Detecting Road Surface Anomalies using Multimodal Machine Learning

Joël Luijmes
Student number: 2031107
Email: me@joell.app

THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE IN DATA SCIENCE AND ENTREPRENEURSHIP
JHERONIMUS ACADEMY OF DATA SCIENCE

Supervisors:
Prof. Dr. Willem-Jan van den Heuvel
Dr. Dario Di Nucci

Jheronimus Academy of Data Science
1 December 2021

Abstract

The current process of performing road maintenance takes long, is costly and largely relies on subjective observations. In recent years, the need for data driven inspection emerged to perform automatic road quality assessment. In this work, we contribute a novel method combining both visual and accelerometer data to detect road surface anomalies. We evaluate the performance of the multimodal model with that of respective unimodal models. The visual based model performs best with an precision of 0.86, followed by the hybrid fusion model with precision of 0.66. Our results indicate that multimodal machine learning is successful in detecting manholes.

Contents

1 Introduction

Various types of road damage may occur due external factors such as weather and traffic load. For example, freezing and thawing of asphalt causes cracks. Repeated traffic loads causes asphalt to fatigue and rutting occurs, which is when the pavement surface is deformed along the wheel paths. Poor road surface condition causes discomfort for drivers and more importantly, imposes safety risks. In addition, roads play a vital part in the economic infrastructure of a country, by providing access to education, health and employment services. Adequate routine maintenance prevent major repairs and extend the life of the road, which also safes costs for the maintainer. Conventionally, visual and manual methods are adopted to assess the state of roads. However, they are generally subjective, tedious, labor-intensive, and time consuming.

To address these difficulties, automated detection methods have been developed by various private companies and researches. Most commonly researched are image based methods to automatically identify damages [1], [2]. Images can be complemented by depth information through infrared sensors or LIDAR [3], which also mitigates image specific drawbacks such as lightning and shadows. More recently researchers use low-cost sensors such as smartphones to automatically detect road damages [4]–[6]. Accelerometers, which measure vibrations, have also been used to assess the pavement quality [7]–[9].

In the Netherlands, public infrastructure maintainers (i.e., the state, provinces, municipalities, and water authorities) have the legal responsibility for maintaining their assets. Maintaining is defined by law as "ensuring that all roads within the area are in good condition" [53]. Road maintainers are obliged to maintain facilities regularly and in a sustainable way [54]. In order to give some guidance on how to assess the state of roads, standards are made by the national knowledge platform "CROW". Among its activities, CROW prescribes road maintainers how to perform road inspections [55] to help maintainers in quality-driven asset management.

Road maintenance is a big expenditure of the state's budget, it is annually around 2.5 - 3.5 billion EUR [56]. Depending on the type of the road, a different type of government is responsible for the maintenance. For instance, highways (indicated with A) are maintained by the state, provincial roads (indicated with N) by the provinces, and local roads (indicated with street names) by municipalities. Public bodies are legally obliged to report "maintenance capital goods" in their annual budgets [57]. Within the report they have to state the policy framework for maintenance and the financial implications. Road maintenance is planned through a multi-year plan, which requires clear insights in current road conditions.

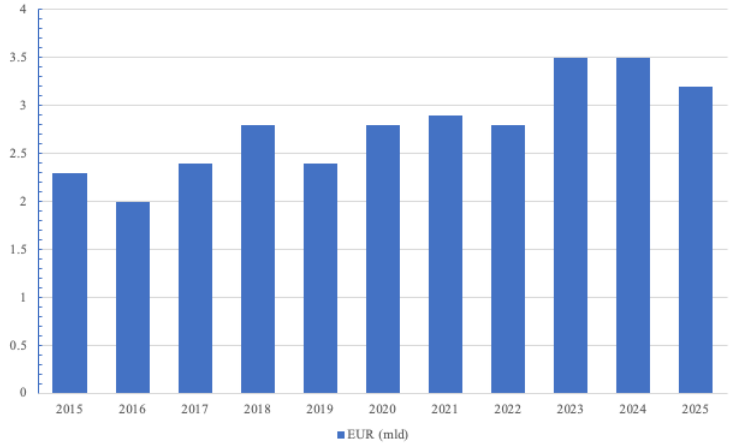


Figure 1: Budget for maintenance of public roads [56]. Budget is in billion EUR.

Road maintenance is performed according to an annual cyclical process. The road owner or maintainer (i.e., state, province or municipality) initiates a tender for inspection. After which, an inspection company inspects the road with specialized vehicles. This vehicle contains multiple cameras to record the road surface, infrared sensor to measure the evenness, and other types of sensors. The recorded video data is manually inspected by engineers according the prescribed CROW method [55]. The inspection company delivers a report with recommendations to the road owner indicating which roads needs maintenance. On this recommendation, the road owner initiates another tender to perform the actual maintenance. Interestingly to note is that companies that can perform inspections, often also can perform the construction.

The condition of the road is measured through various quantifiable data sources. These sources are manually inspected according to CROW method to assess the state of the road. Below is a list given of data sources which are often used within the Netherlands for road inspection:

- Visual inspection: video data is manually annotated to mark damages.
- ARAN measurements: laser sensor which measures distance between the road and vehicle across the pavement, and can be used to measure the surface unevenness or rutting.
- Ground penetrating radar: geophysical method which uses radar pulses to get an image of the subsurface by measuring the reflected signals to assess the quality of the asphalt and deeper layers, detect sinkholes and objects below the surface.
- Falling weight deflector: similar to ground penetrating radar, but using radar pulses, a weight is dropped and reflections are measured. This method is more commonly used in civil engineering.
- Skid resistance: describes the force when a locked tire (i.e., a wheel that is prevented from rotating) slides along the surface. Measurements are performed by measuring the friction on the locked wheel to detect if the road is too slippery leading to aquaplaning.
- Static data: besides actual measurements, there is also static data on roads such as:
  - Materials passport describing the different layers of the road. Contains information when and how a layer was constructed and which material was used.
  - Historical maintenance reports.
  - Future date of maintenance (i.e., multi-year plan).
  - Road intensity (i.e., how many vehicles travel the road).

1.1 AssetWorx

Keeping track of these various sources of data is cumbersome. Although many road owners manage this data manually, there are some tools to support their workflow. These software tools are known as “asset management software” or “management systems”. One of the available tools on market is Predictive Road Maintenance (PRM), a platform for data driven asset management (see figure 2) developed by AssetWorx[1], where the use cases of this thesis is derived. PRM was initially developed during an incubator program known as “Startup in Residence” funded by the Overijssel province [58]. The goal of AssetWorx is to create an uniform data platform for asset management. The idea is to be a industry disrupter by providing data driven insights concerning road quality, maintenance and planning as an independent organization.

1.2 Relevance

Although there are other asset management tools in the market, they typically only incorporate one or two data sources, mainly visual inspections. Instead, PRM is designed as a big data platform and operates on different types of data. As described beforehand, currently in Netherlands, all road assessment rely on visual inspections based on the CROW systematic [55]. This procedure is intensive and largely relies on experience and inspectors’ and maintainers’ intuition. The cycle of tendering → inspection → tendering →

[1] Pronounced as Asset Works

Figure 2: Screenshots of Predictive Road Maintenance (PRM) software.

maintenance, typically takes one year. However, when there is a damage in the pavement, the severity of that damage increase as it remains unrepaired. For instance, a pothole starts relatively small but over time, as more traffic passes, it can become a significant damage. Repairing a small pothole is easy and quick, but as the damages increases, so does the financial costs of repairing the road. Therefore, timely monitoring of road damages saves costs for the maintainer while increasing driving comfort and safety.

AssetWorx aims to systematically and continuously collect data of all roads instead of the conventional annual cycle of inspection and maintenance. The collected data is then processed to automatically determine the state of the road, which is the goal of this thesis. In other words, we aim at automatically detect road surface damages that none of the existing management tools on the market can identify, although some construction companies tried to pursue this goal [59]. Reaching this goal will provide a large business value for AssetWorx and for the community by reducing the cost of road maintenance.

Although automated defect detection have societal and business value, it is scientifically not something new. However, as found from literature survey, existing research only utilizes a single source of data (unimodal) to detect damages. For this research, we are collecting various types of data: camera, accelerometer, and GPS. Multimodal fusion aims to integrate data of different sources and types in a unified representation [10]. The idea is that by fusion richer information is provided than a single modality can provide. For example,

damages may be detected with images, but it is difficult to assess the severity of that damage due lack of depth information. By fusing vibration data from the accelerometer, an additional dimension is added which could describe the severity of a damage. Within existing literature on road defects detection, multimodal fusion has not been applied. Fusion of accelerometer and visual data is largely researched in the context of odometry / robot navigation.

However, fusion of visual and accelerometer data proposes an interesting challenge in this context. The camera is facing towards the road, and therefore sees an object in advance at timestamp t_{visual}. However, at some later point the vibrations are measured at timestamp $t_{accelerometer}$. Between these sensors there is some delay τ. This delay is composed of two additive delays: $\tau = \tau_{capture} + \tau_{detection}$. $\tau_{capture}$ refers to the delay between capturing visual and accelerometer data, it is expected that there is a very small delay. $\tau_{detection}$ refers to the earlier mentioned problem that an object is detected in advance, before the accelerometer measures that event. The latter delay is variable and depends on the angle of the camera, and the travelling speed.

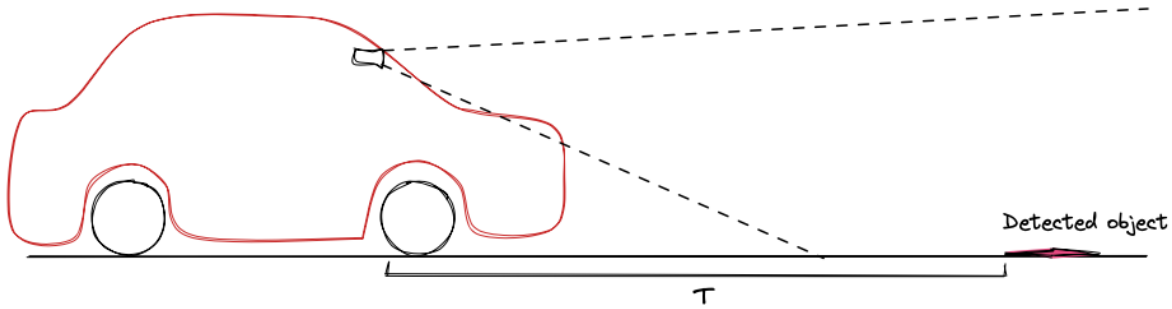


Figure 3: Illustration of the setup of collecting data.

1.3 Research Questions

During this research, data is continuously collected from driving on roads. In particular, it is collected by a smartphone with a customized app, which collects GPS, accelerometer, and video data that need to be combined. Visual data can only be recorded in decent conditions i.e., it depends on weather and lighting. Complementary, vibrations can always be measured regardless of external conditions. At the same time, interpreting vibration data is a more difficult task for humans. Combining both data sources helps to interpret the data. With these points in mind, this thesis aims to answer the following research question:

- **RQ: To what extent can visual object detection and accelerometer data be combined in a multi-modal machine learning pipeline to correctly classify road surface anomalies?**

To answer this question, we divided the question into smaller parts. From quick literature survey, it is known that research exists on classifying road surface defects on single source of data. As the aim is to combine visual- and accelerometer data, it makes sense to first research what is known for each respective source in the field of road surface defect detection. Detecting defects using visual data is basically object detection: localization and classification of objects (i.e., damages) in an image. The first two sub research questions are:

- *SQ1: What is the state of art in machine learning pipelines using object detection to classify road surface anomalies?*
- *SQ2: What is the state of art in machine learning pipelines using accelerometer data to classify road surface anomalies?*

The final answer to the RQ is evaluated based on the empirical results. When we observe that the classification accuracy of the pipeline with combined data is sufficient, we can conclude that combining both sources was successful.

1.4 Contributions

This thesis provides interesting contributions, both scientifically from the performed research, and business value from the developed application. First of all, this is the first research to combine both visual and accelerometer data to detect road surface anomalies. The effect of combining sources is rigorously evaluated by comparing the difference in accuracy between combined pipeline and of the respective unimodal pipelines. When the combined pipelines perform adequately we expect a higher performance than when a single data source is used. Additionally, it allows to perform classifications when only one source of data is available.

Secondly, current asset management tools only store and organize a single source of data. When the developed model is incorporated into the PRM software, AssetWorx has a competitive advantage. Additionally, with the developed model the maintenance cycle becomes data driven and allows for quicker iterations. In turn, this means that maintainers can act quickly on damages and prevent large costs.

1.5 Remainder

The remainder of this thesis is structured as follows. In section 2 we describe the related work. What currently is known on detecting road surface defects using machine learning. It lays the foundation to answer the two sub research questions. Additionally, it introduces the field of multimodal machine learning. Next, section 3 provides background information on the used techniques. Section 4 describes the method of collecting the data, and preprocessing steps. In section 5 we describe the various machine learning models that are developed. Subsequently the results of the various models are presented in section 6. Next in section 7 we discuss the results and answer the various research questions. Finally, section 8 provides a summary of the work and concludes our research.

2 Related Work

The aim of this section is to describe the existing work that has been done on multimodal machine learning to classify road defects. However, to my knowledge, this has never been researched yet. What we did find is research using a single source of data in machine learning pipelines. This section will therefore review what is known in automatic road defects classification. Finally, we look briefly at current literature on multimodal machine learning in general.

From our review we discover that there is whole research field around data driven road defects classification. Motivations for research is that the conventional way of assessing the road quality is perceived as: expensive, labor-intensive, and requires (subjective) domain expertise. As mentioned earlier, there is already a lot of types of data collected to assess the road quality. This is typically done with a specialized vehicle (see 4). As this vehicle is expensive, it is difficult to apply on large scale. Researchers therefore focus on alternative forms to collect the data with cheaper methods. Smartphones are for instance often used as they already contain a camera and accelerometer. In literature identified we the following fields researching road quality based on the used data: “visual surface defects” uses camera based sensors, “sensor surface defects” and “roughness evaluation” uses the accelerometer, and “noise labelling” uses an external microphone.

Figure 4: Automatic Road Analyzer (ARAN) is specialized vehicle to gather data about the road surface [9].

2.1 Visual Surface Defects

Visual surface defects are based on visual techniques to automatically classify damages. Especially more recently smartphones are used. However, researchers have also used dashcams, and Microsoft Kinect. Kinect is equipped with a RGB camera, and an infrared camera to measure depth. In Jahanshahi *et al.* (2012) [1] the authors uses a Kinect sensor to collect image and depth data. Recording of the road has been performed from a top-faced perspective. The authors classify cracks, potholes and patches with accuracy scores of respectively 78%, 92% and 90%. By including depth information, the problem of detecting defects is relatively trivial. Classifying defects is done by checking if the depth is greater than some threshold.

In 2016, the first model was published using machine learning to detect road cracks by Zhang *et al.* (2016) [2]. The data is collected by smartphones from a top-faced perspective of known defects. Their research objective is to train a deep convolutional neural network to classify if a specific image patch is a damage or not. Related to this work is Zhang *et al.* (2017) [3], the authors use 3D pavement images to classify each pixel in a patch individually as crack or no crack. Their objective is focused on acquiring pixel-level accuracy. This is interesting as it describes the severity of the damage (i.e., the size and location). Both methods are similar in collecting and training a deep convolutional neural network. However, their models are specifically trained on small patches, omitting the actual global context (i.e., the actual road). This means that their models wouldn’t generalize well and are hard to deploy in the real world.

One of the earlier methods that collects data in a realistic setting is that of Chatterjee *et al.* (2018) [4]. The authors perform road crack detection on cycling roads in Germany. They use a smartphone attached to a bicycle, and record the road surface from a front-faced perspective. The data is then pre-processed using various computer based algorithms. Subsequently, the extracted features are used in different machine learning models to evaluate performance. Although their method works well, it is not a complete end-to-end machine learning pipeline.

The first complete pipeline is performed in Maeda *et al.* (2018) [5]. Data was collected from a smartphone in a typical holder attached to the windshield. The authors extensively collect data about Japanese roads. Contributions of their research include their results, but additionally contribute the collected data, and a pre-trained model. The dataset is known as Road Damage Dataset by Maeda *et al.* (2018) [60], and receives updates the following years. This is also the first research to include multiple types of damages, i.e., fading road markings, and distinction between types of cracks. Instead on focusing on specific image patches, the authors frame the problem as object detection.

The research is extended in Arya *et al.* (2020) [11], where the authors update the dataset by collecting data from Czech Republic and India. The researchers use transfer learning to evaluate the generalization of the developed model in different countries. They conclude that it is difficult to achieve decent performance when using the model in a different country. Although no explicit reasons are given, this can be expected as different countries build their infrastructure in different ways. However, when the model is transfer learned on images from that respective country, the performance does increase more rapidly, and the yielded model is more generalizable.

To improve performance, the authors of Maeda *et al.* (2020) [6] use a Generative Adverserial Network (GAN) to augment the dataset with more data. This also improves generalizability. With a GAN two neural networks are trained and compete with each other. One model, the generator, tries to generate samples as if they came from the original distribution set. Another model, the discriminator, receives either a real image or a generated sample, and tries to classify if a given sample is real or fake. Overtime, the generator mimics the original distribution, and is capable of generating more samples. With more training data, the model performs better as it is less likely to overfit, and generalizes better. The model is retrained on this augmented dataset, and the authors report a 5% increase in performance.

Another effort to improve the performance of the model is that a public competition was held Arya2020-competition. The aim of the competition was to push the state-of-the-art of detecting road surface defects forward. Current best performance for a single class was F1 score of 0.40. The winner of the competition achieved an average (for all classes) F1 score of 0.67 [61]. The earlier models [5], [6], [11] all used Single Shot Detector to detect the damages. Whereas, the winning model was trained on YOLOv5 [62]. This model is currently regarded as state-of-the-art in object detection. How object detection works is later described in section 3.1.

2.2 Sensor Surface Defects

Sensor surface defects use non-visual data sources to classify defects. Most commonly used is the accelerometer. This is a sensor which measures the change in velocity over time. Typically an accelerometer measures in three directions: X, Y and Z. When the vehicle drives over a damage, e.g., a pothole, the car vibrates and these vibrations are measured with an accelerometer. The output is in G-forces, where $1G = 9.81m/s^2$. In stationary position, the accelerometer always measures 1G on the vertical axis due gravitational pull of the earth.

Detecting potholes using accelerometer of smartphone is often researched. For instance are the works of Wu *et al.* (2020) [12] and Basavaraju *et al.* (2019) [13]. Both papers use machine learning to detect potholes using smartphones. Interestingly, both works follow similar methodology, consisting of four stages: (1) data acquisition, (2) data processing, (3) feature extraction, and (4) classification. The data is processed into sliding windows. On these windows various signal processing operations are performed, and features are extracted. Detection of damages is subsequently done with different machine learners (e.g., Random Forests, Support Vector Machines).

During the data processing stage, both papers perform the following operations: resampling, reorientation, and filtering. The key data processing operations is the reorientation (or realignment) of the accelerometer with respect to the vehicle. In order to use the data, the coordinate systems of the accelerometer and the vehicle must coincide. For instance, when the vehicle experiences a vertical force, we want that respective force also measured on the vertical axis of the accelerometer. In practice is it is unlikely that the accelerometer in the smartphone is perfectly aligned with that of the car. This problem is also applicable in this work, and the process of reorientation will be discussed in depth later in section 4.4.2.

There are a few differences in the research of Basavaraju *et al.* (2019) [13]. First of all, it performs multi-class detection between cracks, potholes, and smooth roads. Another key difference is that the authors of Basavaraju *et al.* (2019) [13] also evaluate an end-to-end pipeline with a neural network. In that experiment, they skip the feature extraction but directly learn from the preprocessed signal. As the authors expected, the performance deteriorated slightly. The main advantage is the time saved on feature extraction. Given enough data, the authors expect that the neural network can perform just as well.

2.3 Road Roughness (IRI)

Besides detecting surface defects, accelerometer is also used to estimate the "road roughness". Which is defined as "the deviation of the surface from the true planar surface". This deviation affects vehicle dynamics and ride quality. Two methods to classify the roughness of a profile are the "International Roughness Index" (IRI) [14] and the International Standards Organisation (ISO) [15] classification.

The conventional method to measure the road's profile is done with an inertial profiler. This is a device which scans the surface of the pavement with lasers to measure the distance between the road and the sensor. This sensor is also equipped on the specialized Automatic Road Analyzer (ARAN) vehicle (see figure 4). From review, we see that there is a large amount of research focused on estimating IRI using an accelerometer found in smartphones [7]–[9], [16]. All researchers followed a similar approach. The accelerations were collected with a smartphone. From the collected data, only data from the vertical axis was extracted. This was subsequently passed through a proprietary tool to calculate the IRI profile. The resulting profile was compared with that of the ground truth, and all research found that smartphones can accurately measure IRI in most cases.

2.4 Noise Labelling

The road quality influences the noise and vibrations emission caused by the interaction between tires of the vehicle and the road. Within the EU, member states are obliged to publish noise maps for their infrastructure every five year [63]. These emissions can be recorded with specialized equipment, e.g., a Close Proximity Trailer (CPX). In Van Hauwermeiren *et al.* (2019) [17], the authors try to replicate the results of CPX measurements by using a sensor box containing a microphone. The microphone is located in the trunk of the car. The authors were able to accurately estimate the road texture. However, this experiment was performed under strict conditions by eliminating background noises (e.g., the radio was turned off, and no talking). The authors continued their research in Van Hauwermeiren *et al.* (2021) [18], where they use multiple vehicles and non-standard driving conditions. They found that below 1600 Hz, that the gathered results were accurate enough to replace the CPX measurement.

2.5 Combination of Visual and Accelerometer

As previously mentioned there is no current work on combining visual and accelerometer data in a complete end-to-end machine learning pipeline. However, in the paper of Lekshmipathy *et al.* (2020) [19], the researchers compare both methods. Accelerometer data is collected with an smartphone attached to the windshield. Without further processing, the data was fed to a neural network and classified either as: no defect, pothole, patch, crack, or bump. The model was able to distinct the different types of damages with

Figure 5: Close Proximity (CPX) trailer measures the sounds emitted by reference tyre [64].

an accuracy of 80 %. For the visual method, the authors collected image data from a top down view perspective. After applying various image processing operations, the authors use a threshold based method to classify visual surface defects. The resulting model slightly outperforms the vibration based method with accuracy of 84 %. The authors argue that for continuous monitoring the state vibration based method is sufficient, and resort to the visual method when there is need for higher accuracy.

Another research that uses both data sources is that of Lee *et al.* (2021) [20]. Again, the authors don't use both sources simultaneously in an end-to-end pipeline. However, this research is still interesting. The researches use a smartphone to collect both sources of data. This time, the image data is recorded from a front-faced perspective. The researches created a visual model to detect road anomalies, and incorporated it into their collection app. When the app detected an anomaly, the accelerometer data is recorded for three seconds. From this data, they make a comparison between the different type of damages, and the response on the accelerometer. They argue that it is hard with only visual data to measure the severity of an anomaly. By including the accelerometer data classifying the severity becomes easier.

2.6 Multimodal Machine Learning

We experience the world as multimodal: we see objects, feel vibrations and hear sounds. Modality refers to the way in which something happens or is experienced. A research problem is characterized as multimodal when it includes multiple of such modalities [10]. Commonly referred example of an everyday multimodal experience is the McGurk effect [21]. This is the perception between hearing and vision in speech: when we hear the syllable /ba/ while watching the lips of someone saying /ga/, we perceive it as /da/.

In the work by Baltrusaitis *et al.* (2017) [10] multimodal fusion is described as integrating information from multiple modalities with the goal of predicting an outcome. There are three main benefits it can provide. First, having access to multiple modalities that observe the same event allow for better predictions. Second, multiple modalities might complement each other. Third, a multimodal system can still operate when one of the modalities is missing.

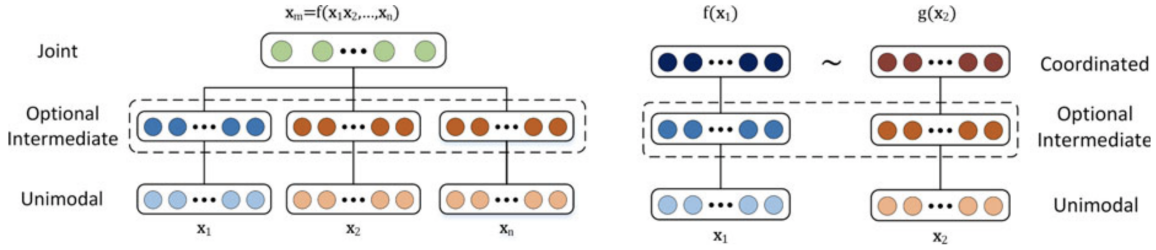
Multimodal research has a long history from audio-visual speech recognition to more recent interest due deep learning [22]. It has been proven that multimodal learning algorithms performs really well on various tasks, such as (audio-visual) speech recognition [23], image sentence matching [24] and RGB-D object recognition [25]–[27].

Baltrusaitis *et al.* (2017) [10] identified five challenges dealing with multimodal machine learning:

1. **Representation**: how to represent and summarize multimodal data to exploit the complementary and redundancy of multiple modalities. Distinction is made between *joint representation* - which combines unimodal signals in the same space, and *coordinated representation* - which processes unimodal signals separately, but enforces similarity constraints. See figure 6 below for an illustration.

2. **Translation**: how to translate data from one modality to another. For example, given an image the task is to give a caption to describe the image.
3. **Alignment**: how to identify direct relations between (sub)elements from multiple modalities. For example, given an image and a caption, find the area in the image describing the caption.
4. **Fusion**: how and when to fuse / join information from multiple modalities. Historically the original topics of multimodal machine learning, with emphasize given on early-, hybrid- and late-fusion.
5. **Co-learning**: how to transfer knowledge between modalities. For example by exploiting knowledge of a rich modality, to aid modelling of a less describing modality.

There are various methods to implement multimodal machine learning. One of these methods is using neural networks. The inputs of the data sources can directly be used as inputs to the neural network. One of the advantages of using neural networks is that it is easy to train an end-to-end model [10], [22]. However, the disadvantage is their lack of interpretability.



(a) Joint representation

(b) Coordinated representations

Figure 6: Structure of joint and coordinated representation [10]

As mentioned before there hasn't been any research performed on multimodal fusion on road defects. In the broader sense, multimodal fusion has been applied to detect damages. Examples of which are: combining audio-visual data to detect conveyor belt damages [28], gearbox fault detection based on vibrations and acoustic signals [29]. For more examples of applied data fusion in manufacturing industry, see [30].

Further research shows that visual and accelerometer (IMU) data have been combined in odometry. With odometry the task is to estimate the location over time. For instance, calculating the current position, based on the previously known position. This is commonly applied in intelligent robots. Different motion sensors are fused to overcome "dead-reckoning", the loss of signal. Which can happen when the GPS data is lost or unstable [31], [32].

One of the key challenges for this thesis is to synchronize the data sources. As mentioned earlier, this problem occurs because the camera sees an object on the road where the car will be driving after some delay. See also figure 7. Often in data fusion literature, modalities are misaligned due varying sampling rates, clock synchronization, or transmission delay between sensors. There are various known approaches to fix these issues. In general, the solution is designing a robust synchronization protocol to provide common notion of time. Existing methods often rely on a similarity measure or a trigger event between the different sources to fix this issue. Sensor synchronization is studied extensively in domain of sensor networks. When there is a similarity measure between the signals, the time delay between the signals can be calculated [33].

2.7 Conclusion

Automatic classification of road defects have been extensively researched. Main topic of interest is the usage of low-cost devices such as smartphones to collect the data. From the literature we find that only the following types of defects are classified: cracks, patches, holes and faded line markings. Other types of damages that are recognized by the CROW [55], but haven't been researched are raveling, rutting and skewed signs. Although this work doesn't try to classify those damages either, it illustrates an interesting gap in this field.

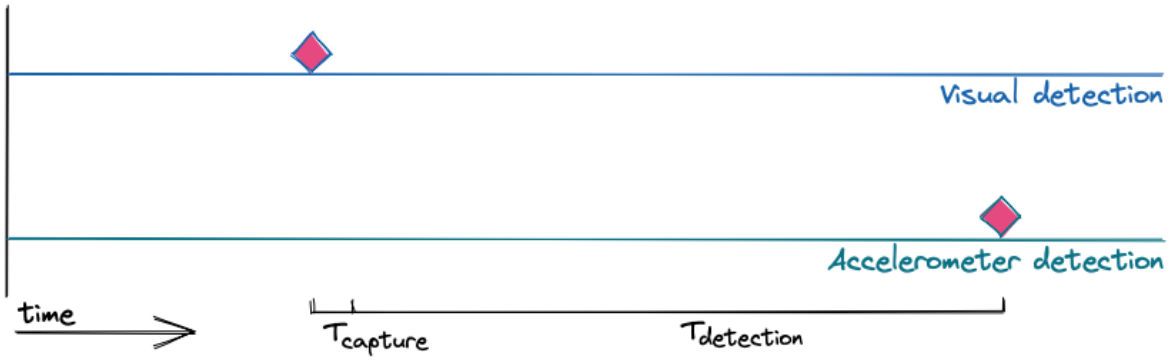


Figure 7: Illustration of the synchronization problem: there is a delay between detecting the object and driving over that object.

From our review, we conclude that there are some interesting papers to work from. First of all, we have the public data and models of the Road Damage Detector [34]. The authors frame the problem of detecting surface defects as object detection. This makes sense as damages in a image are basically objects. The best performing model uses the state-of-art model YOLOv5 [61], [62]. During this work we'll also use YOLO to detect damages from visual data. As the authors describe in Arya *et al.* (2020) [11], it unexpected that the pretrained model yields good results in a different country. By transfer learning the model on roads collected in the Netherlands, we improve the generalizability of the model.

Secondly, the research on detecting potholes using accelerometers uses a common methodology [12], [13]. We'll also use this methodology, and apply the same processing steps of resampling, reorientation, and filtering. The implementation of these steps will be described later.

Finally, we see that there hasn't been any research on multimodal machine learning in road defects detection. This makes this work novel, and an interesting contribution to the literature. From literature we learn that using neural networks is an easy method to implement multimodal machine learning [10].

3 Background

This section describes the used techniques in this study. First we describe multimodal machine learning, and their key challenges. Followed by background information on object detection, and the evolution of models until the current state-of-art YOLOv5. Finally, we explain some key signal processing operations that are often applied on accelerometer data.

3.1 Object Detection

State of the art visual road defects detection rely on object detection. Object detection is the combination of classifying and locating the object. The output of the model is the detected class and the bounding box where that object is detected. Within a single image there can be one or multiple defects, sometimes from different types. With object detection it is possible to locate all these different instances.

Object detection is well researched topic, and applied in all sorts of domains. The current state of the art model is known as YOLOv5 [62], and has evolved from earlier models. The following sections describe the evolution of object detection. Simultaneously there has been other models but for relevance we limit this section to evolution of YOLO.

3.1.1 R-CNN

The concept of image classification can be easily extended to perform object detection. A naive approach to solve the problem of localizing objects is to slide a window over the image, and for each window perform image classification. The major problem with this approach is that objects have different locations, sizes and aspect ratios. For this approach to work, an infinite possibility of regions need to be computed.

Girshick *et al.* (2013) [35] solves this problem by proposing a model called Regions with CNN (R-CNN). Their approach is to extract 2000 regions using a "region proposal algorithm". In the paper they use selective search [36] - an algorithm which extracts regions from image by grouping pixels on their similarity. For each region a convolutional neural network is used to extract features, and the object classifications are made by a SVM classifier.

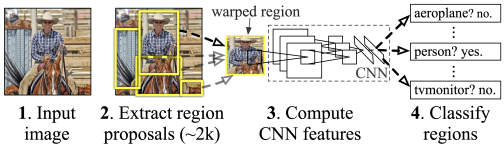


Figure 8: R-CNN: Object detection overview [35].

3.1.2 Fast R-CNN

The initial R-CNN model worked, but it was quite slow. This slowdown was because each of the 2000 extracted regions, was passed through the CNN. The same author continued his research to solve this problem. His new model was called Fast R-CNN [37]. Instead of passing each region through a CNN, the full image is passed through. The generated regions are then projected on the feature map generated by the CNN. The resulting projection is known as "Region of Interest" (ROI). Additionally, with Fast R-CNN, the SVM classifier is replaced with fully connected layers. The original R-CNN takes about 50 seconds per image to detect objects, whereas the Fast version was able to process the image in 2 seconds.

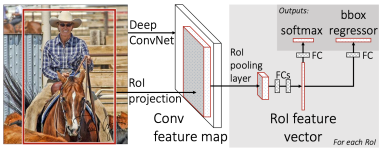


Figure 9: Fast R-CNN: Object detection overview [37].

3.1.3 Faster R-CNN

Although Fast R-CNN is significantly quicker to detect objects, it still takes about 2 seconds to localize the object. The bottleneck now was the region proposal algorithm, which is implemented on the CPU. Improvements over the used selective search were researched. However, Ren *et al.* (2015) [38] argued that it would be more interesting to develop a model which combines region proposal and object detection. Their model was called Faster R-CNN and is able to process an image in 200 milliseconds.

Faster R-CNN consists of two modules. The first module is a deep fully convolutional network which proposes regions, known as Region Proposal Network (RPN). The RPN takes an image as input and outputs a set of object proposals, each with an *objectness score* (measurement if something is an object or background). This output is fed into the second module, which is the Fast R-CNN object detector.

The unified model has several advantages over the Fast R-CNN model. Because the region proposal are generated within the network, the model can be trained end-to-end. Making deployment easier, requires less resources, and performs much faster. Additionally, this allows the region proposals to be tuned according to the detection task.

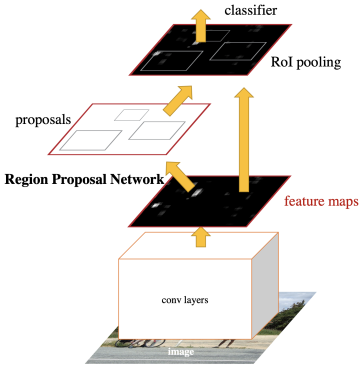


Figure 10: Faster R-CNN: Object detection overview [38].

3.1.4 YOLO

The previous detectors all use regions to localize an object. The models look at parts of the image with high probability of containing an object. YOLO has a different architecture by looking at the complete image. A single neural network predicts bounding boxes and class probabilities in one evaluation. Using the system, "you only look once" (YOLO) at an image to see what objects it contains Redmon *et al.* (2016) [39].

YOLO as several benefits. First of all is that YOLO is extremely fast. Due the new architecture, it doesn't require a complex pipeline, and can be trained end-to-end. Secondly, unlike previous models, YOLO reasons globally about the image. Most mistakes from earlier models are due mistaking background for an object. YOLO makes less than half the number of background errors compared to Fast R-CNN. Finally, YOLO is capable to learn generalizable representations of objects. For instance, when YOLO is trained on natural images and tested on art-work, YOLO outperforms top detectors such as R-CNN. YOLO does lag behind state-of-the-art detectors in terms of accuracy. Although it can quickly detect objects, its struggles to precisely localize small objects.

YOLO works by taking an image and split it into an SxS grid. Each grid predicts B bounding boxes, and a confidence scores for those boxes. These scores reflect how confident the model is that the box contains an object. The model outputs for each bounding box 5 predictions: the location (x, y), size (w, h) and confidence score. Additionally, each grid cell also predicts C conditional class probabilities. The complete output is then passed through a deep convolution neural network to make detections.

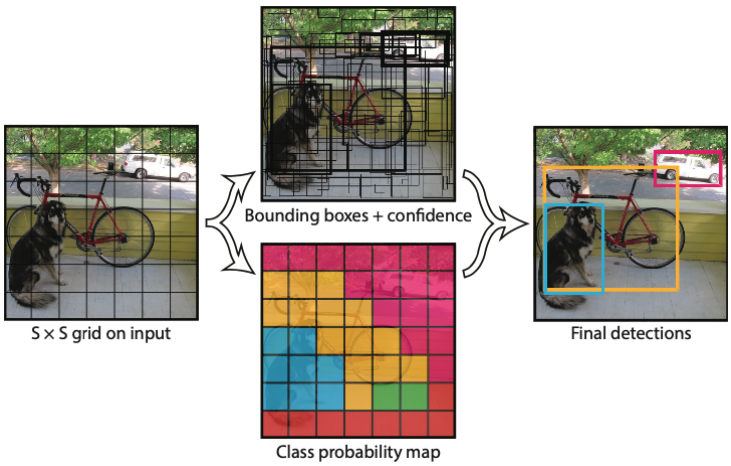


Figure 11: You Ony Look Once (YOLO) detector model [39].

3.1.5 YOLOv2

The authors of YOLO continued their work and introduce two new detectors YOLOv2 and YOLO9000 [40]. They argue that current object detection datasets are limited (contain about hundred thousand images) compared to datasets used for classification and tagging (contain about millions of images with thousands of categories). They propose a novel method to harness large amount classification data and use it to expand current detection systems. Using this method they develop YOLO9000, a real-time object detector that can detect over 9000 categories. To do so, they first improve the base YOLO detection system to produce YOLOv2.

YOLO suffers from various shortcomings relative to state-of-the-art detection systems. YOLO makes significantly number of localization errors compared to Fast R-CNN. In deep-learning, better performance often comes from training larger networks or ensambling networks together. With YOLO, the authors opt to simplify the network and introduce several architectural changes. In short, most improvements stem by implementing state-of-the-art deep learning techniques. As the details are not relevant for this work, please refer for details to [40]. Below is shown an comparison of accuracy and speed between various detectors to demonstrate the effects.

3.1.6 YOLOv3

YOLOv3 is another improved version of YOLO by the same authors [41]. It uses a new base model which is a bit slower but more accurate. YOLOv3 predicts boxes at 3 different scales, based on the idea of Feature Pyramid Network [42]. This allows the model to learn objects at different scales. Remedying the problem with detection of small objects. Additionally, YOLOv3 performs multi-label classification instead of earlier used softmax. This makes it possible to label the same object with multiple labels, e.g., an object can be both a person and a woman.

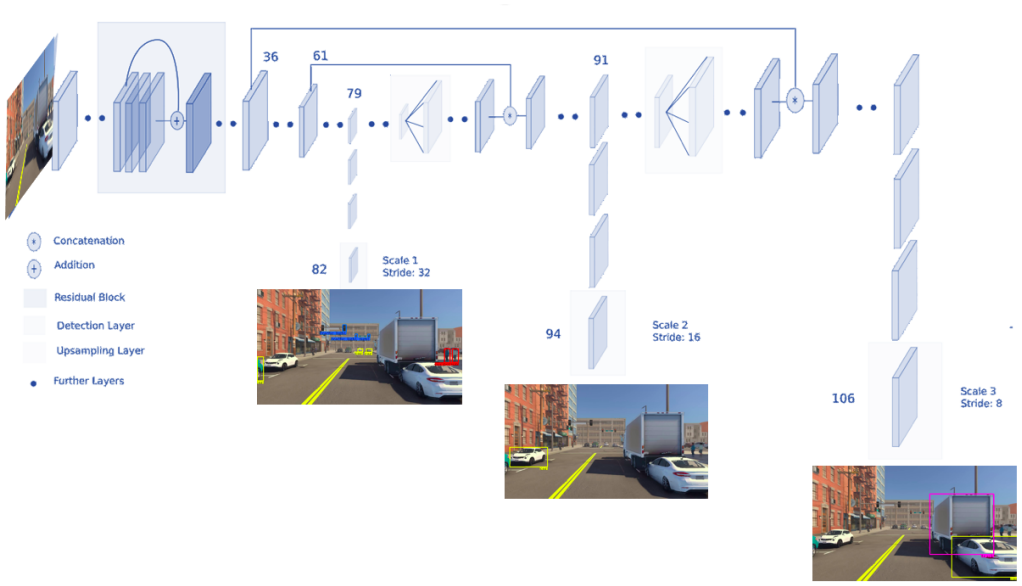


Figure 12: YOLOv3 network architecture [65]. Note that based on the output scale, the model detects objects of different sizes.

3.1.7 YOLOv4

YOLOv4 is the successor of YOLOv3. Unlike earlier models, YOLOv4 is developed by other authors Bochkovskiy *et al.* (2020) [43]. In their paper the authors explore various deep learning techniques to improve the performance of YOLO, some deriving from earlier research, as well some novel techniques. The results is an improvement of 10% accuracy and about 12% increase in speed.

One of these novel techniques is named Mosaic. It is a data augmentation method that mixes 4 training images. Thus mixing 4 different contexts. This allows detection of objects outside their normal context. In addition, when applying batch normalization, it calculates statistics from 4 different images. Reducing the need for large mini-batch sizes.

Another novel technique is Self-Adversarial Training (SAT). This is also a data augmentation technique that operates in 2 stages. In the first stage, the network modifies the image instead of the weights. Thereby performing an adversarial attack on itself. In the second stage, the network is trained to detect an object on this modified image.

3.1.8 YOLOv5

Currently, there is no paper released on YOLOv5. At this time, it seems that YOLOv5 is still under active development [62]. Nevertheless, the authors claim it outperforms all current object detectors. In the Road Damage Competition [34], YOLOv5 it was already used in the winning model [61]. In this work YOLOv5 is also used. In figure 13 we can see the neural network architecture.

The authors provide various pretrained configurations of the model. The overall architecture remains the same but each configuration differs in the network size. For instance, the small `YOLOv5s` has 7.2 million parameters and the larger `YOLOv5m` has 21.2 million parameters. Each model is pretrained on the COCO dataset [44]. This is a benchmark dataset used for object detection. It contains thousands of images for 80 different classes.

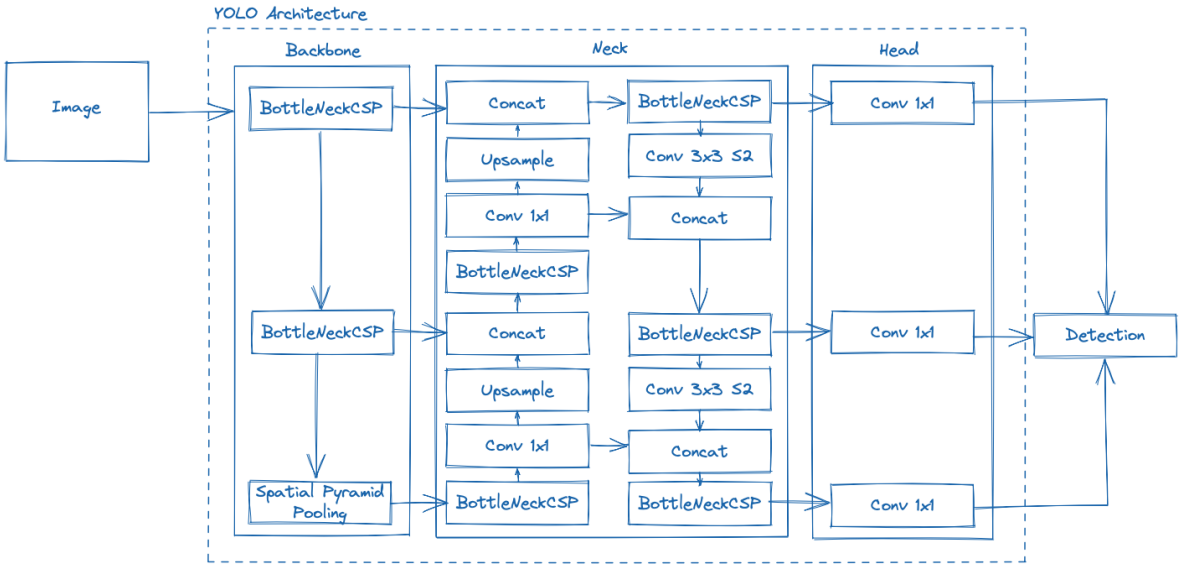


Figure 13: Architecture of the YOLOv5 network. It is derived from the source code [62].

3.2 Fast Fourier Transform

Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain. Fast Fourier Transform (FFT) is an algorithm that computes the periodic components by convoluting sine waves having different frequencies with the signal [45]. Basically, it describes the original signal as combination of many periodic sine waves. The output of the algorithm is known as the frequency spectrum. It tells us all the frequencies that are present and in what proportion. The inverse Fourier Transform constructs a signal given the periodic signals.

The Fourier Transform has many uses in signal processing. For instance, convolution in the time domain is equal to multiplication in the frequency domain. For discrete signals it is almost always faster to implement the convolution from the frequency domain. Finally, in the research of detecting potholes using accelerometer, the frequency spectrum is used as feature extraction [12].

3.3 Butterworth Filter

Accelerometer data is prone to noise. As is common with signal processing, data is first passed through a filter to obtain a clearer signal. Digital filter operates on sampled, discrete-time signal (i.e., time series) to reduce or enhance certain aspects of that signal. One of the most commonly used filters is the Butterworth filter. It is filter which operates in the frequency domain. It is designed to have frequency response that is as flat as possible in the “passband”. The pass-band is the range of frequencies that are allowed to pass i.e., not filtered. For instance, a low-pass filter only allow allows frequencies in the signal below a certain threshold. This threshold in signal processing is known as the cutoff frequency.

Existing research on detecting potholes uses a variety of different filter configurations to clean the data. Some apply a low-pass Butterworth filter [9], whereas other researches use a high-pass Butterworth filter [12], [46]. Due the different usages, we assume that it differs per application and approach it as a possible tuning parameter.

4 Data Collection and Processing

This chapter aims to describe all data preprocessing steps. After these steps, the data is ready for analysis, and machine learning models can be developed. First, we describe data collection, followed by an architectural overview of all data pipelines i.e., how the data is processed, and with what tools. Finally, we describe how each collected data source (i.e., GPS, accelerometer, and visual data) are preprocessed.

4.1 Data Collection

Initially data was collected using a sensor box connected with a webcam. Specifically, an AutoPi [66] was connected to the OBD-2 port of the vehicle. The OBD-2 connector is mandatory in all cars since 2004 in the EU [67]. Through the port, data about the vehicle internals is accessible e.g., a mechanic reading the engine status. Because the AutoPi is connected with that port, it logs various sensors such as RPM, speed, engine temperature etc. It depends on the specific vehicle what kind of data is available. Additionally, the AutoPi is equipped with sensors such as an accelerometer and GPS.

Unfortunately, there are two problems with this collection setup. First, the used webcam was unable to accurately record visual data. A webcam is designed to be used indoors and failed to deal with the variable conditions while driving (e.g., lighting, vibrations). Initially, this problem was solved by recording visual data with an external smartphone. However, there was another issue related to the accelerometer data provided by AutoPi. The sensor could only be sampled at maximum frequency of 12.5 Hz. Some initial experiments demonstrated that this low rate was not sufficient for proper data collection.

The visual data was already collected with a smartphone. Therefore, we decided to also collect accelerometer and GPS data with that same device: an Apple iPhone 12 Mini. The phone was located in a generic phone holder attached to the windshield, see figure 14. All data collection was performed with the same vehicle, a Volkswagen Polo 2012 1.2 TSI. Data collection was done with an open-source app, which was slightly modified and deployed on the smartphone.[2] With this app, the following data is collected:

- Visual data: recorded in 1280 x 720 at 30 frames per second.
- Accelerometer data: sampled at 100 Hz.
- Location data: sampled at 1 Hz and consisting of GPS coordinates, travel heading, and traveling speed.

Figure 14: Setup of collecting data. An Apple iPhone 12 Mini is located in a generic smartphone holder, attached to windshield of a Volkswagen Polo 2012 1.2 TSI.

[2] See [68] for modified fork and [69] for original. Main modification is to keep the smartphone from sleeping while collecting data.

4.2 Data Platform Architecture

The collected data is systematically processed through various data pipelines. All data pipelines combined are holistically presented as a *data platform architecture*. The developed data pipelines are designed to be executed using cloud services. Reason for this choice is twofold. First, some data operations take a long time to be processed. Using cloud services processing of the data is more efficient, i.e., due the availability of highly scalable resources. For instance, multiple data pipelines can run in parallel. Secondly, the developed solution is to be deployed within the existing application PRM (see section 1.1). With this architecture the solution is easily integrated within the PRM software.

Implementation and execution of the data platform uses Google Cloud Platform (GCP). It uses the following services provided by GCP: Cloud Storage, BigQuery - a data warehousing solution, and Kubernetes - a platform for executing container workloads. Orchestration of all data pipelines is done using Prefect [70], a "data workflow automation tool". This tool is designed to orchestrate large scale workflows for data processing. However, it also provides an easy to use library to create and run data pipelines locally.

The solution uses a modern data platform architecture with loosely coupled layers. A data architecture is either modelled based on the used data sources, or on the specific "areas" (or maturity) of the data. In this case, the latter approach is used. Maturity refers to the readiness of consumption of that respective data. Data comes in its raw form into the data platform (landing area), but often needs some transformations (i.e., cleaning, deduplication) to make it ready for consumption (production area). Figure 15 shows an overview of the data platform, which is composed of three major layers: ingestion, storage, and processing.

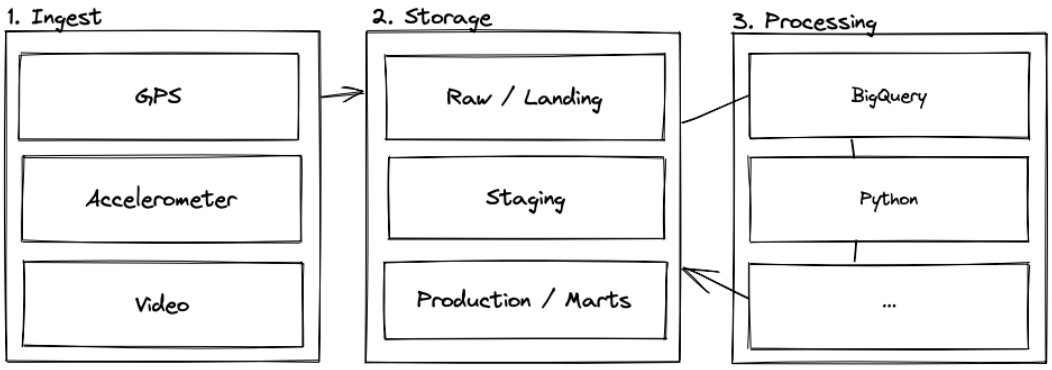


Figure 15: Overview of data platform architecture.

1. Ingestion Layer: it sends the data to the data platform. Data is manually pulled from the smartphone to a laptop. From this laptop the data is uploaded to the cloud platform using automated scripts. Note, to ensure privacy concerns, visual data is first anonymized to remove any personal identifiable data (e.g., cars containing license plates, persons).
2. Storage Layer: data is stored in the data platform in two services: BigQuery - a managed data warehouse solution and Cloud Storage - a generic object store. Within these two services, the data is split in three layers denoting the maturity of the data.
   (a) Raw / landing: it contains data in the form collected by the smartphone.
   (b) Staging: an intermediate layer to preprocess data.
   (c) Production / marts: it contains data ready to be analyzed data and train the models.
3. Processing Layer: it consumes data from the storage layer and processes it. Some processing steps are implemented in SQL and executed in BigQuery; whereas others are implemented in Python (using Prefect) and executed locally or in the Kubernetes cluster.

A complete overview of all data pipelines is given in appendix **??**. The appendix describes the input and output of each data pipeline, and where it is executed (e.g., BigQuery, or Kubernetes). Additionally, a cohesive flowchart is shown. The rest of this chapter describes the data pipelines from a functional perspective.

4.3 GPS Data

GPS data describes the location of the vehicle in coordinates. Specifically, the coordinates are recorded as latitude and longitude (i.e., WGS84 standard [71]). The data is sampled at 1 Hz. GPS data is further processed to compute distinctive trips, speed and heading information of the travelling vehicle.

Figure 16: Figure displaying all processing operations for GPS data.

4.3.1 Determine Trips

The GPS sensor records the location of the vehicle. After ingestion, this data is stored in a raw form on Google Cloud Storage (GCS), as json lines. This is a file format where each line in the file is a single json document. Within this document the location of the vehicle is recorded with the respective time. From Cloud Storage the data is ingested into BigQuery, which detects duplicated trips with the algorithm described below. Each trip has an unique identifier that is used to combine different data sources e.g., finding the accelerometer data for a specific trip.

1. Deduplication: the GPS sensor always records location data. If the car is standing still (e.g. traffic light), the data is still recorded, resulting in duplicate records. The first step is to deduplicate the data. This is done by selecting the first record for each unique location in subsequent time series. See listing 1 for psuedo implementation.
2. Delta calculation: the algorithm computes the time and distance difference for each record.
3. Split trips: trips are detected by splitting the records where the "dwell time" is higher then some threshold. Dwell time is the time period when the vehicle on the same location. From literature, we find that 120 seconds is often used as threshold [47]. See listing 2 for psuedo implementation.
4. Reset trips: for each first record of a trip, the delta time and distance is reset to zero.
5. Cleanup trips: the algorithm keeps only trips with more than five records and at least 500 meters travelled to cleanup invalid data.

```
def deduplicate(source_data: list):
    deduplicated = []
    for x in source_data:
        # For all following records, find first record that
        # has different coordinate.
        x_at = index_of(x)
        for y in source_data[x_at:]:
            if x.coord != y.coord:
                deduplicated.append(x)
                break

    return deduplicated
```

Listing 1: Psuedo implementation of deduplicating GPS coordinates.

```
def split_trips(source_data: list):
    # Period in seconds after which new trips are detected
    dwell_time = 120
    trip_id = 0

    # Iterate through all data points
    previous = source_data[0]
```

```
    for x in source_data[1:]:
        # New trip is detected after period of dwell
        if previous.timestamp - x.timestamp > dwell_time:
            trip_id += 1

        # Appoint trip_id to GPS record
        x.trip_id = trip_id
        previous = x
```

Listing 2: Psuedo implementation of splitting GPS data into trips.

Time	Latitude	Longitude
14:00	**52.3644**	6.46369
14:01	52.3644	6.46369
14:05	**52.3654**	6.46369
14:06	**52.3674**	6.46369
14:07	**52.3684**	6.46369
14:08	52.3684	6.46369

Time	Latitude	Longitude
14:00	52.3644	6.46369
14:05	52.3654	6.46369
14:06	52.3674	6.46369
14:07	52.3684	6.46369

Trip	Time	Latitude	Longitude
1	14:00	52.3644	6.46369
2	14:05	52.3654	6.46369
2	14:06	52.3674	6.46369
2	14:07	52.3684	6.46369

Figure 17: Example of trip deduction algorithm. Left shows original data, center shows after deduplicating based on the location, right shows the resulting trips.

4.3.2 Snap Trips to Roads

GPS data is generally regarded as noisy. The accuracy of the sensor varies over time; thus, some coordinates are aligned with roads, while other samples are outside them (see figure 18). This is an issue when processing data further. In our case, we want to join GPS data with road information (further described below) by snapping GPS coordinates to actual roads. To this aim, we use an external service named “Snap Points to Roads” from Bing Maps [72]. However, we could leverage alternatives as well. For instance Google Maps offers a similar API, but costs more.

Usage of these services is straightforward. For each trip, all coordinates are send to the API. The service responds with a a list of modified coordinates snapped to the roads. The service automatically derives the driving direction based on the timestamps. The driving direction is used to correctly snap the coordinate to the right lane. Thus, if a coordinate is erroneously recorded on the oncoming lane, the service corrects for it.

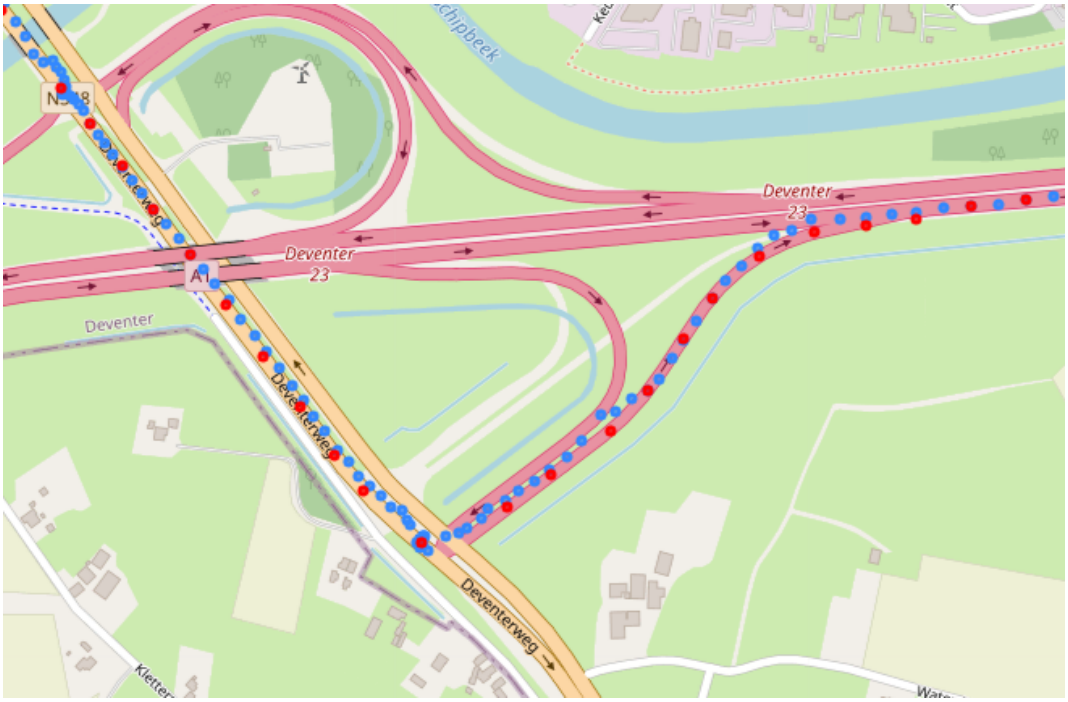


Figure 18: Display of snapping coordinates to roads. The blue dots are the measured GPS coordinates, and the red dots are the inferred coordinates by external service. Note that some of the blue dots are on the wrong lane or beside any roads.

4.3.3 Link Trip Coordinates with Road Sections

Basisregistratie Grootschalige Topografie (BGT) is the registry of "large-scale objects in the public space" in the Netherlands. This registry records all geographic locations of so called large-scale objects. These include roads, buildings, and estate. The BGT is regulated by law and is mandatory for governments to use and maintain while operating in the public space [73]. The BGT registers roads as "road sections". The size of a section varies to a single street bump in a town, to several hundred meters of highway. Road sections are annotated with various types of attributes. In this case, we are interested in the type of road: concrete, asphalt, or brick. Brick roads are common in urban areas in the Netherlands and generate significant vibrations, which might need to be filtered out.

The road snapped coordinates are used to find the respective road section from the BGT. For each GPS record it finds the road section where that coordinate is in contained. See also listing 3 for psuedo implementation.

```
def join_gps_with_bgt(gps_data: list, bgt: list):
    for gps in in gps_data:
        for road_section in bgt:
            if road_section.contains(gps.coordinate):
                gps.road_section = road_section
```

Listing 3: Psuedo implementation of joining GPS data with road sections.

4.4 Accelerometer Data

Accelerometer measures acceleration, the change of velocity over time. In the smartphone the sensor measures acceleration in three axis: longitudinal (X), lateral (Y), and vertical (Z) axes. The output is in G-forces, where $1G = 9.81m/s^2$. In stationary position, the accelerometer always measures 1G on the vertical axis due gravitational pull of the earth. Acceleration data describes the vibrations of the vehicle. However, before the data can be used, it needs to be pre-processed. The most important prepocessing step is the alignment of sensor orientation with the direction of the vehicle. All processing steps are described below.

Figure 19: Figure displaying all processing operations for accelerometer data.

4.4.1 Resampling

Resampling the data ensures a constant sampling frequency. This is necessary for further signal processing operations such as calculating FFT or filtering noise. The effect of resampling is shown in figure 20, which represents the data collected with AutoPI, as the effect is less noticeable with the more consistent data from the smartphone. The top blue graph shows the raw readings from the accelerometer. The bottom red graph shows the resulting data after re-sampling. Note that the interval between readings of the top graph vary, whereas the bottom is uniformly spaced.

Resampling of the data is done by fitting a B-spline curve for all the known samples. From this spline, the data points are uniformly sampled at the expected frequency of 100 Hz. To perform these calculations, the open-source Python library scipy is used [74]. See listing 4 for psuedo implementation.

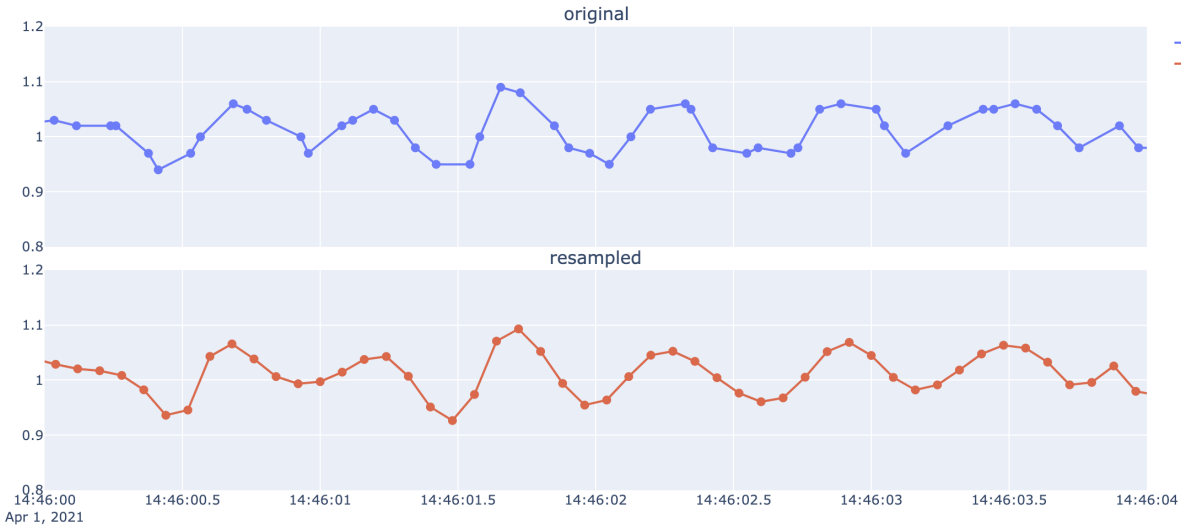


Figure 20: Top graph displays the raw data from the accelerometer. Bottom graph displays the accelerometer data after re-sampling. Note this example is made on data from AutoPi, where the effect is more prominent.

```
from scipy import interpolate

def resample_accelerometer(acc_data):
    # Create uniform timestamp ticks
```

```
    x_min = round(acc_data.timestamp.min().seconds)
    x_max = round(acc_data.timestamp.max().seconds)
    x_resampled = list(x_min, x_max, 1 / 100)

    # For all accelerometer axes
    for var in ['x', 'y', 'z']:
        # Fit a B-spline using scipy
        tck = interpolate.splrep(acc_data['timestamp'], acc_data[var])
        # Sample uniformly from the spline
        resampled = interpolate.splev(x_resampled, tck)

        # Overwrite the data
        acc_data[var] = resampled
```

Listing 4: Psuedo implementation of resampling accelerometer data.

4.4.2 Reorientation

The coordinate system of the accelerometer may not coincide with that of the vehicle. In other words, when the vehicle drives over a bump we expect to see a acceleration in vertical axes. However, when the vehicle is not aligned, the acceleration is shown on a different axes or as a product of axes. There are two causes for this misalignment. First, the orientation of the sensor in the smartphone is not known, making it hard to align it correctly. Secondly, we cannot expect to place the smartphone always equally in the holder. Before any data can be analyzed, the coordinate system of the accelerometer must be aligned with that of the vehicle. Therefore, we develop a method to automatically align the coordinate systems and making it possible to generalize data collection to any vehicle and accelerometer sensor. This process is inspired by Poeze *et al.* (2019) [75] and actually is also applied in Wu *et al.* (2020) [12]. See listings 5 - 7 for pseudo implementation of these processing steps.

It consists of two steps. First, the coordinate system is aligned on the vertical Z-axes using a rotation matrix R_z. Subsequently, the horizontal plane is aligned using a rotation matrix R_x. Calculating these rotation matrices is done by measuring the accelerometer sensor a_s when a known force is exerted on the vehicle, a_v. Using Euler angles we can calculate the rotation matrix to align the force a_s to that of the known force a_v. In the figure 21 the two-step method is visually presented.

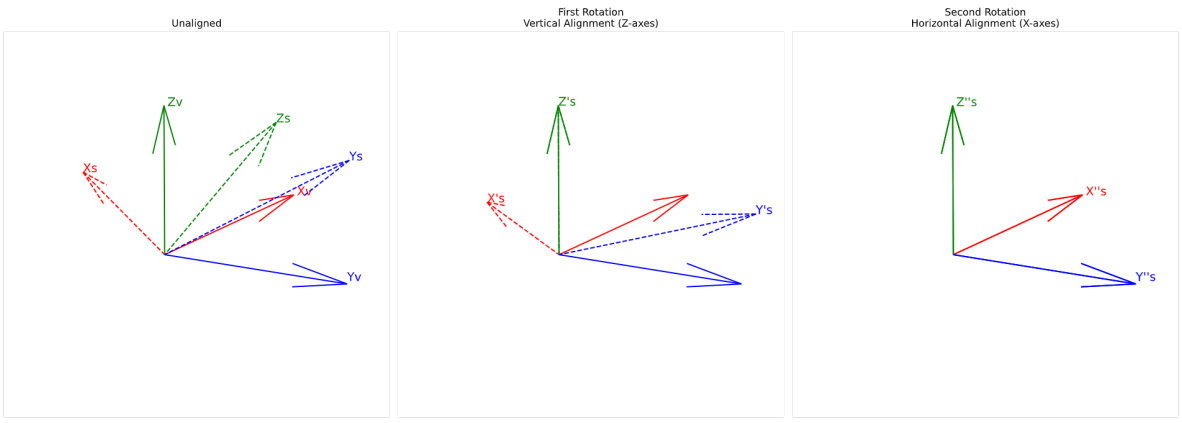


Figure 21: Figure displaying the process of aligning the axes. Zv is the Z-axes for the vehicle, Zs is the Z-axes for the sensor. Left: shows the original, unaligned situation. Middle: shows the situation after aligning the Z-axes. Right: shows situation after aligning the X-axes.

For the first step, aligning the vertical Z-axes, the known force is the gravity of the earth. When the vehicle is stationary, the expected force is $a_v = (0, 0, 1G)$. Stationary points are found by querying the GPS data when the car is not moving. Ideally, only accelerometer data is used when the vehicle is on a level surface. As we do not have access to this information, we rely on taking the average of many samples, as it can be assumed that on average the vehicle is mostly stationary on a flat surfaces. This assumption seems plausible in the Netherlands, where the country is largely flat. The rotation matrix R_z is calculated to align vector a_s onto vector a_v.

The second step is to align the coordinate system in the longitudinal X-axes. Again, we take readings of the accelerometer when a certain force is expected. In this case when the car is braking, the expected acceleration is $a_v = (-x, 0, 1G)$, where x depends on the deceleration of the vehicle. Braking windows are selected based on the derived speed from the GPS coordinates. In this step we calculate a two dimensional rotation matrix R_x to align vector a'_s onto vector a_v in the longitudinal X-axes. a'_s refers to the measured acceleration after applying the first rotation.

$$
\begin{aligned}
\text{Let } v &= a_s \times a_v \\
\text{Let } c &= a_s \cdot a_v \\
\text{Let } skew &= \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix} \\
R_z &= I + skew + skew^2 \frac{1}{1+c}
\end{aligned}
\qquad
\begin{aligned}
\sin\theta &= a'_s \times a_v \\
\cos\theta &= a'_s \cdot a_v \\
R_x &= \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\ \theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

Figure 22: Equations to calculate the rotation matrix. Left, calculates the 3D rotation matrix to align the Z axis. Right, is the second rotation matrix to rotate in the X axis [76].

```
def calculate_mean_stationary_readings(acc_data):
    # Minimum amount of seconds the car is stationary
    stationary_window = 3

    # Query GPS data for given trip
    gps_data = query_gps(acc_data.trip_id)

    # Find stationary windows
    stationary_moments = []
    previous = gps_data[0]
    for gps in gps_data[1:]:
        # Note, after deduplicating each GPS record registers
        # how long that GPS location was measured, i.e., how
        # long the vehicle is at this point.
        if gps.delta_seconds > stationary_window:
            stationary_moments.append(gps.timestamp)

        previous = gps

    # Calculate mean accelerometer reading
    readings = []
    for acc in acc_data:
        for stationary in stationary_moments:
            if  acc.timestamp >= stationary and
                acc.timestamp <= stationary + stationary_window:
                    readings.append(acc)

    return mean(readings)
```

Listing 5: Pseudo implementation of calculating stationary readings.

```python
def calculate_mean_braking_readings(acc_data):
    # Minimum amount of seconds the car is stationary
    stationary_window = 3

    # Query GPS data for given trip
    gps_data = query_gps(acc_data.trip_id)

    # Find consecutive records where the speed is declining
    braking_moments = []
    previous = gps_data[0]

    # First record in consecutive window that starts braking
    start_braking = None
    for gps in gps_data[1:]:
        if gps.speed <= previous.speed:
            # Set first record
            if start_braking is None:
                start_braking = gps

        # Speed not declining anymore, but we have found at least one braking record
        elif start_braking is not None:
            # Register the start and end timestamp
            braking_moments.append(start_braking.timestamp, gps.timestamp)
            start_braking = None

        previous = gps

    # Calculate mean readings, see listing 5
    return calculate_mean(braking_moments)
```

Listing 6: Pseudo implementation of calculating mean readings while the vehicle is braking.

```python
def realign_vertical(acc_data):
    # Get stationary readings (a_s)
    stationary_readings = calculate_mean_stationary_readings(acc_data)
    # The expected force we want to orient towards, i.e., 1G vertical force
    target_vector = (0, 0, 1)

    # Calculate rotation matrix
    R = calculate_3d_rotation(stationary_readings, target_vector)

    # Apply it for each element
    return R.dot(acc_data)
```

Listing 7: Pseudo implementation of reorienting accelerometer data vertically. Implementation for the second rotation matrix is similar. Except then we use mean readings while braking, and calculate a 2D rotation matrix.

4.4.3 Noise filtering

Accelerometer data is prone to noise. As is common with signal processing, data is first passed through a filter to obtain a clearer signal. Noise in the signal comes from vibrations generated by vehicle while driving. Existing research uses a variety of different filter configurations to clean the data. Some apply a low-pass Butterworth filter [9], whereas other researches use a high-pass Butterworth filter [12], [46]. Refer back to section 3.3 for explanation of Butterworth filter. As the configuration differs per research, we assume it depends per application.

In figure 23, various configurations of a 5th-order Butterworth filter are displayed. Butterworth filter is applied using the `butter` and `lfilter` functions from the computational library SciPy [74]. The top left (blue) plot shows the original signal while driving over a manhole. The other plot shows the resulting signal after applying a specific configuration. For instance, the middle right (orange) is after passing the data through a high pass filter with cut-off at 2 Hz. Meaning that all frequencies below 2 Hz are filtered out. As this signal seems most clean, we use this configuration as an initial starting point. In future, other types of configurations could be applied to optimize performance.

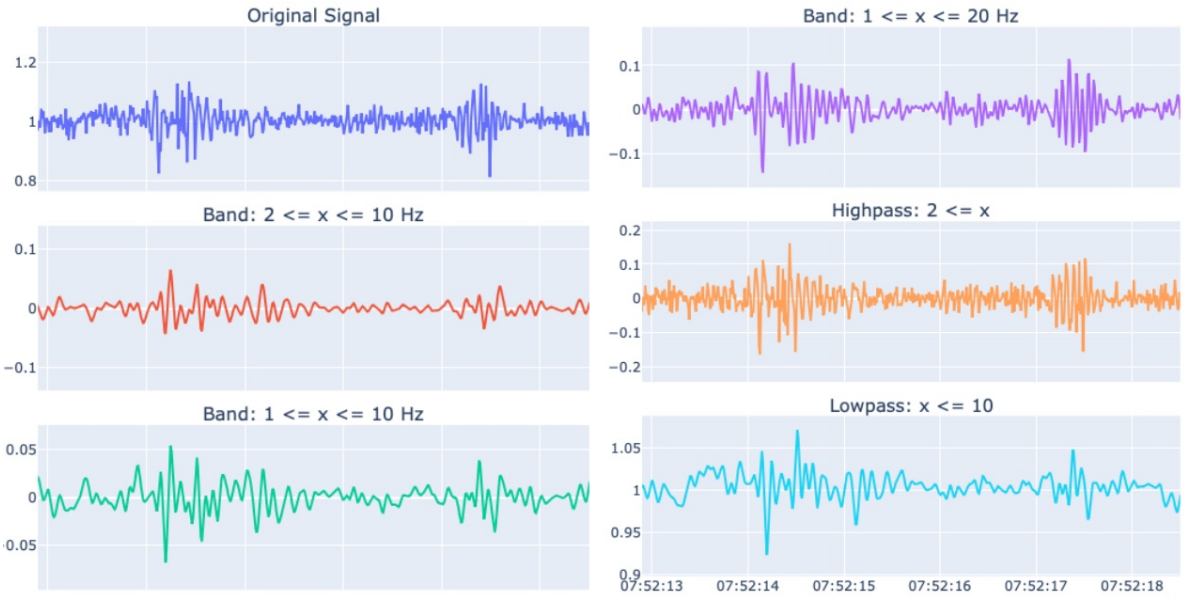


Figure 23: Figure displaying various graphs after applying a 5th-order Butterworth filter to the accelerometer signal on the vertical Z-axis. The event shown here is when the vehicle drives over a manhole.

4.5 Visual Data

The road surface is recorded with an Apple iPhone 12 Mini. The data is recorded at resolution of 1280 x 720 at 30 frames per second (FPS). The phone is located in a generic phone holder attached to the windshield. As can be seen in figure 14, the camera records the road from front faced perspective. The camera collects also other data than the road e.g., other vehicles, the sky, and other surroundings. Below we describe the processing steps for the visual data. We anonymize such information before uploading it to the data platform. To this aim, these processing steps are ran on a local machine, as opposed to previous steps.

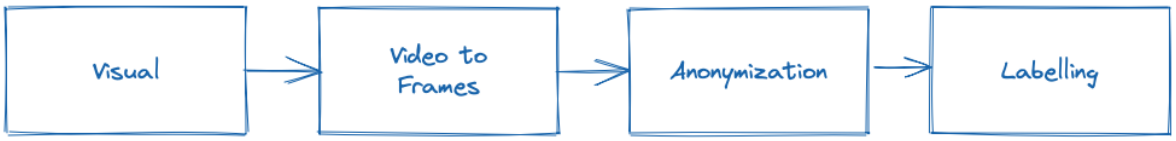


Figure 24: Figure displaying all processing operations for visual data.

4.5.1 Video to Frames

All visual data processing operations operate on single frames instead of video files. Therefore, the first step is to extract the distinctive frames from the video. We use the open-source FFmpeg [77] to convert a video into frames. Due the front faced perspective, the visual data contains a wide view of area. For instance, the horizon and the sky are also visible in the frame. Objects in the distance are smaller and more blurry making them harder to detect.

Therefore, we crop the exported frames to a smaller region where 1) there is less noise due surroundings, and 2) objects are more distinctive. Additionally, due the smaller image size, further computations are more efficient. The cropped area is selected as: the bottom half of the image, with an margin of 100 pixels from the bottom. With this margin we skip largely the area of the hood. Cropping of the frames is also implemented with the FFmpeg tool. An example is shown in figure 25, the red rectangle designates the cropped area.

Figure 25: Figure showing one of the collected frames. The red rectangle designates the cropped area.

4.5.2 Anonymization

Although not intentional, the camera may record other cars, persons, and other PII data. To safeguard any privacy concerns, the data is anonymized by blurring PII data from the individual frames. Detection of PII data happens with pre-trained object detector YOLOv5 model [62]. This model is trained on the COCO dataset [44], and is capable of detecting various classes. The following classes are blurred: person, bicycle, car, motorcycle, bus, train, and truck. After the frames are anonymized, the frames are uploaded to the data platform.

4.5.3 Labelling

As described earlier visual data is recorded with a smartphone at 30 frames per second. The visual data is manually annotated with open-source Computer Vision Annotation Tool (CVAT) [78]. Labelling of a frame is done by selecting all areas where a damage is present. See figure 26 for an example. Subsequently, the annotated data is exported and stored in the Data Platform. Note, annotating all frames is tedious as the change between between frames is minimal at 30 fps. Instead we downsampled the annotation to every tenth frame (i.e., 3 fps).

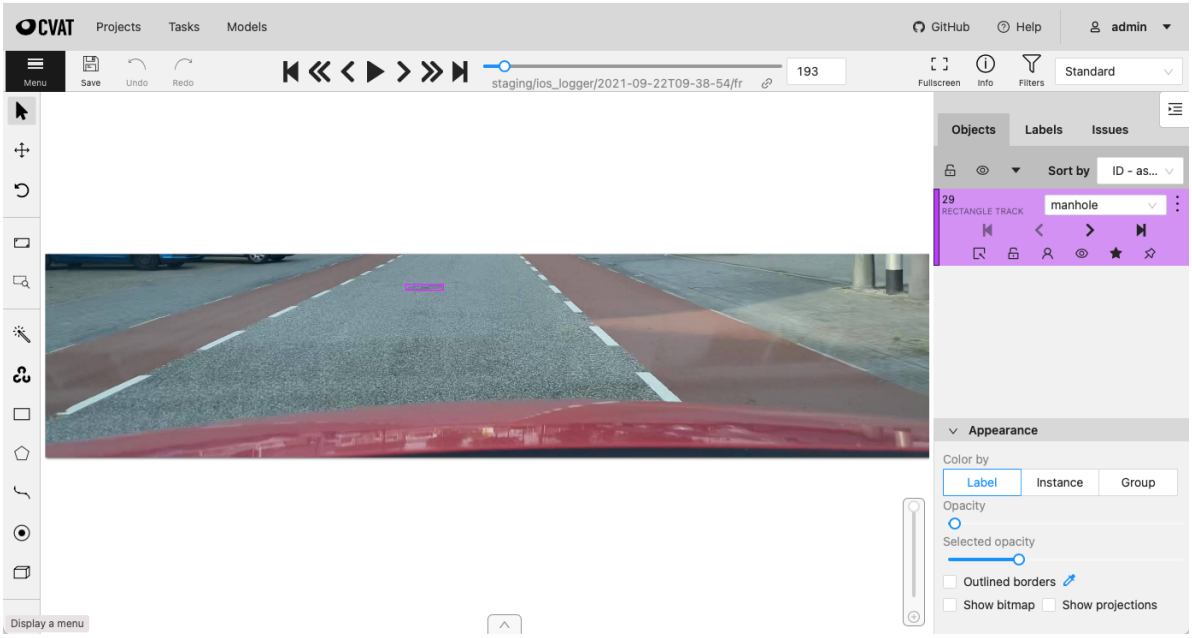


Figure 26: Screenshot of the annotation tool CVAT [78].

4.6 Exploratory Data Analysis

After applying the processing steps, we obtained a final dataset. The dataset consists of 17 trips, a total distance travelled over 246 km in 4 hours (average speed 62 km/h). The data is recorded over 10 different days. All data is collected in the Netherlands. A map of the travelled roads can be seen in figure 27. Note, this includes only data collected with the smartphone approach. Data from the AutoPi is ignored as the accelerometer data is deemed unreliable.

4.6.1 Road Types

To scope our research we only focus on detecting damages on asphalt roads. These types of roads cause less amount of vibrations than other types as roads (e.g., brick paved). These additional vibrations introduce additional noise for the accelerometer. When there is too much vibration, it is unlikely that the accelerometer is able to distinguish damages.

Additionally, brick paved roads are typically only used within urban areas. The majority of the roads (in Netherlands) are made from asphalt. Most types of damages are actually only present on asphalt roads. The proportion of road types per trip is shown in figure 28. The type of road is determined by the BGT [73].

Figure 27: Map where the data is collected. The red lines indicates the coordinates travelled while collecting data.

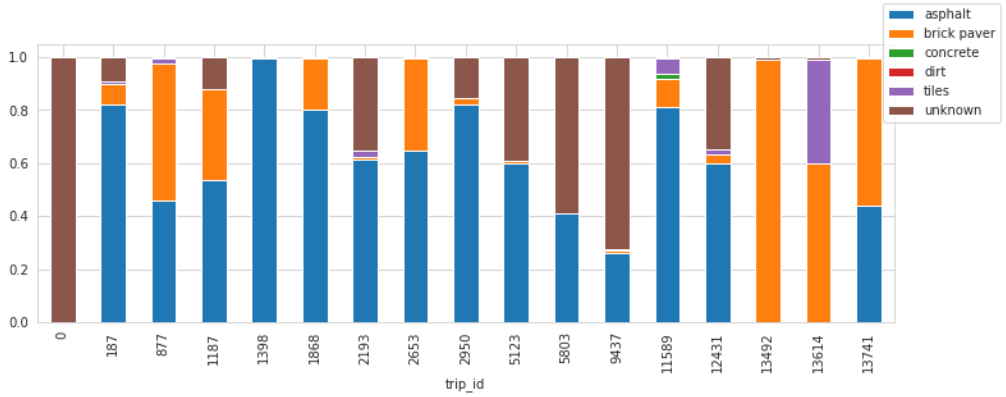


Figure 28: Proportion of the different road types per trip.

4.6.2 Labels

As described, the visual data is manually annotated. We use these labels as ground truth when making predictions. In figure 29 is shown the amount of labels per class. This illustrates a problem for our research. From the figure we observe that we have collected very few damages. Note, this figure includes the labelled data that was collected in combination with the AutoPi, i.e., data points where the accelerometer is unreliable. If we omit these labels, there are fewer damages.

To train a machine learning model we need many instances. However, with this few instances of damages it is impossible to train a model to detect road surface damages. The most prominent damage “Road mark” are faded road markings. This type of damage unlikely to be observed with the accelerometer, as the vehicle doesn’t drive over road markings often. During the research period, efforts were made to discover more damages throughout the country. However, we argue that the current state of the Dutch infrastructure is in such decent condition it is infeasible to collect enough data in the limited time period of this study.

However, the developed methodology using multimodal machine learning can still be applied in this context. Instead of detecting road surface damages we apply the method to detect manholes. In Netherlands manholes are present everywhere on local roads. Additionally, they are often located at a place where the wheel path travels over. This makes manholes a suitable target for our method. In figure 29, the orange bar represent the amount of manholes annotated.

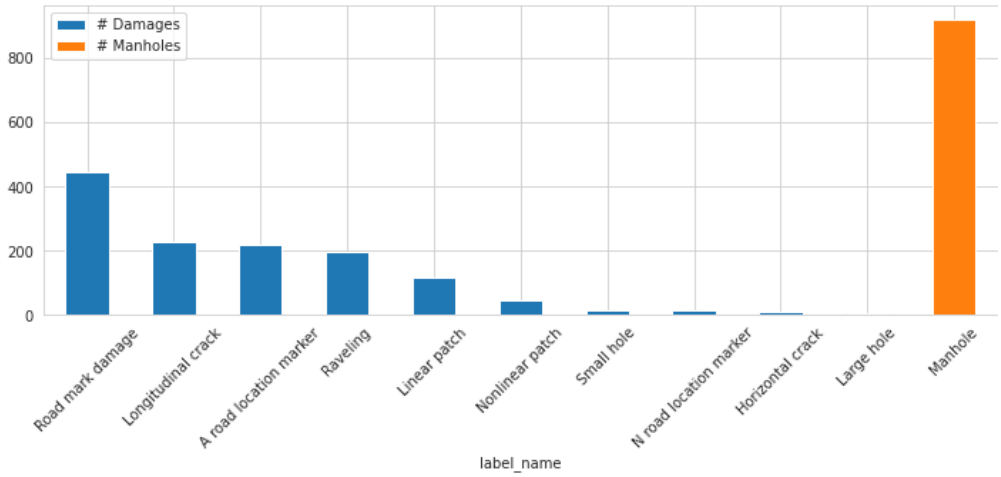


Figure 29: Figure displaying the distribution of labels. Blue bars denote the amount of damages. Orange bar denote the amount of manholes.

5 Experimental Research Designs

The aim of this research is detecting road defects using multimodal machine learning. As described earlier in section 2.6, multimodal means that a modal uses multiple data sources. In this case, we are dealing with visual- and accelerometer data. From research is known that there are typically three moments in the pipeline where the modalities can be fused: early-, middle-, or late fusion [10]. However, we can also create a modal on the distinctive modality. For instance, training a classifier solely using visual data. To this aim we compare the performance of a multimodal machine learning with that of a unimodal machine learner.

This means that we are developing three different machine learners: 1) model using visual data, 2) model using accelerometer, 3) model using combination of visual and accelerometer data. We refer to the respective experiment designs as *tracks*. The rest of the section will explain the rest of the research. We first explain how the dataset is constructed. Then we describe the evaluation metrics used to compare the models. Followed by section describing the machine learning model for each respective track.

Experiments in the different tracks are executed on the same machine. The machine has the following specifications: Intel Core i7-7700K CPU, NVIDIA GeForce GTX 1080 GPU, and 32 GiB RAM. The machine runs Debian 11 as operating system. Machine learning configurations, training performance, and results are tracked with an open-source tool called Weights and Biases (wandb) [79].

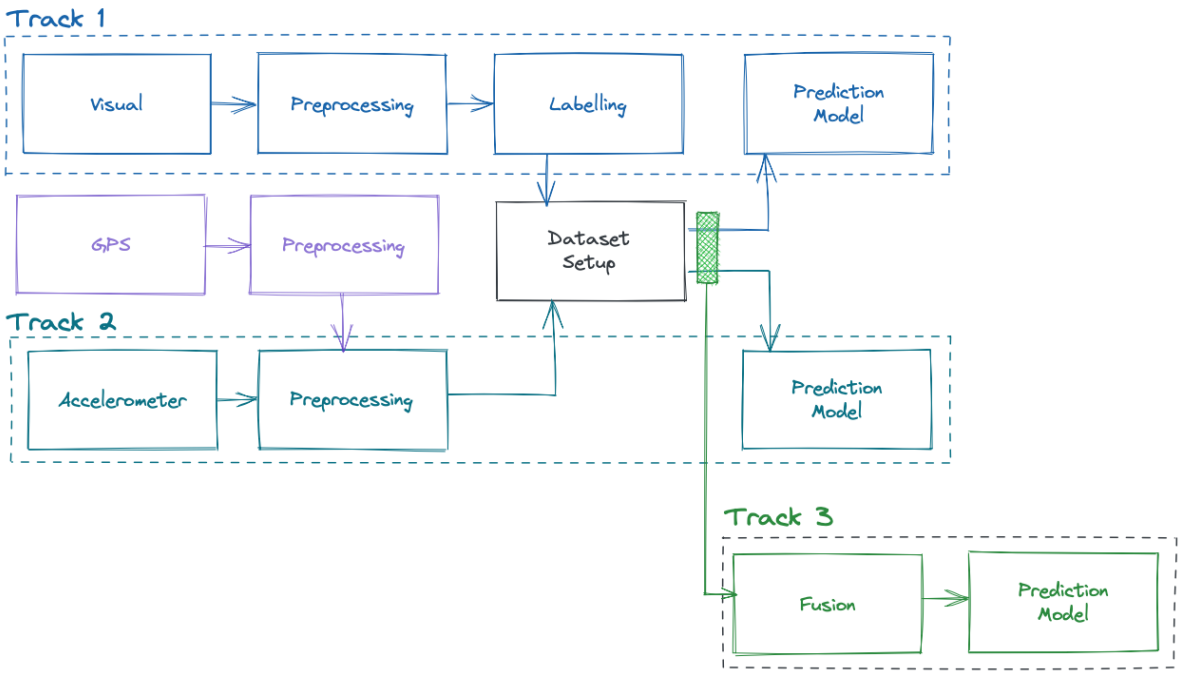


Figure 30: Visualization displaying the different research tracks. Each track is a different machine learning model.

5.1 Dataset Setup

In this work we develop three different machine learners using different data types as input. However, to make a comparison between the different machine learners, we create a common dataset that is used in each model. We cannot directly compare the performance of a visual model with that of an accelerometer based model. Both models need to be grounded to the same event. That is, we need to give both models input that covers the same road anomaly to make a valid comparison.

As mentioned earlier, to scope our research, we only look at manholes present on asphalt roads. Asphalt roads cause the least amount of vibrations while driving. It is assumed when noisy data is provided, the accelerometer is unable to learn from the data. Replicating the research on different road types is an interesting direction for future research.

Both data sources are annotated with timestamp of recording. We use this common denominator to segment the data in windows covering a fixed time period. Subsequently we label each segment positively if they contain a manhole, and negatively otherwise. Labelling of the segments uses a heuristic based on the frames of the visual data.

The target variable of the machine learners is whether a manhole is present in a segment. This makes the task a binary classification problem. The predictive variables depends on the respective research track. For track 1 (visual model) it is the frame. For track 2 (accelerometer model) it is the acceleration signal. And, for track 3 (fusion) it is both the frame and acceleration signal.

The data is segmented in sliding windows of one second in length, and each segment has an overlap of 50%. This is shown in figure 31. Detecting if a manhole is present in a segment is done with the following heuristic. First, we query all frames around the start time of the accelerometer segment. Specifically, we query the frames between -0.5 and $+0.2$ seconds. Remember that the visual data detects manholes earlier due the front faced perspective. Secondly, we filter the frames on following criteria: 1) there are one or multiple frames with a manhole, 2) the manhole in the image is detected at location it is likely the wheel drives over, 3) a threshold is used to judge if the segment contains a significant vibration. See listing 8 for a pseudo implementation. The window length of one second, and period of selecting the frames was empirically shown effective.

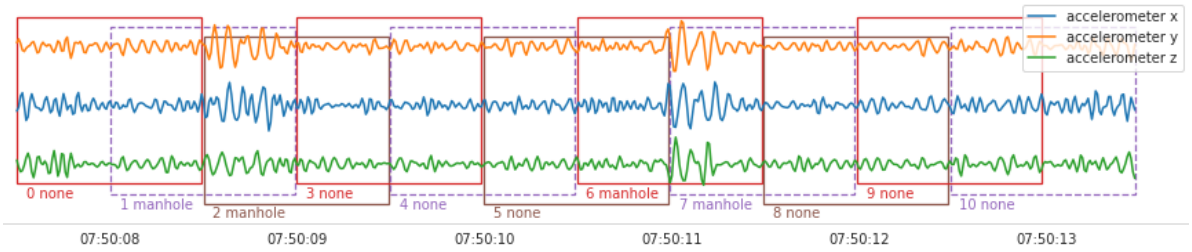


Figure 31: Example of segmentation. Each window is 100 samples / 1 second in length, and has overlap of 50%. Note, windows 1 / 2 and 6 / 7 are labelled as manhole. (Coloring of the segments are solely for illustrative purposes.)

```
def label_segments(segments):
    for segment in segments:
        # Find respective frames
        frames = query_frames(
            segment.start_at - 0.5 second,
            segment.start_at + 0.2 second
        )

        # Filter based on manhole detection
        frames = frames.where(frame.label == "manhole")

        # Filter based on vertical position
        # Should be in the bottom half of image
        frames = frames.where(frame.y > 0.50)

        # Filter based on wheel path
        frames = frames.where(
            (frame.x > 0.10) and (frame.x < 0.40) or
            (frame.x > 0.60) and (frame.x < 0.90)
        )

        # Calculate RMS sum of x, y, and z axis
        rms_sum = rms(x) + rms(y) + rms(z)

        if any(frames) and rms_sum > 0.05:
            segment["label"] = "manhole"
        else:
            segment["label"] = "none"
```

Listing 8: Pseudo implementation of automatically labelling segments.

5.2 Dataset Preparation

The dataset is split into training, validation, and test set. Each model is trained on the training set, optimized on the validation set, and the test set is used for calculating evaluation metrics. The splits are based on stratified sampling of their respective trip. This ensures we are not leaking any data, i.e., data from trip A is completely contained in one of the splits.

We aim to create a training set with 70% of the data, validation set with 21% of the data, and test set with 9% of the data. The dataset is sampled with the `GroupShuffleSplit` provided by scikit-learn [80]. However, due this sampling method it is not always possible to achieve the requested distribution. This depends on the amount of samples per trip.

Additionally the dataset is highly imbalanced. There are many more segments without a manhole labelled (1423), then samples containing one (212). An imbalanced dataset can be problematic while training a machine learning model. The training model will learn mostly from negative examples, and not learn enough from positive ones. The resulting model typically returns the majority class instead of actually learning the correct representation. Therefore, we randomly undersample the training set to be balanced. Figure 32 shows the distribution of the dataset splits after preparation.

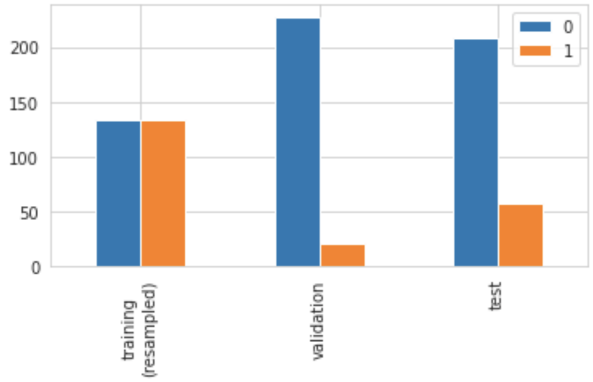


Figure 32: Distribution of the dataset sizes. Training set has 134-134 both positive-negative samples. Validation set has 21-227 positive-negative samples. Test set has 57-208 positive-negative samples.

5.3 Evaluation Metrics

The classification task is a binary problem. This makes the evaluation metrics tied to the amount of correct and incorrect predictions. This is often summarized in a confusion matrix. See figure 33. True positives and true negatives are predictions that are correctly classified. False positives are errors that are incorrectly marked as positive (i.e., the actual label is negative). Likewise, false negatives are errors that incorrectly marks as negative (i.e., the actual label is positive). From the amount of (in)correct predictions, we calculate the following metrics: Precision, Recall, F1 score and Average Precision.

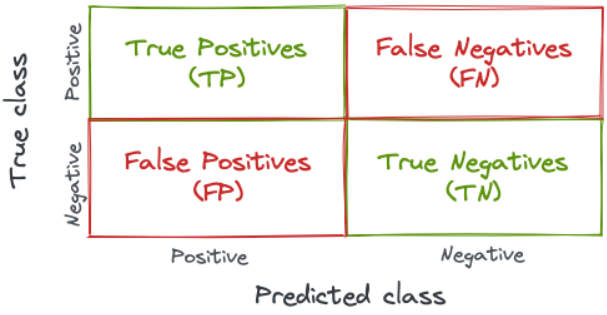


Figure 33: Confusion matrix summarizes the performance of a classification task.

Precision calculates the proportion of correct identified instances. It describes the correctness of positive classifications. Higher precision means that the model correctly classifies positive samples.

$$Precision = \frac{TP}{TP + FP}$$

Recall calculates the proportion of positives that are identified. It describes the sensitivity of the model. Higher recall means that the model finds more positive samples.

$$Recall = \frac{TP}{TP + FN}$$

F1 score combines precision and recall into one metric by calculating the harmonic mean between those metrics. It is possible to generalize F-score to give more importance to precision over recall, or vice-versa. In this thesis we use F1 score.

$$F1 = \frac{2}{\frac{1}{Recall} \times \frac{1}{Precision}}$$

Average Precision is the area under the precision recall curve (PRC). The precision recall curve visualizes the trade off between precision and recall at every possible recall threshold. See figure 34 for illustration. Similar to the AP, often reported is the Area Under the Receiver Operator Curve (AUROC). However it is generally regarded that usage of AP is preferred over AUROC when the dataset is imbalanced [48].

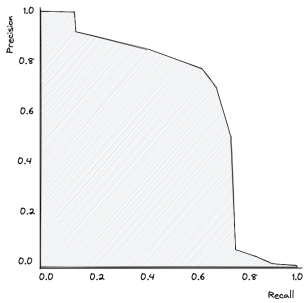


Figure 34: Example of precision recall curve. The area under the curve is known a Average Precision.

5.4 Track 1: Detecting Road Anomalies with Visual Data

The first track is to train a model solely using visual data to detect road anomalies. In this setting we can frame the task as object detection: classifying and locating defects in an image. As we know from literature survey (refer back to 3.1), YOLOv5 [62] is regarded as state-of-the-art for object detection. The authors provide various model configurations. Where each configuration differs in the size of the network.

During this track we train the visual model in two iterations. In the first iteration we train the original YOLO model to detect manholes. During this iteration the task is framed as object detection. We evaluate various configurations of YOLO. In the second iteration, we modify the best model to replace the object detection head. Instead, of detecting location of objects we want to compare the results with the other research tracks. Therefore, we replace the detection head with dense layers and frame it as binary classification task.

The rest of the section is outlined as followed. We first explain how the data is prepared to train YOLO. Then, we discuss how an object detector evaluates predictions. Followed by description on the experiment setup, i.e., how YOLO is trained. Finally, we describe the second iteration. How the model is transformed from object detector to a binary classifier.

5.4.1 Data Preparation

As described earlier in section 4.5, visual data is annotated with CVAT [78]. The labelled annotations are exported and converted from CVAT format to YOLO format. YOLO expects a certain dataset file (`dataset.yaml`) and directory structure to load the data. The dataset file describes which object classes exists, and allows configuration for hyperparameter tuning. Within the directory structure we have separate directories for training and validation data. Images containing one or multiple objects describe their respective objects in a label text file. In this label file, there is a single line for each object. The specification is `class x_center y_center width height`. Where `class` is the numeric index of the object class as defined in `dataset.yaml`. `x_center` and `y_center` describe the coordinates of the object, and `width` and `height` the size of the object. All values must be normalized between 0 and 1. Meaning that `x_center` and `width` are divided by the total width of the image, and `y_center` and `height` by the image height.

From literature we found a public dataset known as Road Damage Dataset [34]. This dataset contains annotated frames of road damages and manholes. The data was collected in Czechia, Japan, and India (see also section 2.1). During this track we supplement our training dataset with data from RDD. To this aim, we evaluate if we can improve our performance by introducing more data. The data from RDD is converted from Pascal VOC [49] format to YOLO format.

5.4.2 Evaluating Classifications

An object detector returns for an image all the detected objects. For each of the objects, the model returns: the detected class, the confidence, and the bounding box - that is where the object is located in the image. *Intersection over Union* (IoU) is used to determine whether the model positively classified a detected object. IoU measures the overlap between the detected bounding box and the ground truth bounding box (i.e., the annotated area). IoU is calculated by the ratio of the *area of intersection* over the total *area of overlap*. This ratio is visually displayed in figure 35. When the IoU is above some threshold, the object is positively classified as that respective class. Common used value for threshold is 50%.

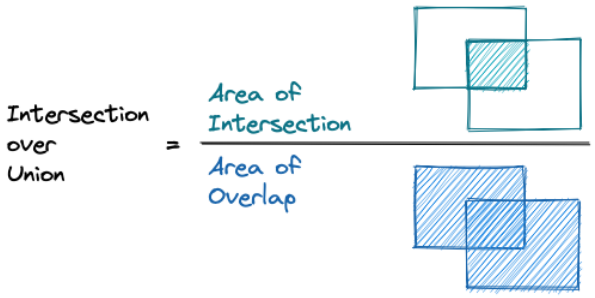


Figure 35: Visualization describing the calculation of the Intersection over Union (IoU).

5.4.3 Iteration 1: Training YOLO

For the first iteration, we directly use the visual annotated frames as input, i.e., we skip the segmented dataset as described earlier. The frames are split into training (70%), validation (21%) and test split (9%). The dataset is highly imbalanced: there are many more images without any anomalies. In object detection, images without an object are known as “background images”.

From experience, object detection models are typically less sensitive to imbalanced datasets when there are multiple classes. However, an object detector does fail to learn when the majority of the images does not contain an object. Therefore in this iteration we resample the training dataset in majority of frames with objects. This is done by selecting at most x% background images from each respective video. This yields an imbalanced dataset in favor of the object images. The author of YOLO also recommends to have small portion of background images to reduce false positives [81]. Note, we keep the original distribution for the validation and test set as it reflects the real world.

The author of YOLO provide various configuration of the model based on the network size. All configurations are pretrained on the COCO dataset [44]. This is a object detection dataset, containing 80 classes. When training YOLO to detect different types of objects (e.g., manholes), it is recommended to transfer learn on pretrained models instead of training from scratch [81]. Unfortunately, training the model takes relatively long (multiple hours). It is therefore not trivial to perform cross validation or optimize for hyperparameters. Therefore we only experiment with YOLO on the following configurations: model size, image input size, proportion of background images, and proportion of RDD images.

Generally larger models perform better. The smallest model `YOLOv5s` has 1.9 million parameters, and the largest model `YOLOv5x` has 86.7 million parameters. YOLO internally resizes images to a configured input size. When larger image size is configured smaller objects are more visible. Both configurations affect the training time, and are limited to the available resources. Larger models and images require more GPU memory. Unfortunately our setup is relatively limited, thus we are not able to train all possible configurations. We select the largest possible batch size for each configuration without exceeding the available memory.

5.4.4 Iteration 2: Modifying to Binary Classification

In order to compare the different tracks, we need to evaluate on the same dataset as described in section 5.1. To this aim, we modify the best performing YOLO model, and evaluate on the said segmented dataset. First, we export the trained model to Tensorflow [82]. Then, we replace the object detection head of YOLO with dense layers: 128 nodes with ReLU activation, and 1 output node with sigmoid activation. See also figure 36. Finally, the modified model is trained on the segmented dataset. This modified model performs a binary classification task whether the input frame of the segment contains a manhole.

Note, the layers of the YOLO model are frozen. This means that we keep the same parameters, and only train our replaced dense layers for binary classification. With this methodology we use the powerful YOLO model for feature extraction. The alternative approach is to train a network from scratch. However, transfer learning requires less resources e.g., less computational resources and less data [50].

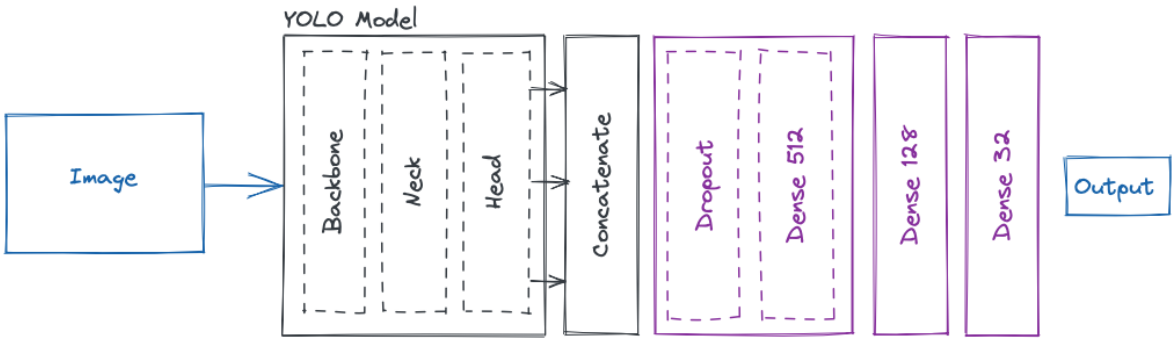


Figure 36: Architecture of the model for track 1: detecting manholes with visual data. It transfer learns retrained YOLO model to make binary classifications.

5.5 Track 2: Detecting Road Anomalies with Accelerometer Data

In this track we aim to detect anomalies solely using accelerometer data. We train a model that accepts a short data sequence. The sequence is short segment extracted from the continuous accelerometer signal. The segment is classified as positive when the car drives over a manhole, otherwise it has a negative label. As described in section 5.1, segments are one second in length. As the accelerometer data is sampled at 100 Hz, this means each segment contains 100 samples.

The models are evaluated on two different inputs, either the raw input or after performing feature extraction. When training on raw data, we directly pass the data from a segment to the model. As the segments are 100 samples in length, this means that we have 300 inputs (three channels for each of the X, Y, and Z axis).

The rest of this section describes the research design. We first explain the preprocessing operations such as feature extraction and feature scaling. Then, we describe the various machine learning models.

5.5.1 Feature Extraction

From earlier research we learned that Fast Fourier Transform (FFT) is often used as feature extractor for accelerometer data [7], [12], [13], [46]. Refer back to section 3.2 for more information on FFT. In this track we also evaluate the effect of extracting features with FFT on model performance. For each segment, we perform the FFT on each of the respective accelerometer axes.

We use scipy's `rfft` function to perform the calculation [74]. When calculating Fourier transform on real signals, the output is mirrored in real and negative halves (in literature this is known as conjugate symmetry [51]). Using `rfft` optimizes computation by only transforming the real half. The function returns complex numbers. In our case we are only interested in the presence of each frequency component. Thus, we calculate the magnitude of the complex number. This output is then used as input to the neural network. As we only use the real half, the input vector after feature extraction is 50 per channel (i.e., in total we have 150 inputs). See also listing 9 for more details.

```
from scipy.fft import rfft

for ax in axes:
    # ax is the raw input (i.e., 100 samples)
    yt = rfft(ax)
    # yt is now of length 50 (only real half of signal)
    # abs calculates the magnitude of the complex vector
    yt = abs(yt)
```

Listing 9: Pseudo implementation of performing feature extraction using FFT.

5.5.2 Feature Scaling

Using large values as input to a neural network leads to deteriorated performance. Models with large weights are usually unstable and are difficult to train [50]. Therefore we always perform feature scaling, regardless if we train the model on the raw acceleration data or after feature extraction. Feature scaling is done with function `StandardScaler` of scikit-learn's library [80]. The function scales all features by subtracting the mean and dividing by the standard deviation.

$$Z_i = \frac{X_i - \mu}{\sigma}$$

5.5.3 Experiment Setup

In this track we will train various deep neural networks to classify the segments. We also evaluate the performance of the networks based on the provided input. This either is the raw acceleration data or data after performing feature extraction (i.e., Fourier transform). There are endless possibilities when designing a neural network: how many hidden layers, how many nodes per layer, hyperparameters, etc. To scope our research we limit to three different types of architectures: Dense Neural Network (DNN), Convolutional Neural Network (CNN), and Long-Short Term Memory (LSTM). All models use Adam to optimize the gradients [52]. Learning rate is adjusted for each model for optimal performance.

Dense Neural Network. Also referred to as fully connected network, is a basic multi-layer neural network. Each node in each layer is fully connected with each node in the following layer [50]. In this experiment we use a neural network with the following architecture. 1) the input layer is reshaped from 100 x 3 to wide shape of 300, 2) a block is constructed with a dense layer with n nodes, dropout layer, and batch normalization layer. This block is repeated five times, where each following block has smaller node size, in this case: $256 \rightarrow 128 \rightarrow 64 \rightarrow 32$. The activation function of the dense layer is ReLU. 3) finally we have an output layer with sigmoid activation function. See also figure 48.

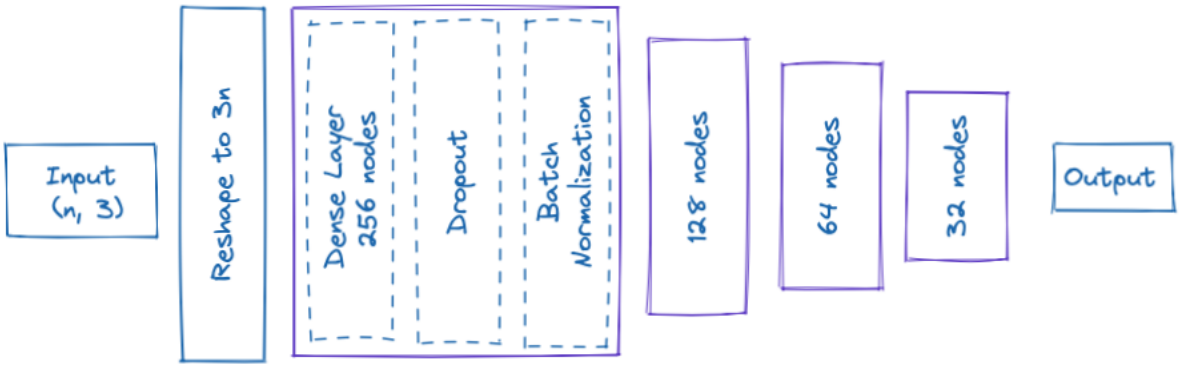


Figure 37: Architecture of the Dense Neural Network. *n* is 100 when raw input is passed, and 50 after applying feature extraction.

Convolutional Neural Network. With a CNN, the input is passed through one or multiple convolutional layers. A convolutional layer uses convolution kernels to convolve its inputs. It works as a filter and is then activated by non-linear activation function [50]. In this work we create the following network. 1) input of the three channels is directly passed through two convolutional layers. Both layers have 16 filters, a kernel size of 5, and ReLU as activation function, 2) the output of the filters is passed through Global Max Pooling layer, 3) subsequently two blocks of dense, dropout, and batch normalization are used (see above), 4) finally the output layer with the sigmoid activation function. See also figure 38.

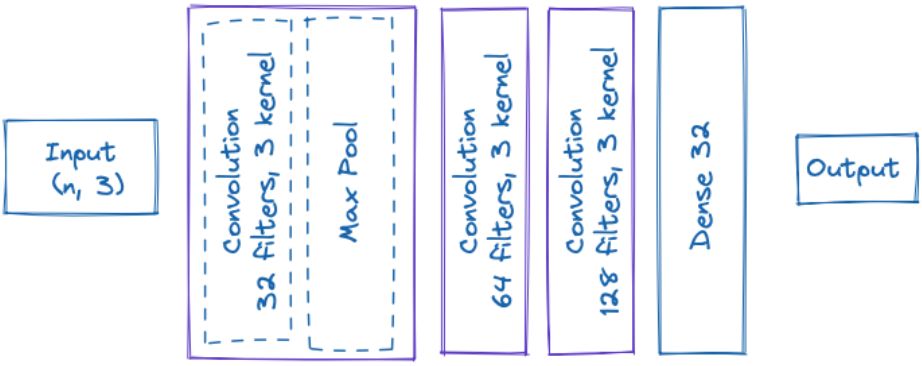


Figure 38: Architecture of the Convolutional Neural Network.

Long-Short Term Memory. A Recurrent Neural Network (RNN) can learn from the temporal relation between sensor readings. However RNN is very susceptible to gradient vanishing, making it difficult to learn. LSTM is variety of RNN. LSTM units allow gradients to flow unchanged. This makes LSTM networks able to learn long-term dependencies more easily [50]. LSTM networks are well suited for classifying time series data. We build our network as follows. 1) we have 3 LSTM layers with each 32 cells, 2) followed by dense layer of 32 nodes, 3) output. See also figure 39.

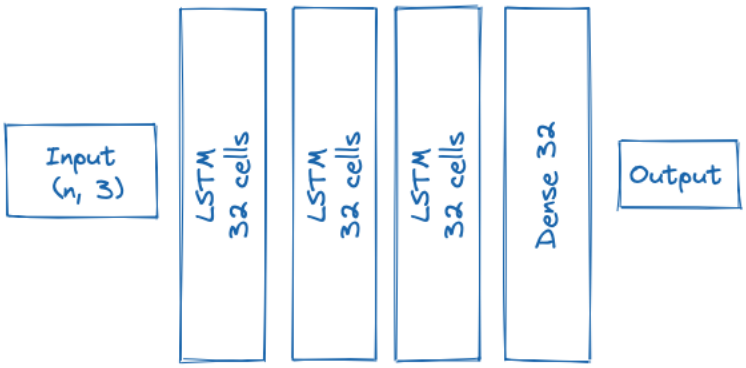


Figure 39: Architecture of the LSTM Network.

5.6 Track 3: Detecting Road Anomalies with Multimodal Data

In the final track we combine the models of track 1 and track 2. In this track we train a model that takes both visual and accelerometer data from each segment as input, and predicts if the respective segment contains a manhole. During this track we evaluate both hybrid and late multimodal fusion. Refer back to section 2.6 for more information about multimodal machine learning. Additionally, we design an experiment to validate that both data sources contribute to the outcome of the multimodal model.

5.6.1 Experiment Setup

First, we evaluate hybrid fusion (also called middle fusion). With hybrid fusion we aim to learn a joint representation inside the model. We concatenate the second last layers of both unimodal models. On this concatenated layer, we add a new detection head. This head consists of a dense layer with 32 nodes with ReLU activation. Followed by a single output node making the classification. The architecture of this network is shown in figure 40.

Secondly, we evaluate late fusion. In this experiment we keep both unimodal models intact i.e., each model makes their respective classification. The outputs are combined with a voting classifier. In this case we average the prediction probabilities of both models. Predictions are positively labelled when the averaged probability exceeds the threshold of 0.5. This method is also referred to as ensemble learning [10].

Finally, we validate that the hybrid model learns from both data sources. We evaluate this by presenting the model with only one source of data. The other data source is zeroed out. For instance, we evaluate on the test with only providing visual data and z zeros for the accelerometer input. When the model learns a joint representation, we expect that the model still is able to make correct classifications [10].

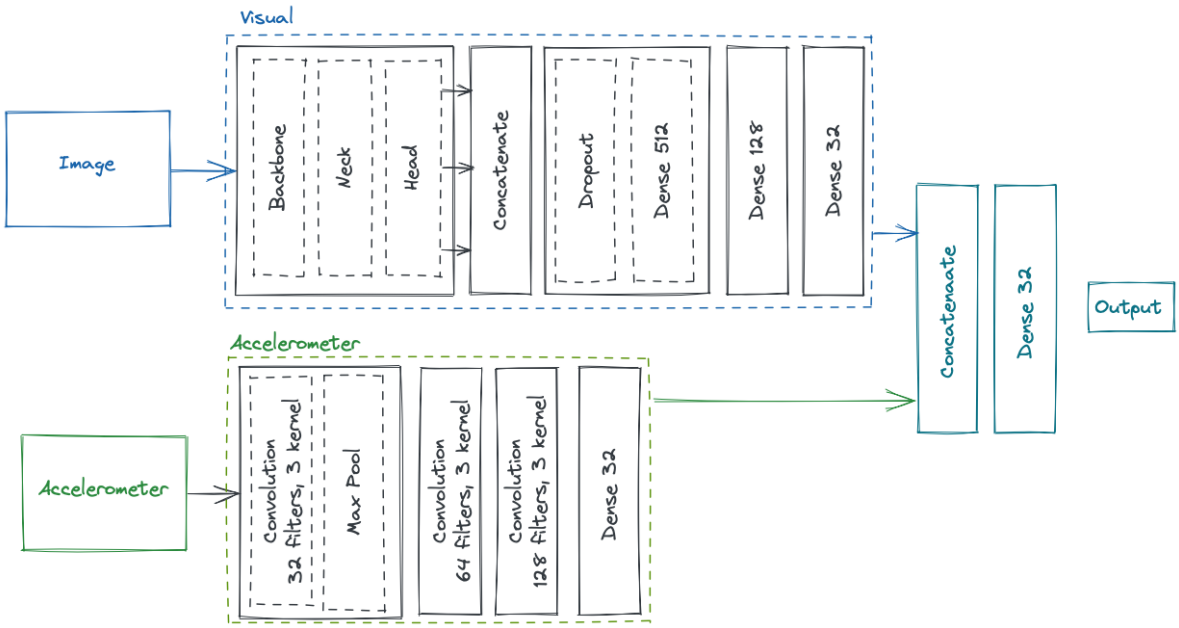


Figure 40: Architecture of the multimodal network.

6 Results

In this section we present the results obtained by following the research tracks as explained in section 5. We first evaluate the performance on the visual only model. Next, we evaluate which accelerometer based model performs best. Finally, we look at the performance after fusion both data sources.

6.1 Track 1: Detecting Manholes with Visual Model

As described section 5.4 we retrain YOLO to detect manholes. We evaluate with various configurations of network sizes, input image sizes, and including proportion of RDD dataset.

6.1.1 Iteration 1: Training YOLO

In the first experiment of this track we evaluate the effect of the model size and the input size. The result summary are listed in table 1, and the Average Precision over each epoch is shown in 41. Note, the results are for the first iteration, i.e., training YOLO as object detection.

From these results we conclude that the model `v1_s_1280_8` performs best. The Average Precision between this model, and the second best (`v1_m_1280_4`) are close (0.484 for small vs 0.473 for medium). Although the latter model has a higher precision, its recall is lower and the running time is significantly longer. When detecting road anomalies it can be argued that a higher recall is preferred over precision.

From the results we see that training with larger network size doesn't guarantee better performance in this task. This is likely as detecting manholes is relatively easy for YOLO. Increasing the input size of the image does improve performance. We can see that both model sizes have higher performance when the input size is larger. This makes sense as more pixels are preserved with a higher input size.

Note, during experiments the batch size is set to the largest possible value without exceeding the available resources. As the author of YOLO recommends, too small batches should be avoided as it impacts the learning performance [81]. Unfortunately the current machine was not capable of training larger YOLO models or when increasing the input size.

Name	Model Size	Input Size	Batch Size	Precision	Recall	AP	Runtime
v1_m_1280_4	YOLOv5m	1280	4	0.766	0.396	0.473	3h 46m
v1_m_640_16	YOLOv5m	640	16	0.521	0.345	0.316	51m
v1_s_1280_8	YOLOv5s	1280	8	0.622	0.465	0.484	1h 26m
v1_s_640_16	YOLOv5s	640	24	0.512	0.245	0.298	36m

Table 1: Evaluation metrics of various YOLO configurations. From these results we conclude that the model `v1_s_1280_8` performs best.

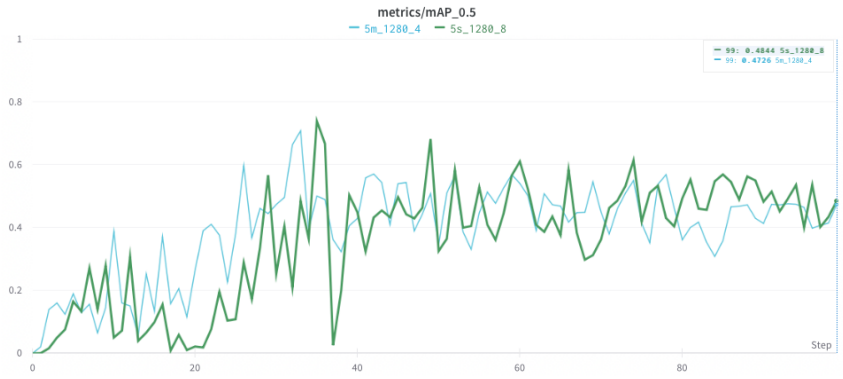


Figure 41: Graph showing the Average Precision for the two best models over each epoch.

In the next experiment, the model (`v1_s_1280_8`) is further trained on additional data from the Road Damage Dataset (RDD) [60]. From the RDD we select 80 random images (10% of the training set) where a manhole is visible, and 40 random background images (5% of the training set). Remember that the dataset is collected in different countries, therefore the manholes in those countries look different. To this aim, we evaluate if our model is able to learn from a more generalized dataset.

The results are shown in table 2. As we can see the performance of both models are very comparable after training for some while. Note, the model without RDD early stopped after training for 72 epochs. This is because the validation loss stopped declining, indicating that the model is saturated and does not learn further. From figure 42, we see that the model with RDD takes more epochs to reach comparable performance. However, with this dataset the model learns from a more generic dataset.

Name	Epochs	Precision	Recall	AP	Runtime
v2_with_rdd	100	0.752	0.599	0.649	2h 56m
v2_without_rdd	72	0.746	0.610	0.644	1h 4m

Table 2: Evaluation metrics of augmenting the training

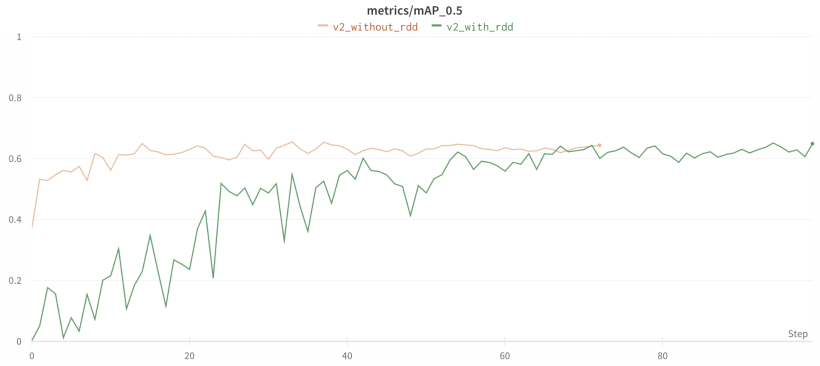


Figure 42: Graph showing the Average Precision for the two best models over each epoch.

6.1.2 Iteration 2: Modifying to Binary Classification

In the second iteration, we modify the best YOLO configuration. We replace the object detection head with a head capable of binary classification. This makes the model suitable for comparison with the models developed in other tracks. The model `v1_s_1280_8` is used as a base. As we found that supplementing the dataset with RDD did not yield significant improvements.

The model is modified as described in section **??**. The layers of the YOLO model are kept frozen, i.e., the parameters are kept and not further trained. The final layers of the model are trained on the segmented dataset to perform classification. This makes the learning task of the model easy. The model is trained for only 20 epochs. The batch size of the model is configured to be 1. Higher batch sizes causes memory errors.

The training results can be seen in the plots of figure 43 and in table 3. The top-left shows the loss over epoch. After 20 epochs, we can see that the training and validation loss are both low. This indicates that the model learned sufficiently and is neither over- or underfitting. This is also confirmed by the top-right plot. Here is shown the Precision Recall curve. The model has a perfect performance. This confirms that the learning task of making classifications is easy for the model.

Model	Precision	Recall	AP	F1
Visual Only	0.86	1.0	1.0	0.93

Table 3: Evaluation summary of the visual model.

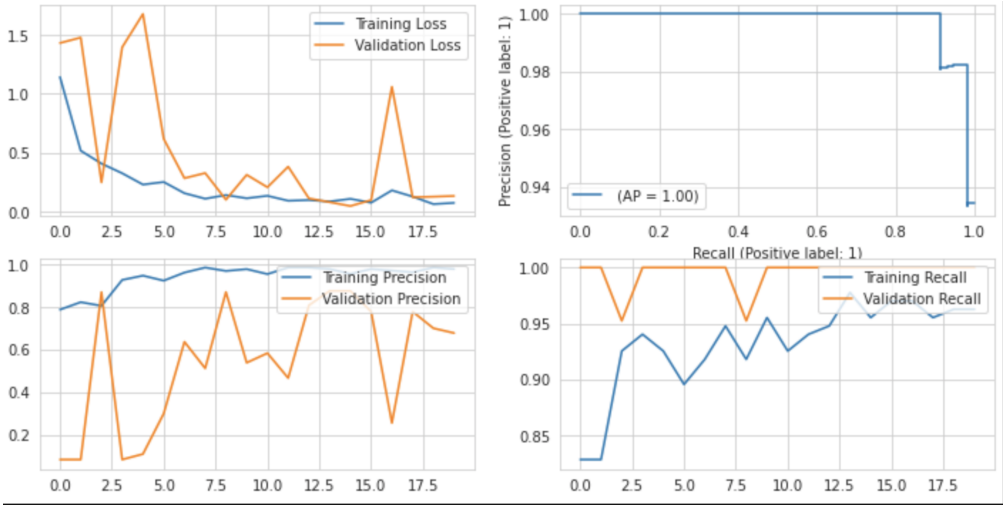


Figure 43: Showing various plots of the learning performance of the visual only model. In the top-left is the training and validation loss over each epoch. In the top-right the precision recall curve for the test set. The bottom plots show the precision and recall over each epoch.

6.1.3 Conclusion

After iteration 2, we developed a model capable of classifying whether a segment contains a manhole. This model performs really well on the test set. This is attributed due the YOLO model. As YOLO is currently state-of-the-art in object detection, making classifications is relatively easy task.

6.2 Track 2: Detecting Manholes with Accelerometer Data

As described in section 5.5 we train various models using accelerometer data. The input to the models is either the raw accelerometer data, or after performing Fourier transform. During this track we evaluate three network architectures: Dense Neural Network, Convolutional Neural Network, and LSTM Network. In total this means we train and evaluate 6 different machine learners.

Each model is optimized using Adam with learning rate of 0.001. The model is trained for 100 epochs, and restored to the best weights before evaluating test scores. To this aim we achieve better generalization in case the model overfits on the training set [50]. The results can be found in table 4. The reported statistics are calculated on the held out test set. Note, the threshold for positive prediction are all kept constant at 0.5 for a equivalent comparison.

In general, we see that all model architectures perform better on the raw accelerometer data instead after applying feature extraction. It can be seen that the CNN models outperform the other models. This is followed by the LSTM network operating on raw data. Finally, the DNN based model performs relatively poorly. Next we dive into the specific learning performance for each model. As all models operating on the raw data performs better, we only report on the raw version.

Name	Precision	Recall	AP	F1
CNN (raw)	0.29	1.0	0.51	0.45
LSTM (raw)	0.39	0.95	0.48	0.55
Dense (raw)	0.28	0.77	0.34	0.41
CNN (FFT)	0.28	0.95	0.51	0.44
LSTM (FFT)	0.31	0.89	0.25	0.46
Dense (FFT)	0.36	0.42	0.41	0.39

Table 4: Evaluation summary of the various machine learning models.

6.2.1 Dense Neural Network

In figure 44 we see the learning performance. As could be seen earlier in the table, the DNN model does not yield great performance. Especially lacking is the recall of the model. From the top-left plot we can see that the model learns from the data. After epoch 50 starts overfitting. This is indicated as the validation loss keeps increasing with further epochs.

Figure 44: Showing various plots of the learning performance of the DNN accelerometer only model. In the top-left is the training and validation loss over each epoch. In the top-right the precision recall curve for the test set. The bottom plots show the precision and recall over each epoch.

6.2.2 Convolutional Neural Network

From the accelerometer based models, the CNN performs best. The learning performance is shown in figure 45. Again the model shows sign of overfitting after epoch 50. This can be attributed due the high recall rate. However, this might indicate that this model is affected due the imbalanced dataset. Regardless, from the evaluation on the test set in the precision recall curve, we see acceptable performance. At recall rate of 0.8, the model evaluated with a precision of almost 0.7.

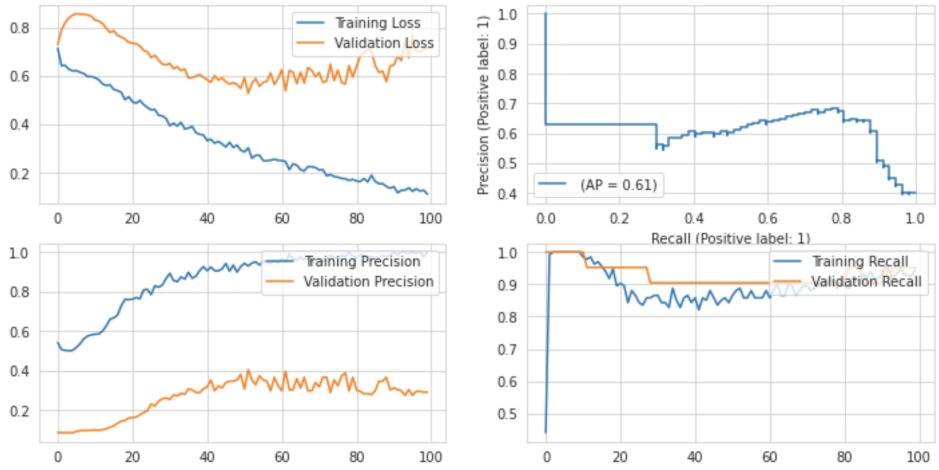


Figure 45: Showing various plots of the learning performance of the CNN model. In the top-left is the training and validation loss over each epoch. In the top-right the precision recall curve for the test set. The bottom plots show the precision and recall over each epoch.

6.2.3 LSTM Neural Network

Finally we have the LSTM based model, see figure 46. This model scores second best, but from the top-left loss plots we see that this model quickly overfits. Additionally, due the spikes in the validation loss, we observe that this model is not training very well. The precision recall curve shows a relative flat curve. However, the overal precision of the model is underperforming with that of the CNN based model.

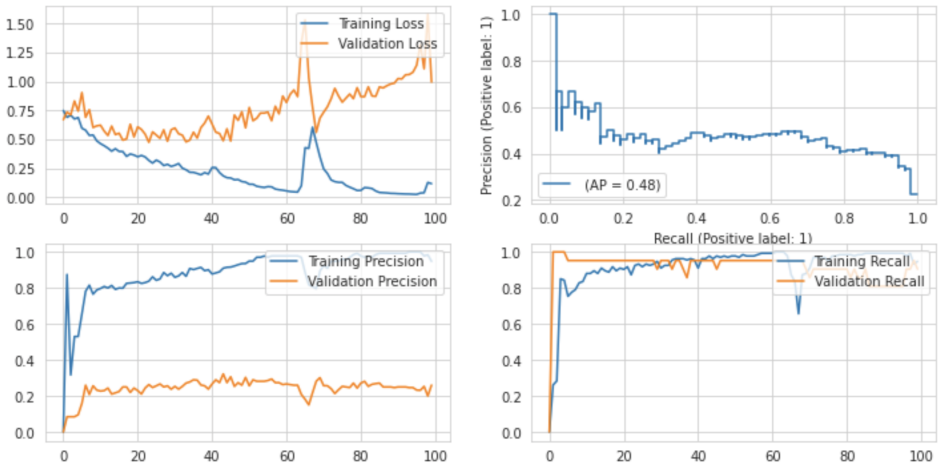


Figure 46: Showing various plots of the learning performance of the LSTM model. In the top-left is the training and validation loss over each epoch. In the top-right the precision recall curve for the test set. The bottom plots show the precision and recall over each epoch.

6.2.4 Conclusion

From the summary table we see that the precision and recall scores are relatively close for the CNN and LSTM based model. However, the Average Precision (AP) of the CNN model is significantly higher. This is also confirmed when comparing the respective Precision-Recall curves, see figure 47. The figure confirms that the CNN model (orange line) is able to make more accurate predictions at higher recall rates. This is indicated as the precision remains high, while the recall increases.

From the results we conclude that the CNN model without feature extraction performs best. We use this model in the following section, where we combine both visual and accelerometer data.

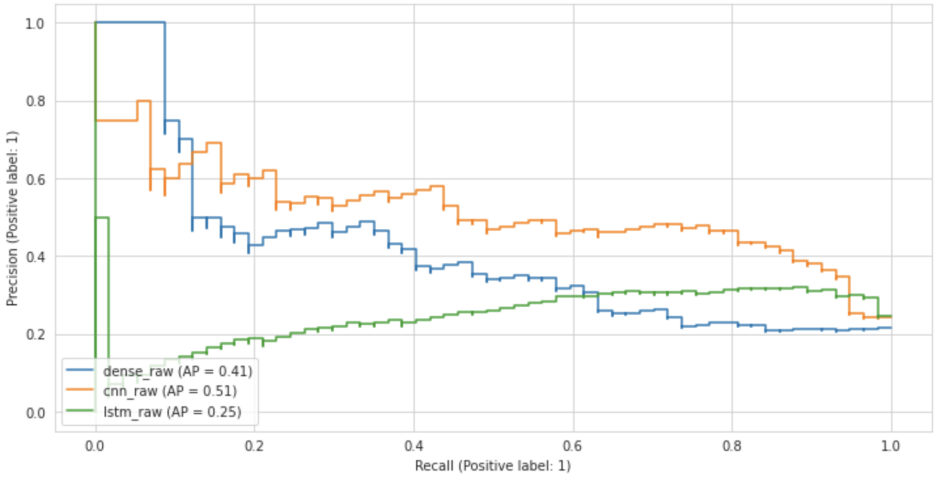


Figure 47: This figure plots the Precision Recall Curve for each of the evaluated network architectures. For clarity only the models operating on raw input are shown.

6.3 Track 3: Detecting Manholes with Multimodal Machine Learning

In the final track, we combine the earlier developed unimodal models. As described in section 5.6, this is done with both hybrid and late fusion. The results can be seen in table 5. For comparison, the performance metrics are repeated of the unimodal models.

From table we can see that all models have a perfect recall of one. This means that all models are always able to identify all positives samples. The models vary in their precision. Unfortunately, the fusion based models do not outperform the unimodal models. The best performing model is that of the visual only model.

Name	Precision	Recall	AP	F1	Parameters
Visual Only	0.86	1.0	1.0	0.93	7,566,817
Accelerometer Only	0.29	1.0	0.61	0.45	35,393
Hybrid Fusion	0.66	1.0	0.88	0.79	7,602,144
Late Fusion	0.62	1.0	1.0	0.77	7,602,210

Table 5: Evaluation metrics on the test set as reported by the respective models.

The hybrid fusion model adds layers to the network to combine both data sources. These layers are trained for 20 epochs using Adam optimizer with learning rate of 0.001. To this aim, we train a model which learns how to combine both data sources. The learning performance of the hybrid fusion model in figure 48. Related to the visual based model we observe that the model quickly starts to overfit. Regardless, the model as a good performance. The precision recall curve indicates a good fit.

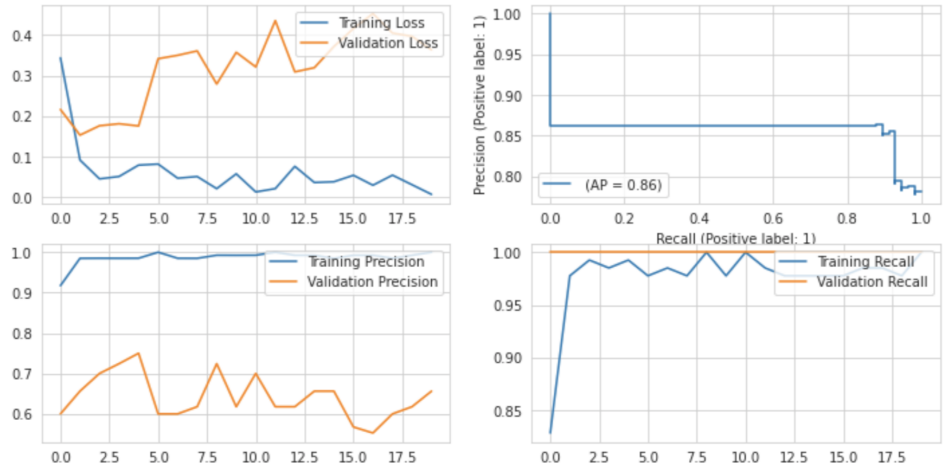


Figure 48: Showing various plots of the learning performance of the hybrid fusion model. In the top-left is the training and validation loss over each epoch. In the top-right the precision recall curve for the test set. The bottom plots show the precision and recall over each epoch.

The late fusion model works by combining both unimodal representations after each respective model has made a decision. The final decision is made by a voting classifier. In this case, we average both probabilities. When it exceeds the threshold of 0.5, we classify the output as positive. This means that the model does not learn a joint representation. Therefore, we cannot report further results on this model except the reported statistics in table 5.

6.3.1 Evaluation on Single Input

To test if the hybrid fused model actually learns a joint representation, we evaluate the performance when one of the inputs is zeroed out. This experiment is performed for both zeroing out the visual and the accelerometer data. The results are shown in table 6.

We can see from the results that the model still works when either input is zeroed out. This confirms that the model is able to learn a joint representation. Interestingly is that the model performance increases when only visual data is provided.

Name	Precision	Recall	AP	F1
Hybrid Fusion with Both Inputs	0.66	1.0	0.88	0.79
Hybrid Fusion with Visual Only	0.90	1.0	1.0	0.95
Hybrid Fusion with Accelerometer Only	0.68	0.77	0.61	0.72

Table 6: Evaluation of the hybrid fusion model where either both or each of the respective unimodal sources is provided.

6.3.2 Conclusion

From the results we conclude that both multimodal fusions work. Admittedly, we see that that the unimodal vision only model performs best. This implies that the accelerometer data is not well prepared. We discuss this further in the following section.

In figure 49, a matrix is shown where each model predicts the classification based on the input. Each image is a frame from the respective segment. For each image is shown the annotated label, and for each model the prediction outcome.

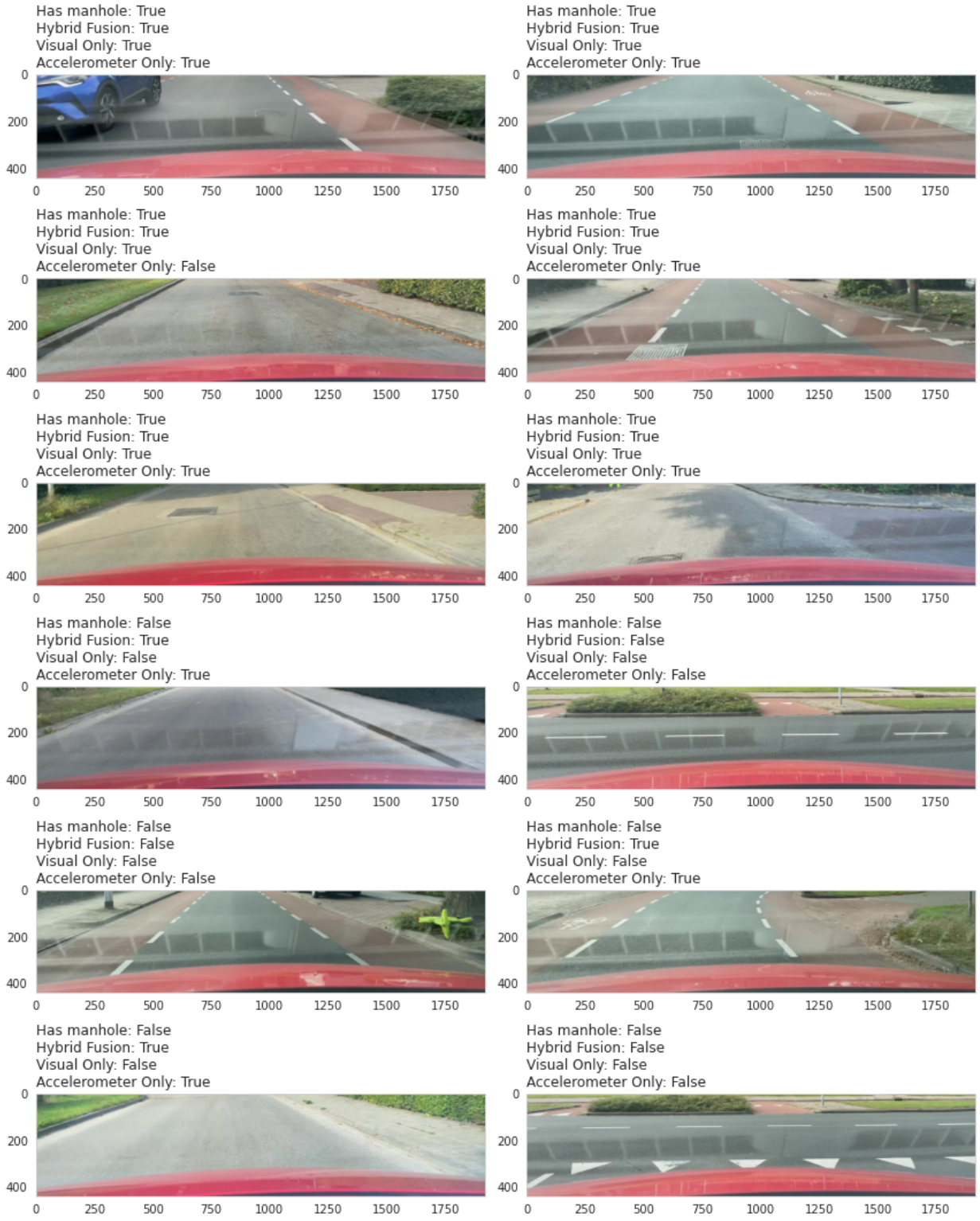


Figure 49: Matrix of prediction examples.

7 Discussion

This section discusses our findings. First, we interpret and discuss the results of our final model. Next, we reflect on the research questions. Then, we propose future research directions worth pursuing. Finally, we disclose limitations of this work.

7.1 Results

In general, the results are good. From the various experiments we observe that the accelerometer based model performs poorly in comparison with the visual model. In addition, from the results of track 3, we can see that the hybrid fused model performs best when it is only provided with visual data. We argue that there are two reasons for this observation.

First of all, during data collection we suffer from partial observability. The visual data captures data from a front faced perspective. This means that the visual data source is able to capture the full context. However, we can only observe the same object with the accelerometer when the object is in the path of the vehicle. This results that the accelerometer data in the segmented dataset contains false positives. Learning from data containing false positives deteriorates precision.

Secondly, the dataset is labelled based on the visual data. This means that the visual data acts as the ground truth. It is therefore expected that the visual model performs best. The accelerometer data is positively labelled based on a heuristic. Within the heuristic there are some improvements that can be made. These will be discussed later as future research ideas.

From the experiments we notice that all models have a high recall. However, this is probably an effect of our dataset setup. Overall, the original dataset is highly imbalanced. In order to train machine learning models, we undersample the training set. In addition, the size of the dataset is relatively low. When provided enough training data, we expect a more balanced trade off between precision and recall. Regardless, we argue that a high recall is preferred when detecting road surface anomalies.

7.2 Research Question

The main research question is supported by the following two sub research questions:

- *SQ1: What is the state of art in machine learning pipelines using object detection to classify road surface anomalies?*
- *SQ2: What is the state of art in machine learning pipelines using accelerometer data to classify road surface anomalies?*

The sub research questions are answered in our literature survey in section 2. In summary, we found that there is a lot of research known on detecting road surface defects using visual or accelerometer data. Unfortunately, we are unable to compare our work to existing literature. In the limited time period we were unsuccessful in collecting enough road surface damages to train machine learning models.

In this work the following research question was central. *To what extent can visual object detection and accelerometer data be combined in a multimodal machine learning pipeline to correctly classify road surface anomalies?*

From our method and results, we successfully developed a multimodal machine learning pipeline. Specifically this is achieved with both both hybrid and late fusion. Due the limited dataset we classified manholes instead of actual damages. Regardless, our work contributes a novel method to detect road surface anomalies using multimodal machine learning.

7.3 Future Research

In this work there are a couple opportunities for further research. First of all, is applying the proposed method to detect actual damages. Due the limited time period of this work it was infeasible to collect enough data.

Second opportunity is improving the method how the dataset is constructed and labelled. Labelling based on the visual data is an easy method. However, as mentioned earlier this also introduces false positives for the accelerometer input due partial observability. For future work, it is advised to reevaluate the heuristic in determining whether the car travels over an object.

Thirdly, this work only evaluated binary classification whether a manhole is present in a segment. Combining visual and accelerometer data however has an additional benefit that is worth pursuing. With visual data we have information about the location of an object, and accelerometer data provides information about the impact on the vehicle. In case of road damages, the accelerometer can describe the severity of the damage. Combining the data sources allows to create a model that can both localize an anomaly - based on visual data, and describe the impact - based on the accelerometer data.

Finally, generalization of the method is interesting for future research. Current results were only collected with single smartphone and car. Additionally, the results were evaluated on a single type of road. Finally, the data was collected in a single country. The infrastructure in different countries might be different. For future work, it is interesting to see if the results can be replicated in different circumstances.

7.4 Limitations

In this work we address the limitations based on the four threats to validity: internal, external, construct, and conclusion.

Internal Validity. The dataset used in this study is labelled according to the annotated visual frames. It can be assumed that for the visual model the results are valid. Note, annotations can be improved when they are reviewed by a second person. Regardless, based on the visual annotations, we automatically label the data of accelerometer. Due this automatic labelling we introduce false positives which threatens our results for the accelerometer based models.

External Validity. All data was collected with the same smartphone device, device holder and vehicle. Generalizability to other settings might be limited. In this work, care was taken that the data processing steps are generalizable (e.g., the automatic reorientation of the accelerometer axes). Additionally we expect that the visual data is generalizable. However, a different smartphone or vehicle might report accelerometer data differently.

Secondly, we have only researched the effects on single road type, i.e., asphalt. When generalizing to other types of road, we expect that a different signal on the accelerometer sensor. For instance, road types with more vibrations (i.e., brick pavers) introduce more noise.

Construct Validity. We validated that the multimodal actually learns from the data. We tested this in the last experiment, where we zeroed out either of the input source. The model was still able to make correct classifications.

Conclusion Validity. During our research it was infeasible to perform cross validation due the long training times of the neural networks. In future research it is advised to use cross validation to validate the conclusions.

8 Conclusion

In Netherlands the current process of performing road maintenance takes long, is costly and largely relies on subjective observations. In recent years, the need for data driven inspection emerged to perform automatic road quality assessment. Research has been performed to build machine learning models to classify models. However, existing work only uses a single source of data. In this work, we contribute a novel method combining both visual and accelerometer data to detect road surface anomalies.

Unfortunately, our dataset was too small to classify actual damages. Instead we resorted to classifying manholes. We created three models to detect manholes. The first model only uses visual data. The second model only uses accelerometer data. The third model uses a combination of both visual and accelerometer data.

Our results indicate that multimodal machine learning is successful in detecting manholes. Although the combination of both data sources does not outperform unimodal models, the approach is an interesting scientific contribution. After training the multimodal model, predictions can be made when only one data source is provided.

For future work, there are interesting opportunities to pursue. For instance, by combining visual and accelerometer data enables a model to both localize and describe the impact of a road surface anomaly.

9 References

9.1 Scientific Articles

[1] M. Jahanshahi, F. Jazizadeh, S. Masri, and B. Becerik-Gerber, "An unsupervised approach for autonomous pavement defect detection and quantification using an inexpensive depth sensor," *Journal of Computing in Civil Engineering*, vol. 27, Aug. 2012. DOI: 10.1061/(ASCE)CP.1943-5487.0000245.

[2] L. Zhang, F. Yang, Y. Zhang, and Y. Zhu, "Road crack detection using deep convolutional neural network," Sep. 2016. DOI: 10.1109/ICIP.2016.7533052.

[3] A. Zhang, K. Wang, B. Li, E. Yang, X. Dai, P. yi, Y. Fei, Y. Liu, J. Li, and C. Chen, "Automated pixel-level pavement crack detection on 3d asphalt surfaces using a deep-learning network: Pixel-level pavement crack detection on 3d asphalt surfaces," *Computer-Aided Civil and Infrastructure Engineering*, vol. 32, Aug. 2017. DOI: 10.1111/mice.12297.

[4] S. Chatterjee, P. Saeedfar, S. Tofangchi, and L. Kolbe, "Intelligent road maintenance: A machine learning approach for surface defect detection," Jun. 2018.

[5] H. Maeda, Y. Sekimoto, T. Seto, T. Kashiyama, and H. Omata, "Road damage detection and classification using deep neural networks with smartphone images: Road damage detection and classification," *Computer-Aided Civil and Infrastructure Engineering*, vol. 33, Jun. 2018. DOI: 10.1111/mice.12387.

[6] H. Maeda, T. Kashiyama, Y. Sekimoto, T. Seto, and H. Omata, "Generative adversarial network for road damage detection," *Computer-Aided Civil and Infrastructure Engineering*, vol. 36, Jun. 2020. DOI: 10.1111/mice.12561.

[7] T. Hanson, C. Cameron, and E. Hildebrand, "Evaluation of low-cost consumer-level mobile phone technology for measuring international roughness index (iri) values," *Canadian Journal of Civil Engineering*, vol. 41, pp. 819–827, Sep. 2014. DOI: 10.1139/cjce-2014-0183.

[8] W. Buttlar and S. Islam, "Integration of smart-phone-based pavement roughness data collection tool with asset management system," Nov. 2014.

[9] A. Gupta, S. Hu, W. Zhong, A. Sadek, L. Su, and C. Qiao, "Road grade estimation using crowd-sourced smartphone data," Jun. 2020.

[10] T. Baltrusaitis, C. Ahuja, and L.-P. Morency, "Multimodal machine learning: A survey and taxonomy," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PP, May 2017. DOI: 10.1109/TPAMI.2018.2798607.

[11] D. Arya, H. Maeda, S. Ghosh, D. Toshniwal, A. Mraz, T. Kashiyama, and Y. Sekimoto, "Transfer learning-based road damage detection for multiple countries," Aug. 2020.

[12] C. Wu, Z. Wang, S. Hu, J. Lépine, X. Na, D. Ainalis, and M. Stettler, "An automated machine-learning approach for road pothole detection using smartphone sensor data," *Sensors*, vol. 20, p. 5564, Sep. 2020. DOI: 10.3390/s20195564.

[13] A. Basavaraju, J. Du, F. Zhou, and J. Ji, "A machine learning approach to road surface anomaly assessment using smartphone sensors," *IEEE Sensors Journal*, vol. PP, pp. 1–1, Nov. 2019. DOI: 10.1109/JSEN.2019.2952857.

[14] M. Sayers, T. Gillespie, and W. Paterson, "Guidelines for conducting and calibrating road roughness measurements," *World Bank Technical Paper Number 46*, 1986.

[15] ISO 8608, "Mechanical vibration–road surface profiles–reporting of measured data," *International Standards Organisation*, 1995.

[16] J. Jeong, H. Jo, and G. Ditzler, "Convolutional neural networks for pavement roughness assessment using calibration-free vehicle dynamics," *Computer-Aided Civil and Infrastructure Engineering*, vol. 35, Mar. 2020. DOI: 10.1111/mice.12546.

[17] W. Van Hauwermeiren, K. Filipan, D. Botteldooren, and B. De Coensel, "Assessing road pavement quality based on opportunistic in-car sound and vibration monitoring," *26th International congress on Sound and Vibration*, 2019. [Online]. Available: https://biblio.ugent.be/publication/8630738/file/8631289 (visited on 03/11/2021).

[18] ——, "Opportunistic monitoring of pavements for noise labeling and mitigation with machine learning," *Transportation Research Part D: Transport and Environment*, vol. 90, p. 102 636, 2021. DOI: https://doi.org/10.1016/j.trd.2020.102636.

[19] J. Lekshmipathy, S. Velayudhan, and S. Mathew, "Influence of surface distresses on smartphone-based pavement roughness evaluation," *International Journal of Pavement Engineering*, pp. 1–14, Jan. 2020. DOI: 10.1080/10298436.2020.1714045.

[20] T. Lee, C. Chun, and S.-K. Ryu, "Detection of road-surface anomalies using a smartphone camera and accelerometer," *Sensors*, vol. 21, p. 561, Jan. 2021. DOI: 10.3390/s21020561.

[21] H. McGurk and J. MacDonald, "Hearing lips and seeing voices," *Nature*, vol. 264, pp. 746–8, Dec. 1976. DOI: 10.1038/264746a0.

[22] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Ng, "Multimodal deep learning," Jan. 2011, pp. 689–696.

[23] K. Noda, Y. Yamaguchi, K. Nakadai, H. Okuno, and T. Ogata, "Audio-visual speech recognition using deep learning," *Applied Intelligence*, vol. 42, Dec. 2014. DOI: 10.1007/s10489-014-0629-7.

[24] L. Ma, Z. Lu, L. Shang, and H. Li, "Multimodal convolutional neural networks for matching image and sentence," Dec. 2015, pp. 2623–2631. DOI: 10.1109/ICCV.2015.301.

[25] A. Eitel, J. Springenberg, L. Spinello, M. Riedmiller, and W. Burgard, "Multimodal deep learning for robust rgb-d object recognition," Sep. 2015, pp. 681–687. DOI: 10.1109/IROS.2015.7353446.

[26] X. Xu, Y. Li, G. Wu, and J. Luo, "Multi-modal deep feature learning for rgb-d object detection," *Pattern Recognition*, vol. 72, Jul. 2017. DOI: 10.1016/j.patcog.2017.07.026.

[27] V. Sindagi, Y. Zhou, and O. Tuzel, *Mvx-net: Multimodal voxelnet for 3d object detection*, Apr. 2019.

[28] J. Che, T. Qiao, Y. Yang, H. Zhang, and Y. Pang, "Longitudinal tear detection method of conveyor belt based on audio-visual fusion," *Measurement*, vol. 176, p. 109 152, 2021, ISSN: 0263-2241. DOI: https://doi.org/10.1016/j.measurement.2021.109152. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0263224121001767.

[29] C. Li, R. Sánchez, G. Zurita, M. Cerrada, D. Cabrera, and R. Vasquez, "Gearbox fault diagnosis based on deep random forest fusion of acoustic and vibratory signals," *Mechanical Systems and Signal Processing*, vol. 76, Feb. 2016. DOI: 10.1016/j.ymssp.2016.02.007.

[30] A. Diez-Olivan, J. Del Ser, D. Galar, and B. Sierra, "Data fusion and machine learning for industrial prognosis: Trends and perspectives towards industry 4.0," *Information Fusion*, vol. 50, Oct. 2018. DOI: 10.1016/j.inffus.2018.10.005.

[31] W. Jiang and Z. Yin, "Combining passive visual cameras and active imu sensors for persistent pedestrian tracking," *Journal of Visual Communication and Image Representation*, vol. 48, Mar. 2017. DOI: 10.1016/j.jvcir.2017.03.015.

[32] M. Brossard, A. Barrau, and S. Bonnabel, "Ai-imu dead-reckoning," vol. PP, pp. 1–1, Mar. 2020. DOI: 10.1109/TIV.2020.2980758.

[33] S. Liu, B. Yu, Y. Liu, K. Zhang, Y. Qiao, T. Li, J. Tang, and Y. Zhu, "The matter of time – a general and efficient system for precise sensor synchronization in robotic computing," Mar. 2021.

[34] D. Arya, H. Maeda, S. Ghosh, D. Toshniwal, H. Omata, T. Kashiyama, and Y. Sekimoto, "Global road damage detection: State-of-the-art solutions," Nov. 2020.

[35] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Nov. 2013. DOI: 10.1109/CVPR.2014.81.

[36] J. Uijlings, K. Sande, T. Gevers, and A. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, pp. 154–171, Sep. 2013. DOI: 10.1007/s11263-013-0620-5.

[37] R. Girshick, "Fast r-cnn," Apr. 2015. DOI: 10.1109/ICCV.2015.169.

[38] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, Jun. 2015. DOI: 10.1109/TPAMI.2016.2577031.

[39] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," Jun. 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[40] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," Jul. 2017, pp. 6517–6525. DOI: 10.1109/CVPR.2017.690.

[41] ——, "Yolov3: An incremental improvement," Apr. 2018.

[42] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," Jul. 2017, pp. 936–944. DOI: 10.1109/CVPR.2017.106.

[43] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, *Yolov4: Optimal speed and accuracy of object detection*, 2020. arXiv: 2004.10934 [cs.CV].

[44] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. Zitnick, "Microsoft coco: Common objects in context," vol. 8693, Apr. 2014, ISBN: 978-3-319-10601-4. DOI: 10.1007/978-3-319-10602-1_48.

[45] J. Cooley and J. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, pp. 297–301, Jan. 1965. DOI: 10.1090/S0025-5718-1965-0178586-1.

[46] J. Lekshmipathy, S. Velayudhan, and S. Mathew, "Effect of combining algorithms in smartphone based pothole detection," *International Journal of Pavement Research and Technology*, vol. 14, Jul. 2020. DOI: 10.1007/s42947-020-0033-0.

[47] J. Wolf, R. Guensler, and W. Bachman, "Elimination of the travel diary: Experiment to derive trip purpose from global positioning system travel data," *Transportation Research Record*, vol. 1768, pp. 125–134, Jan. 2001. DOI: 10.3141/1768-15.

[48] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets," *PloS one*, vol. 10, e0118432, Mar. 2015. DOI: 10.1371/journal.pone.0118432.

[49] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–338, Jun. 2010. DOI: 10.1007/s11263-009-0275-4.

[50] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[51] S. Smith, "The scientist and engineers guide to digital signal processing," *California Technical Publishing*, pp. 551–566, Jan. 1997.

[52] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, Dec. 2014.

9.2 Other Sources

[53] Rijksoverheid. (1932). "Artikel 15 van de wegenwet," [Online]. Available: http://wetten.overheid.nl/jci1.3:c:BWBR0001948&hoofdstuk=IV&artikel=15 (visited on 05/15/2021).

[54] ——, (2020). "Artikel 162 van de wegenwet," [Online]. Available: https://wetten.overheid.nl/jci1.3:c:BWBR0005289&boek=6&titeldeel=3&afdeling=2&artikel=174&z=2018-09-19&g=2018-09-19 (visited on 05/15/2021).

[55] CROW. (2011). "Wegbeheersystematiek (publicatie 147)," [Online]. Available: https://www.crow.nl/thema-s/wegbeheer-en-wegonderhoud/crow-beheersystematieken (visited on 06/15/2021).

[56] Rijksoverheid. (2020), [Online]. Available: https://www.rijksoverheid.nl/ministeries/ministerie-van-infrastructuur-en-waterstaat/documenten/begrotingen/2020/09/15/a-infrastructuurfonds-rijksbegroting-2021 (visited on 06/23/2021).

[57] Rijksoverheid. (2019). "Besluit begroting en verantwoording provincies en gemeenten," [Online]. Available: https://wetten.overheid.nl/BWBR0014606/2019-07-01 (visited on 06/15/2021).

[58] S. in Residence. (2020). "Startup in residence overijssel," [Online]. Available: https://startupinresidence.com/overijssel/ (visited on 03/01/2021).

[59] bam. (2018). "Automatische asfaltschade herkenning met kunstmatige intelligentie," [Online]. Available: https://www.baminfra.nl/nieuws/bigger-data-in-asset-management-de-groeiende-mogelijkheden-van-automatische-schadeherkenning (visited on 10/05/2021).

[60] H. Maeda, Y. Sekimoto, T. Seto, T. Kashiyama, and H. Omata. (2018). "Road Damage Detector," [Online]. Available: https://github.com/sekilab/RoadDamageDetector/ (visited on 11/01/2021).

[61] V. Hedge, D. Trivedi, A. Alfarrarjeh, A. Deepak, S. H. Kim, and S. Cyrus. (2018). "Road Damage Detector Challenge 2020," [Online]. Available: https://github.com/USC-InfoLab/rddc2020 (visited on 11/01/2021).

[62] G. Jocher, K. Nishimura, T. Mineeva, and R. Vilariño. (2021), [Online]. Available: https://github.com/ultralytics/yolov5 (visited on 05/01/2021).

[63] European Parliament. (2002). "Directive 2002/49/EC," [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32002L0049 (visited on 03/11/2021).

[64] M+P. (2020). "CPX measurements," [Online]. Available: https://mp.nl/en/solution/cpx-measurements (visited on 03/11/2021).

[65] S. Dulepet, P. Maji, M. Harsh, and K. Burke. (2020). "Deploying a scalable object detection inference pipeline," [Online]. Available: https://developer.nvidia.com/blog/deploying-a-scalable-object-detection-inference-pipeline/ (visited on 04/26/2021).

[66] AutoPi. (2021). "Presenting the AutoPi Telematics Unit (TMU) Pi3 3rd Generation device," [Online]. Available: https://www.autopi.io/hardware-dongle/generation-three/ (visited on 03/15/2021).

[67] European Parliament. (1998). "Directive 98/69/EC," [Online]. Available: https://eur-lex.europa.eu/legal-content/en/ALL/?uri=CELEX:31998L0069 (visited on 03/15/2021).

[68] Joël Luijmes. (2021). "Application for camera and sensor data logging (ios) - modified version," [Online]. Available: https://github.com/joelluijmes/ios_logger (visited on 10/04/2021).

[69] Varvar. (2020). "Application for camera and sensor data logging (ios)," [Online]. Available: https://github.com/Varvrar/ios_logger (visited on 10/03/2021).

[70] Prefect. (2021). "The easiest way to build, run, and monitor data pipelines at scale," [Online]. Available: https://www.prefect.io (visited on 06/01/2021).

[71] National Geospatial-Intelligence Agency. (2014). "World Geodetic System 1984 (WGS 84)," [Online]. Available: https://earth-info.nga.mil/index.php?dir=wgs84&action=wgs84 (visited on 11/20/2021).

[72] Bing Maps. (2018), [Online]. Available: https://docs.microsoft.com/en-us/bingmaps/rest-services/routes/snap-points-to-roads (visited on 07/07/2021).

[73] Kadaster. (2018), [Online]. Available: https://www.kadaster.nl/zakelijk/registraties/basisregistraties/bgt (visited on 07/07/2021).

[74] SciPy. (2001). "Fundamental algorithms for scientific computing in python," [Online]. Available: https://scipy.org (visited on 11/20/2021).

[75] E. Poeze and S. Liem. (2019). "Aligning the coordinate systems of accelerometer and vehicle," [Online]. Available: https://viriciti.com/blog/automatic-datahub-orientation/ (visited on 09/04/2021).

[76] Jur van den Berg. (2013). "Application for camera and sensor data logging (ios) - modified version," [Online]. Available: https://math.stackexchange.com/questions/180418/calculate-rotation-matrix-to-align-vector-a-to-vector-b-in-3d (visited on 09/04/2021).

[77] FFmpeg. (2021). "A complete, cross-platform solution to record, convert and stream audio and video.," [Online]. Available: https://ffmpeg.org (visited on 10/04/2021).

[78] OpenVINO. (2021). "Powerful and efficient computer vision annotation tool," [Online]. Available: https://github.com/openvinotoolkit/cvat (visited on 10/13/2021).

[79] Weights & Biases. (2021). "A tool for visualizing and tracking your machine learning experiments," [Online]. Available: https://wandb.ai.

[80] scikit-learn. (2007). "scikit-learn: machine learning in Python," [Online]. Available: https://scikit-learn.org/stable/ (visited on 11/24/2021).

[81] J. Glenn. (2021). "Tips for Best Training Results," [Online]. Available: https://github.com/ultralytics/yolov5/wiki/Tips-for-Best-Training-Results (visited on 11/22/2021).

[82] ——, (2021). "TFLite, ONNX, CoreML Export," [Online]. Available: https://github.com/ultralytics/yolov5/issues/251 (visited on 11/29/2021).