stock maintenance, vehicle refuelling) we incorporate into the problem, the more difficult it would be properly addressed. That's why in practice delivery companies use combinations of those variants coupled with heuristics or AI components. The most advanced companies today add Machine Learning techniques on top of those algorithms, in particular to replace manual heuristics and better estimate in-context durations.

In this PoC, it is promising to conclude that one of the effective ways to get started with RSRP and its variants is probably the implementation of neural network-based Reinforcement Learning framework (Deep Q-learning techniques) due to it holds many advantages compared to classical optimization techniques:

- Offering a general framework for all problems, indeed instead of tweaking the constraints and defining extra variables, we can change the reward, and defining a multi agent problem if needed for rostering optimization. Adding extra information like flexible maintenance threshold is also eased if we can integrate the predictive maintenance algorithm with a similar ML techniques (e.g. Pure Neural Networks)
- Having a ”dynamic” decision making algorithm. Because a RL alorithm is trained iteratively by making the next rostering decision at each state, compared to ”offline” optimization algorithms that study the problem with no unknowns, it inherently would be able to take different decisions if something happened during the experience, which avoids recalculating with classical techniques or starting the computation from scratch again.
- Being robust to unknowns and aleatory perturbations.

6. Conclusions

In this deliverable, the potential of AI solutions towards smarter railway maintenance and inspection activities has been investigated through two experimental PoCs. Two specific case studies, namely, “Smart Maintenance at Level Crossings” and “Predictive Maintenance for Rolling Stock”, partially analysed in previous WP3 deliverables, have now been deeply investigated. Diverse AI approaches have been investigated to cope with tasks and sub-tasks related to the aforementioned case studies. The findings from these explorations indicate that AI could be a valuable asset in enhancing the efficiency and effectiveness of railway maintenance procedures.

The two PoCs analysed in this report are not meant to be the conclusive application of AI, but rather a stepping stone, providing the initial inspiration and setting the stage for future advancements and the development of a comprehensive technology roadmap. These studies represent the first steps in the exploration of AI’s potential in railway maintenance, setting the foundation for subsequent research.

Deliverable D3.4 will offer an in-depth analysis of the results from each case study. It identifies opportunities and gaps, and assesses the strengths and weaknesses observed in the implementation of AI. This will provide valuable insights to inform future exploration and aid in shaping the trajectory of AI applications in railway maintenance and inspection, thereby bringing us closer to realising a smarter, more efficient railway system.

Bibliography

[1] RAILS, "Deliverable 3.1 – WP3 Report on case studies and analysis of transferability from other sectors (predictive maintenance and defect detection)," 2021. [Online]. Available: https://rails-project.eu/downloads/deliverables

[2] ——, "Deliverable 3.2 – WP3 Report on AI approaches and," 2022. [Online]. Available: https://rails-project.eu/downloads/deliverables

[3] N. Rail, "Nr/l2/sig/19608 - level crossing asset inspection and implementation of minimum action codes," 2014. [Online]. Available: https://standards.globalspec.com/std/1689019/nr-l2-sig-19608

[4] ——, "Nr/l2/sig/19608 issue 8 - inspection of level crossing systems," 2021. [Online]. Available: https://standards.globalspec.com/std/14461410/nr-l2-xng-19608-issue-8

[5] SAFER-LC, "Deliverable d1.3: Needs and requirements for improving level crossing safety," 2018.

[6] O. of Rail Regulation, "Level crossings: A guide for managers, designers and operators," 2011. [Online]. Available: https://www.orr.gov.uk/sites/default/files/om/level_crossings_guidance.pdf

[7] S. Hershey, S. Chaudhuri, D. P. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold *et al.*, "CNN architectures for large-scale audio classification," in *Int. Conf. on Acoustics, Speech and Signal Processing*. IEEE, 2017, pp. 131–135.

[8] R. Kulkarni, S. Dhavalikar, and S. Bangar, "Traffic light detection and recognition for self driving cars using deep learning," in *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, 2018, pp. 1–4.

[9] Q. Wang, Q. Zhang, X. Liang, Y. Wang, C. Zhou, and V. I. Mikulovich, "Traffic lights detection and recognition method based on the improved yolov4 algorithm," *Sensors*, vol. 22, no. 1, p. 200, 2022.

[10] L. De Donato, F. Flammini, S. Marrone, C. Mazzariello, R. Nardone, C. Sansone, and V. Vittorini, "A survey on audio-video based defect detection through deep learning in railway maintenance," *IEEE Access*, pp. 1–1, 2022.

[11] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural Computation*, vol. 29, no. 9, pp. 2352–2449, 2017.

[12] Z. He, "Deep learning in image classification: A survey report," in *2020 2nd International Conference on Information Technology and Computer Application (ITCA)*, 2020, pp. 174–177.

[13] J. F. Gemmeke, D. P. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter, "Audio set: An ontology and human-labeled dataset for audio events," in *Int. Conf. on Acoustics, Speech and Signal*. IEEE, 2017, pp. 776–780.

[14] G. Maróti and L. Kroon, "Maintenance routing for train units: the transition model," *Transportation Science*, vol. 39, no. 4, pp. 518–525, 2005.

[15] A. Alfieri, R. Groot, L. Kroon, and A. Schrijver, "Efficient circulation of railway rolling stock," *Transportation Science*, vol. 40, no. 3, pp. 378–391, 2006.

[16] Q. Zhong, R. M. Lusby, J. Larsen, Y. Zhang, and Q. Peng, "Rolling stock scheduling with maintenance requirements at the chinese high-speed railway," *Transportation Research Part B: Methodological*, vol. 126, pp. 24–44, 2019.

[17] K. S. Moghaddam and J. S. Usher, "Preventive maintenance and replacement scheduling for repairable and maintainable systems using dynamic programming," *Computers & Industrial Engineering*, vol. 60, no. 4, pp. 654–665, 2011.

[18] C. Sriskandarajah, A. K. Jardine, and C. Chan, "Maintenance scheduling of rolling stock using a genetic algorithm," *Journal of the Operational Research Society*, vol. 49, no. 11, pp. 1130–1145, 1998.

[19] Y. Wang, A. D'Ariano, J. Yin, L. Meng, T. Tang, and B. Ning, "Passenger demand oriented train scheduling and rolling stock circulation planning for an urban rail transit line," *Transportation Research Part B: Methodological*, vol. 118, pp. 193–227, 2018.

[20] Y. Tsuji, M. Kuroda, Y. Kitagawa, and Y. Imoto, "Ant colony optimization approach for solving rolling stock planning for passenger trains," in *2012 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2012, pp. 716–721.