FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Road Quality Classification
Student:	Martin Lank
Supervisor:	Ing. Magda Friedjungová, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2021/2022

Instructions

Many municipalities and road authorities seek to implement automated evaluation of road damage. However, knowing the degree of road damage can also be useful while planning your journey, especially when you ride a motorbike or an older type of car and want to plan a journey using high-quality roads. To do so, we need to overcome several struggles starting with data collection and ending with the modelling of a quality classification.

1) Review and theoretically describe the state of the art approaches to road quality classification, including publicly available datasets and preprocessing steps.
2) If necessary, collect and annotate data to solve road quality classification with a focus on driving pleasure.
3) Use or implement at least two of the reviewed approaches and experimentally compare their performance on a suitable dataset.
4) Propose a direction for further improvement in the domain of road quality classification.

Electronically approved by Ing. Karel Klouda, Ph.D. on 4 December 2020 in Prague.

Czech Technical University in Prague

Faculty of Information Technology

Department of Applied Mathematics

Bachelor's thesis

Road Quality Classification

Martin Lank

Supervisor: Ing. Magda Friedjungová, Ph.D.

May 13, 2021

Acknowledgements

Firstly, I would like to extend my deepest gratitude to my thesis supervisor, Ing. Magda Friedjungová, Ph.D., for her profound guidance, source of constructive advice and support in all phases of the work. It was a pleasure to work under your leadership.

I also wish to thank my family for their great support and patience during my studies. Special thanks go to my brother, Ing. Vojtěch Lank, who provided me with stable computational resources, which significantly eased the training of the models in this work.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60(1) of the Act.

In Prague on May 13, 2021 .....

Czech Technical University in Prague

Faculty of Information Technology

Citation of this thesis

LANK, Martin. *Road Quality Classification.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021. Available also from WWW: ⟨`https://github.com/lenoch0d/road-quality-classification`⟩.

Abstract

Automated evaluation of road quality can be helpful to authorities and also road users who seek high-quality roads to maximize their driving pleasure. This thesis proposes a model which classifies road images into five qualitative categories based on overall appearance. We present a new manually annotated dataset, collected from Google Street View. The dataset classes were designed for motorcyclists, but they are also applicable to other road users. We experimented with Convolutions Neural Networks, involving custom architectures and pretrained networks, such as MobileNet or DenseNet. Also, many experiments with preprocessing methods such as shadow removal or Contrast Limited Adaptive Histogram Equalization (CLAHE). Our proposed classification model uses CLAHE and achieves 71% accuracy on a test set. A visual check showed the model is applicable for its designed purpose despite the modest accuracy since the image data are often controversial and hard to label even for humans.

Keywords road quality classification, road surface analysis, street view images, road quality image dataset, image classification, convolutional neural networks

Abstrakt

Automatické vyhodnocování kvality vozovky může být užitečné jak správním orgánům, tak i těm účastníkům silničního provozu, kteří vyhledávají vozovky s kvalitním povrchem pro co největší potěšení z jízdy. Tato práce se zabývá návrhem modelu, který klasifikuje obrázky silnic do pěti kvalitativních kategorií na základě jejich celkového vzhledu. V práci prezentujeme nový ručně anotovaný dataset, obsahující fotografie ze služby Google Street View. Anotace datasetu byla navržena pro motorkáře, ale může být použita i pro jiné účastníky silničního provozu. Experimentovali jsme jak s předučenými konvolučními neuronovými sítěmi, jako jsou MobileNet či DenseNet, tak s vlastními architekturami konvolučních neuronových sítí. Dále jsme vyzkoušeli různé techniky předzpracování dat, např. odstraňování stínů či kontrastně-limitní adaptabilní histogramovou ekvalizací (CLAHE). Námi navrhovaný klasifikační model využívá CLAHE a na testovací sadě dosahuje 71% přesnosti. Vizuální kontrola ukázala, že navrhovaný model je i s touto přesností využitelný za účelem, pro který byl navržen.

Klíčová slova klasifikace kvality vozovky, analýza povrchu vozovky, street view snímky, dataset snímků kvality vozovky, konvoluční neuronové sítě

Contents

List of Tables

List of Figures

Introduction

Roads are widely used every day all around the world. They serve us as a mean to move between locations. That may have many purposes, e.g. going to work or on holidays, visiting family, or transporting cargo and supplies. Every time we travel, we tend to optimise the route to get the best directions based on our parameters. The most common parameter is likely to be the time, which we try to minimise. If that were not the case, speeding would not be the top moving violation [1]. Other typical criteria are fluency, absolute distance, or low traffic. Considering these parameters, we try to choose the most suitable route. Either by our knowledge or by using modern navigation software.

However, this navigation software is not suitable for everyone. Certain groups of people have very specific criteria, as they want to use the medium for fun and pleasure, not for transportation purposes. A journey with a vintage car or motorbike is an excellent example of that. Nevertheless, planning such a journey is not an easy task. The typical main optimisation parameter is road quality, for which the least damage is required. Although we can find such roads quite easily on highways and main roads, it goes against the second main parameter – playfulness. That includes low traffic and winding roads, which is quite the opposite of how the highway and main roads look. documentation for sources usage

Satisfying both criteria is very difficult. It is a known fact that the lower traffic a road has, the less care it has. In other words, when damage on the road needs to be repaired, or the road needs to be entirely reconstructed, it takes much

more time on the waitlist before doing so compared to more frequently used roads. It certainly makes sense – the government also optimises its resources to be used most effectively. Moreover, reconstruction of a road for several tens of cars a day rarely outweighs reconstruction of a highway for thousands of vehicles a day. Inevitably, more cars cause quicker degradation of road quality, contributing to the fact that playful high-quality roads with low traffic are harder to find.

Usually, it is a matter of trial and error when such a road is found. Besides, this approach is very time-consuming and with uncertain outcomes. In recent years, services like Google Street View[a] can be used to visually check the road and get an idea of what it looks. Even though this approach is considerably more effective than the previous one, it can easily take an hour to plan a hundred kilometres long journey. Using modern machine learning approaches to automate this visual check could streamline the process even more.

Machine learning is an application of artificial intelligence, where algorithms automatically learn from the data. These algorithms try to find general patterns in our data, so when we feed the algorithm with new data, it can quickly infer its properties. Machine learning principles have been evolving since 1959 [2], but due to the absence of enough powerful hardware, some areas were out of attention for a long time. Nowadays, with modern hardware, we can take full advantage of these algorithms and apply them in real-world applications, such as object detection or image classification.

Objectives

This thesis aims to review and describe the state-of-the-art methods in the question of road quality classification. Based on the findings and with regards to the described task, new data may be necessary to collect and annotate. Another objective is to develop a machine learning model that would use street view images as input and infer one of the predefined qualitative categories as an output.

[a] `https://www.google.com/intl/en/streetview`

Structure of the Thesis

The thesis is organised as follows. In Chapter 1, the task is described more in-depth, along with potential solutions and related works. Chapter 2 covers data preparation, including data collection and annotation. Chapter 3 describes the methodology of how experiments were run, evaluated and introduces implemented approaches. Chapter 4 presents and discusses the results of those implemented approaches. The final Chapter 4.2 reviews our contributions and proposes directions for future work.

Further reading assumes a decent knowledge of machine learning concepts. The fundamentals and underlying principles can be learnt from some other literature, such as [3].

Chapter 1

Road Quality

This chapter defines the problem, which this thesis focuses on more thoroughly. Next, it introduces the conducted survey findings to determine the state-of-the-art methods regarding automated road quality evaluation. It also describes publicly available datasets relevant to this task.

1.1 Problem Definition

As mentioned in the introduction, there is a vast range of uses for roads. Apart from being used out of necessity, they may be used for pleasure, such as driving a sports car, a vintage car or a motorcycle. We also stated a problem with searching high-quality roads satisfying given criteria. However, every use case might have different criteria to fulfil. Since the author is a passionate motorcycle rider and has many years of riding experience on roads, such a use case is mainly considered in this work.

One of the prerequisites for a comfortable and joyful motorcycle ride is a smooth road surface. Any damage such as crack, potholes, or incoherent surface caused by layered repairs can decrease both the levels of fun experienced and the safety of the ride. Automated evaluation of road quality can be done by classifying acceleration data [4, 5, 6], or images [7, 8]. However, from this task's point of view, using acceleration data lacks scalability. There is currently no public provider of such data on a similar scale like Google Street View or Baidu

Maps[a] for images. Besides, the effect of road defects on acceleration data heavily depends on speed and vehicle suspension. Also, the vehicle needs to go through the defects, which is something car owners typically want to avoid to not damage their car. Pictures taken while driving are significantly less affected by these problems, if at all. However, most image-based state-of-the-art methods for road quality evaluation focus on damage segmentation [9, 10, 11, 12, 13]. Therefore, this work focuses on classifying road pictures into several categories based on overall surface quality.

The typical image classification approach is to use Convolutional Neural Network (CNN) [14], a deep artificial neural network designed for image analysis. The neural net can be trained using supervised learning to determine the predefined categories. However, supervised learning, especially when working with images, usually requires large datasets with thousands of samples. If the dataset is relatively small and does not cover enough diversity in a given domain, it is common to perform data augmentation. That is usually done by applying geometrical transformations or colour adjustments. Data augmentation can also be used when a dataset is class imbalanced, i.e., some categories contain significantly more data than others [15]. Sometimes further preprocessing steps are performed before feeding the data into the net, mainly to increase the performance. Such steps can be downscaling, extracting a specific Region of Interest (ROI) by cropping the image or converting colour images into grayscale images.

1.2 Related Work

In this section, the findings of the performed survey are described. As mentioned above, there are several types of works in the question of automated road quality evaluation. Some make use of accelerometer data [5, 6, 4] and others of images [8, 7]. Also, not all approaches solve a classification problem, where a category is assigned to a road based on the overall surface quality, but they focus on damage detection, where every corruption such as a pothole or a crack is detected. For each work, a brief overview of the task, dataset and approach is provided along with the achieved results.

[a] `https://map.baidu.com`

1.2.1 Classification

The authors of [5] aimed to classify road surface by quality into two categories, *smooth* and *damaged*, based on accelerometer data. Dataset was created by the authors, who collected the data using a smartphone attached to a motorbike with the use of Phypox application[a]. They proposed a model based on Support vector machine (SVM) and achieved 93% accuracy.

A similar approach was used in [6], where authors aimed to classify roads into three categories – *smooth*, *rough* and *bumpy*. They also collected data with a smartphone mounted to a bicycle. Proposed models were based on the K-Nearest-Neighbour (KNN) algorithm and the Naive Bayes Classifier. Both performed very similarly, with an overall accuracy of around 77.5%. However, accuracy was only pulled up by *smooth* class with an accuracy of 85.6%. *Rough* and *bumpy* categories achieved an accuracy of 54% and 65%. Their second approach was to detect and count bumps and classify the surface accordingly, but the results were even worse.

Another work where accelerometer data is used was published by Tiwari et al. [4]. Their objective was to classify roads by quality into *good, medium* and *bad*. Data for this work were collected via a smartphone app placed in a car and a dashboard cam looking towards the road. Its recordings were later used to label data obtained from the app. While their SVM approach performed worse than previously mentioned SVM models, their new deep learning approach using CNN significantly outperformed all previous models with 97% accuracy with *good* roads and 99% accuracy with both *medium* and *bad* roads.

In [8] the objective was to use images and CNN to classify road surface by type. They defined six classes: *asphalt, dirt, grass, wet asphalt, cobblestone, snow*. To cover all classes with enough variety, the authors combined several publicly available datasets (KITTI [16], RobotCar [17], NREC [18], New College [19], Stadtpilot (private), Giusti et al. [20]). Interesting is that Google search of similar pictures was used to extend some categories with low samples count. Authors claim they occurred serious overfitting issues on the dataset without augmentation. Therefore random horizontal flip, rotation, scaling and smoothing was applied. In regards to modelling, a transfer learning [21]

[a] A smartphone app for physical experiments allowing data collection from numerous phone sensors. `https://phyphox.org`

approach with pre-trained CNNs (ResNet50 [22] and InceptionV3 [23]) was used. The best model was based on ResNet50 and outperformed the other by 2%, with an average accuracy of 92%. The accuracies for each class were: 99% (*asphalt*), 82% (*dirt*), 100% (*grass*), 81% (*wet*), 91% (*cobblestone*), 100% (*snow*).

In [7] published by Rateke et al. the road type and quality classification were done using images taken by a low-cost camera (about 100 USD). In addition to two existing datasets (KITTI [16] and CaRINA [24]) used in this work, a new "Road Traversing Knowledge" dataset was created by taking frames from a captured video. The final dataset consists of 6,297 annotated images. Categories were assigned as follows: *asphalt, paved* and *unpaved. Asphalt* and *paved* are further subcategorised by their surface quality into *good, regular* and *bad*, whereas *unpaved* only into *regular* and *bad*.

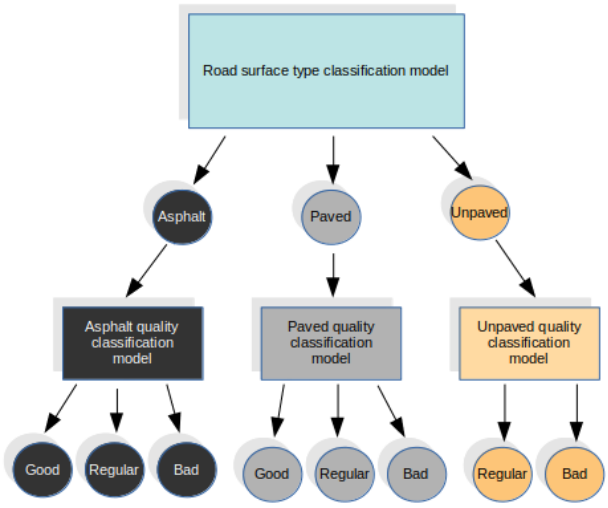

Figure 1.1: The picture depicts the structure of an ensemble classification model used in work by Rateke et al. to first classify roads by surface type and then into a qualitative category. Source: [7]

The authors present an ensemble model consisting of two layers and four models in total, as shown in Figure 1.1. Each model is a CNN. The top layered model was trained to determine the road type. The second-layered models are trained to determine the surface quality for each road type individually. Before training, images were preprocessed. The authors defined a ROI to extract the most relevant information from the image – the road, which mostly appears

only in the specific part of their images. Additionally, data augmentation was performed to compensate for lighting changes by applying random brightness adjustments. Using this approach, the authors were able to achieve admirable results. The top layered model learned on their dataset performed with an accuracy of 95.73% on the test set (98.58% accuracy for the *asphalt* category, 86.24% accuracy for *paved* category and 93.67% for *unpaved* category). The accuracy of the asphalt quality classification was 98.23%, 94.04% and 96.77% for *good, regular* and *bad*.

In work published by Ma et al. [25], street view images were classified by road surface quality into three categories: *good, fair, poor*. The authors created their own large-scale dataset with about 700,000 samples by matching government road inspection data and images from Google Street View. However, there was a 1.2 years time difference between taking the road images the government inspection. Therefore the assigned categories might be inaccurate. Additionally, the dataset was heavily class imbalanced. The authors utilised Fisher-Vector CNN [26], a CNN that extracts features from a given image and encodes them using Fisher Vector. Specifically, they used VGG-D [27] and 64x64 patches for the feature extraction. The descriptors were then normalised and classified using a Random Forest classifier. This approach performed with an accuracy of 72.2% (*poor*), 50.7% (*fair*) and 51.7% (*good*). Data augmentation was not used, and road segmentation was found unuseful as a preprocessing step.

1.2.2 Damage Detection

Thanks to Global Road Damage Detection Challenge[a], which took place in 2020, there are many recent solutions available regarding image-based damage detection. The objective was to find and classify individual defects on road images collected in India, Japan and Czech. The dataset was introduced in [28]. It consists of 26,620 images and 31,343 labels in four categories: *longitudinal crack, transverse crack, alligator crack, pothole*. For the purposes of the challenge, only 21,041 labelled images were released. The rest was used for two testing sets, *test1* and *test2*. Along with the dataset, the researchers proposed a baseline model based on transfer learning using MobileNet [29]. However, the F1-Scores were considerably low. The top challenge contributors were able to outperform the baseline model significantly.

[a] `https://rdd2020.sekilab.global`, `https://github.com/sekilab/RoadDamageDetector`

The best model was proposed by the IMSC team (Hedge et al. [11]). Their proposed solution used Ensemble Learning with Ultralytics-YOLO and Test Time Augmentation (TTA) [30]. TTA is a technique in which input images are transformed, and their predictions are then aggregated, which may increase performance [31]. However, IMSC was the only team using TTA in the final solution. Other teams did not consider it particularly useful. The top team also used classic data augmentation performed before training. Specifically, they used Python Augmenter[a] to generate synthesised images by applying sharpening, additive Gaussian noise and affine texture mapping. With this approach, the top team achieved a mean F1-score of 67.48% on *test1* and 66.62% on *test2*.

The team placed second (Doshi et al. [9]) also proposed an ensemble model, but with YOLO-V4[b] as the base model. As for data augmentation, the authors used a random cropping algorithm with the output of three images next to the original. Their approach achieved 62.75%, and 63.58% mean F1-scores on *test1* and *test2*, respectively.

The team that secured third place (Pei et al. [10]) used Cascade Region Based Convolutional Neural Network (R-CNN) with proposed Consistency Filtering Mechanism. Also, several interesting augmentation techniques were used, such as mixup, Contrast Limited Adaptive Histogram Equalization (CLAHE) [32] and road segmentation. Achieved scores were only slightly worse than in [9] by the team placed second, 62.9% on *test1* and 62.19% on *test2*.

All the other teams participating in this challenge achieved scores of less than 60%. Many of them built models on Faster-R-CNN [33] or ResNet networks and widespread was the use of data augmentation including random geometric transformations (scale, rotation, flipping, translation, resizing) and random colour adjustments (contrast, hue, saturation, brightness). However, some teams did not use any augmentation and still outperformed many other teams.

In work conducted by Chacra et al. [12], the researchers aimed to detect defects in street view images using texture descriptors and contour maps [12].

[a] `https://github.com/mdbloice/Augmentor`
[b] `https://github.com/Tianxiaomo/pytorch-YOLOv4`

Firstly, the authors needed to create a new dataset as there were not any publicly available. That was done by collecting images from Google Street View using their Application Programming Interface (API). Data were then manually annotated. Unfortunately, the authors do not provide more information about the dataset, such as the size or number of annotated defects. Nevertheless, their approach was as follows. Firstly, they segmented the road from a given image. Then the feature patches were extracted using Scale-invariant feature transform (SIFT) [34] and the descriptors were encoded and classified using an SVM into *good* or *damaged* class. Every patch is given either a positive or a negative weight after classification. Further processing then generated a probability map of damaged areas. Lastly, an ultrametric contour map [35] was used to precisely locate the defects in the damaged area. According to the authors, they were able to achieve a mean F1-score of 92.84%.

Lei et al. [13] published a work where the objective was to detect pavement distress in an image and track its deterioration over time. Baidu Street View enables users to view images back in history at a given location (if history exists). The authors utilised this service in order to create a dataset with about 20,000 images and labelled them into eight categories: *deformation, pothole, loose, net-crack, cracks, patched-pothole, patched-net, patch-crack*. Before training, data augmentation was performed with random rotation and Gaussian blur. To detect the defects in the images, the object detection model YOLO-V3[a] based on CNN was applied. Next, to analyse the deterioration over time, historical images were retrieved. Then, features were extracted from the images using SIFT and matched with pictures at the same location. Lastly, a decision tree was designed for evaluating the deterioration over time. To conclude, the proposed model performed with accuracy between 87-89%.

1.3 Available Datasets

This section summarizes publicly available image datasets that could be potentially useful for the task of this thesis. Table 1.1 shows datasets with a task it was designed for and classification classes in the case of a classification task. It can be seen that the majority of the datasets are not designed for classification purposes.

[a] `https://github.com/ultralytics/yolov3`

Table 1.1: This table summarizes the publicly available road image datasets. The table shows a task for which the given dataset was designed. In the case of a classification task, the annotated classes are present. Sample count is also present for annotated datasets.

Dataset	Task	Classes	Samples
RTK [7]	type + quality classification	good, regular, bad	6,297
Paris-Saclay[25]	quality classification	good, fair, poor	∼700,000
KITTI [16]	autonomous driving		
CaRINA [24]	surface detection	easy, medium, difficult	900
RobotCar [17]	autonomous driving		
GRDDC 2020 [28]	damage segmentation		26,620
DIPLODOC [36]	road segmentation		865

KITTI dataset (collected in Karlsruhe, Germany) was created to develop computer vision challenges, including visual odometry, three-dimensional object detection and tracking. The dataset contains information about location, accelerometer data, laser data and camera images. In terms of roads, the dataset does not provide many variations in its quality nor annotated labels. It has some lighting variations in the images (shadows, direct sun).

RobotCar dataset (Oxford, UK) was created in order to extend the KITTI dataset with a more variety of lighting conditions (overcast, night, rain, snow) and road types. Therefore it contains data of a similar kind.

GRDDC dataset (India, Japan, Czech) was created for the Global Road Damage Detection Challenge. The images in the datasets were taken by a smartphone attached to a front car window. The dataset contains annotations of individual defects in four categories.

DIPLODOC dataset (Trento, Italy) was collected by a stereo cam in a car. The images are labelled for the use of road segmentation. The annotated areas are defined as "everywhere a car could drive without going up a step" [36]. That means when e.g. a car is in the picture, it is not excluded, and the area is

annotated over it. Those unwanted objects are annotated as well. Therefore, the visible road can be computed, if necessary.

The first considered dataset intended for road quality classification is CaRINA (São Carlos, Brazil). It contains data collected by an electronic radar[a] along with camera images. The data are labelled into three categories: *easy, medium, difficult*, where the *easy* is the smoothest, and the *difficult* contains potholes and other defects. Although it was collected with a focus on urban areas, the variety of road surfaces is considerable. It contains not only asphalt surfaces but also paved, unpaved and other surfaces. Apart from some shadows, the dataset does not contain diverse lighting conditions, as it was collected on a sunny day.

The next dataset was introduced by Ma et al. at the Paris-Saclay University in France. It was created by matching data collected by government inspectors and Google Street View images, resulting in a large scale dataset with about 700,000 images. The government data came from the New York City Department of Transportation. Based on the government road rating, ranging from 1 to 10, the Google Street View images were obtained using Street View API[b] and automatically labelled into three categories: *good, fair, poor*. The images have a wide view and contain many unrelated objects such as parked cars on the sides, trees or pavements. Also, the dataset is very imbalanced – only 0.6% of images are in the *poor* class and 28.2% in the *fair* class [25]. There was also a 1.2 years gap between the collection of government data and Google Street View images. Therefore, the images might be labelled falsely.

Lastly, the RTK dataset (Santa Catarina, Brazil) consists of 6,297 images collected with a dashboard cam in a car. Images were manually annotated with a surface category *asphalt, paved, unpaved*, and a qualitative category *good, regular, bad* (*unpaved* has only *regular, bad* labels). The dataset contains a wide range of road surfaces but lacks lighting variety (e.g. no tree shadows in *bad* category). It is also less imbalanced than the Paris-Saclay dataset (with about 19% of *bad* and 50% of *good*), but the imbalance is still noticeable.

[a] `https://roboticsknowledgebase.com/wiki/sensing/delphi-esr-radar`
[b] `https://developers.google.com/maps/documentation/streetview/overview`

1.4 Suitable Dataset

In this section, the ideal dataset for the task of this thesis is defined, and the suitability of publicly available datasets is discussed.

This thesis aims to create a classification model, which would automate visual road inspection with the focus on riding for pleasure on a motorbike. The manual visual check can nowadays be done using street view images which are available online. Motorcyclists typically want the road to be as smooth as possible. With increasing damage, the road changes from "smooth" over "rideable" to "dangerous". Some road types are inadmissible, such as dirt road (assuming road tyres). It should be pointed out that even one pothole or crack on an overall smooth road can make it dangerous. Therefore, the classification should be based on overall road quality.

With respect to mentioned riders' road perception and riding experience, the author proposes the following categories:

1. Top-quality. Smooth surface without any damage. The rider can fully enjoy the ride.
2. Overall smooth surface with tiny longitudinal cracks and/or repairs not affecting the safety and riding pleasure.
3. Roads with patches, moderate linear cracks in any directions. Riders must pay some attention to the road.
4. Bumpy roads with potholes or multiple (layered) repairs. Riders must pay very close attention to the road, and the fun factor is gone.
5. Stone paved roads or unpaved roads. Extremely dangerous for motorcyclists, especially in the rain. Only plausible in an emergency with extra caution.

In addition, the dataset should cover various lighting conditions (shadow, brightness, dark, light), various weather conditions (dry and wet surface) and various road types (asphalt, concrete). Next, it should cover roads with and without surface markings, and it should not contain unwanted objects, such as buildings, cars, trees or traffic signs. Also, it should be invariant to seasonal impurities, such as mud or grime.

It is evident that none of the available datasets can be directly used for this task without changes. The annotated datasets discretise quality into three classes, which is insufficient. Hence, the author suggests five categories. Also, the Pair-Saclay dataset might have misleading labels and only focuses on urban roads, which riders typically want to avoid due to higher traffic. Class imbalance is also a problem for RTK and Pair-Saclay dataset. In [7] the authors showed that combining RTK dataset with CaRINA and KITTI did not lead to increased performance on RTK test set, but did lead to an increase in performance on the CaRINA and KITTI test set. To sum up, all available datasets would have to be re-annotated and combined or extended with more data. However, as shown in [7] that might not necessarily increase the performance.

Considering the use case of a model this work aims to propose and the shortcomings in the available datasets, we conclude that it is best to collect and annotate a new dataset.

1.5 Theoretical Background

Convolutional Neural Networks are deep neural networks designed for processing images. The idea behind CNN is that the neural network tries to learn features of an input image represented as a matrix and gives those features higher-order meaning in deeper layers. For example, if a CNN learned to recognise a face, it could first learn to detect eyes, nose, and mouth. Then, if lower layers identified all these features, deeper layers could learn to conclude that there is a face in the picture. That is achieved by architecture with layers of three types – convolutional, pooling and fully connected.

The following text describing CNN layers is inspired by the Deep Learning book written by Goodfellow et al. [14].

1.5.1 Convolutional Layer

The first type layer is called convolutional. Its purpose is to extract features from the input image with the use of a convolutional filter – a matrix containing weights. Its dimensionality depends on the number of channels in an image. The width and the height of the matrix are defined by hyper-parameters, and the weights are trainable parameters. The process of feature extraction is called convolution, and its result is called a convolved feature, which we get by

calculating a dot product of the filter and a subset of the input matrix. A sliding window is used in convolutional layers to get a subset of the input matrix, matching the filter dimensions. The stride of the slide is also defined via hyper-parameter. Generally, it slides from the top left corner of the input matrix to the right. Once it reaches the right edge, the window moves down by one row and continues from the left until it reaches the bottom right corner. All the convolved features form an output matrix, often called a feature map.

Let's consider a grayscale image as an input. Mathematically, it is a matrix $\mathbf{I} \in \mathbb{R}^{i,j}$. Filter is a matrix $\mathbf{F} \in \mathbb{R}^{k,l}$ and a stride is a scalar $s \in \mathbb{N}$. Let $\mathbf{O} \in \mathbb{R}^{m,n}$ be the output matrix (feature map) containig convolved features, where $m = \frac{i-k}{s} + 1$ and $n = \frac{j-l}{s} + 1$. Its elements are defined as:

$$\mathbf{O}_{p,q} = \sum_{u=1}^{k} \sum_{v=1}^{l} \mathbf{F}_{u,v} \, \mathbf{I}_{s(p-1)+u, s(q-1)+v} \tag{1.1}$$

Figure 1.2 visualises a 2D convolution step on a subset of input matrix **I** using a sliding a window and a 3x3 filter **F**. The convolved feature calculated by a dot product of the filter, and the subset is stored in the output matrix **O**.

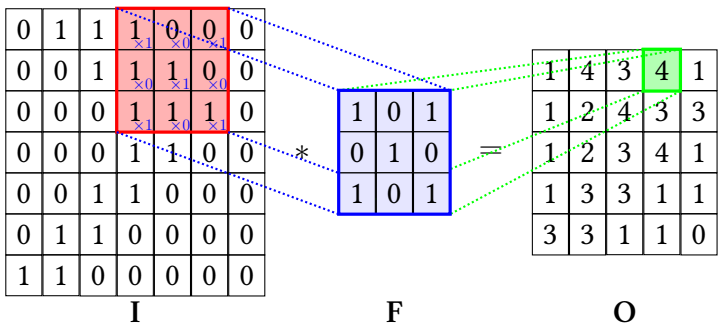

Figure 1.2: Visualisation of a single 2D convolution step on a subset of input matrix **I** using a sliding window and a 3x3 filter **F**. The convolved features calculated by a dot product of the filter, and the subset is stored in the output matrix **O**. Image source: `https://github.com/PetarV-/TikZ/tree/master/2D%20Convolution`

Padding can be applied to the input matrix if a given combination of a filter and a stride would not allow the sliding window to slide. Usually, zeros are used to extend the matrix.

In the end, often, a non-linear function is applied to the features to increase the non-linearity. Generally, rectified linear unit (ReLU)[a] or hyperbolic tangent is used.

1.5.2 Pooling Layer

In practice, it is common to use multiple convolutional layers in a row to extract more features. Consequently, it generates multiple feature maps. However, the data representation becomes spatial and network computationally more expensive. A pooling layer is used to reduce the spatial size and the number of parameters in the network. Also, it summarises the features, making the model more robust to variations.

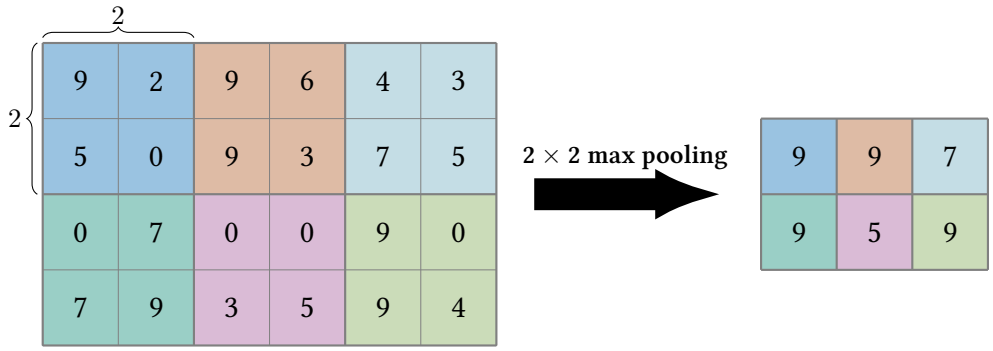

Figure 1.3: An example of the feature map Max Pooling method using 2x2 window and a stride of two. In each window, a maximum value is picked and saved in the output matrix. We can see that the transformed feature map is four times smaller. Image source: `https://github.com/MartinThoma/LaTeX-examples/tree/master/tikz/max-pooling`

Pooling is performed on each input feature map independently. The idea of pooling is that a sliding window $\mathbf{W} \in \mathbb{R}^{m,n}$ with stride $\mathbf{s} \in \mathbb{N}$ is used to downsample the features by mapping values within the patch into a scalar. In the case of a grayscale image, a transformation function $f\colon R \times R \to R$ is applied on every window. The two most common functions are called Max Pooling and Average Pooling. Max Pooling selects the maximum value within the window, whereas Average Pooling calculates the average. Sometimes Global Pooling is used, which is a special case of pooling, where the sliding window size is equal to the feature map size. Therefore, Global Pooling

[a] `https://www.cs.toronto.edu/~hinton/absps/reluICML.pdf`

downsamples the feature map to a single value. Figure 1.3 shows an example of Max Pooling using $\mathbf{W} \in \mathbb{R}^{2,2}$ and a stride of two.

1.5.3 Fully Connected Layer

Fully connected (FC) layer is an equivalent of a hidden layer in basic neural networks. In the case of image classification, there are generally two FC layers placed at the very end of a CNN. They give the network an ability to understand complex high-level features, their relationships and to perform classification.

Firstly, the previous layers' output is flattened to a one-dimensional vector. Then, all units from the vector are connected to all units in a FC layer, where the number of units is set via hyper-parameter. The second FC layer is connected on top of the previous one, whose number of units is typically equal to a classification class count. An activation function is utilised on all output nodes to get predictions for each class. Commonly used activation functions are softmax and ReLU.

1.5.4 Optional Layers

Several other layers may be utilised in the architecture to increase the overall network's performance. They might not always help, but it is generally good to try. Two widely used are mentioned below.

The batch normalisation layer transforms the inputs in a way that the output always has the given mean and standard deviation. The mean and standard deviation values to which this layer normalises are specified by trainable parameters. It aims to stabilise the learning and reduce the number of epochs needed to converge [37].

The dropout layer randomly deactivates input units with a specified rate. It was proven it helps to reduce overfitting issues [38].

1.5.5 Transfer Learning

Training a deep CNN from scratch is computationally very expensive and requires a lot of data. Transfer learning [21] is an approach, which can save time, resources and help to deal with small datasets. The general idea is to take advantage of a model learnt on a diverse large-scale dataset and customise it for a different task.

In the context of CNN, such pre-trained models are sufficient in extracting meaningful features on new datasets. Therefore, we do not have to learn the feature extractors (parameters in convolutional and pooling layers) from scratch. Instead, we use the pre-trained feature extractors and only add and learn FC layers on top to adapt the network to the given task.

One of the biggest publicly available large-scale and diverse dataset is ImageNet[a]. It contains over 1.7 millions labelled images in a thousand object classes. Thanks to an associated classification challenge, there are many publicly available pre-trained models on this dataset[b], such as MobileNet [29], ResNet [22] or VGG [27].

[a] http://www.image-net.org/index

[b] https://paperswithcode.com/sota/image-classification-on-imagenet

Chapter 2

Data Preparation

In the previous chapter, we defined the ideal dataset for this work. Moreover, we learned that no publicly available dataset matches our dataset requirements. Therefore, it was concluded a new dataset would be created. This chapter describes this process from data collection to labelling using a simple web application. Additionally, it covers several data balancing and preprocessing techniques utilised before training, such as shadow removal.

2.1 Data Collection

There are two possible ways to collect images for a new dataset. We can use a dashboard cam to record a video and then extract frames from it, which was done for most datasets in Section 1.3. Alternatively, we can utilise public services like Google Street View to obtain the images.

The first approach is highly time and resources consuming compared to the second one. We would have to drive thousands of kilometres to collect equally diverse data. From an economic point of view, it is not wise to repeat someone else's work, i.e. collect street view images from locations someone else already did and publicly published it[a]. With the motorcyclist path-finding use case in mind, we decided to utilise mentioned public service as the image source.

There are several public services providing street view images. Along with the previously mentioned Google Street View, it can be Baidu Maps or Mapy.cz[b].

[a] Assuming the published images are of sufficient quality.
[b] `http://en.mapy.cz`

We decided to use Google Street View for its large Europe and USA coverage and APIs[a] possibilities.

2.1.1 Related Google APIs Overview

Google Maps platform[b] provides several APIs, which can be utilised by developers based on their needs. The typical use is through a standard HTTPS request with query parameters. It should be noted that most of the APIs require an API key connected to an active billing account, so the user can be billed after exceeding a free quota[c].

One of the APIs is Street View Static API, which can be used to obtain a street view picture. Location (latitude and longitude) of the image origin and the output image size in pixels are specified via parameter. Optionally, a field of view and pitch can be specified. Since the original pictures were taken with a 360° camera, we can also specify a camera compass heading (e.g. 0 points to North, 180 to South). Additionally, we can obtain image metadata, such as the date the photo was taken, through a different request. If there is no imagery available for given parameters, we get a response with a generic image[d].

Another API is the Directions API[e], allowing users to plan a route based on start address, end address and optional waypoints. The API response contains information about the optimal planned route based on the current traffic. Apart from textual instructions or distance, it contains an encoded[f] so-called overview polyline. This polyline is a set of coordinates forming a smoothed path of the resulting directions.

The above mentioned APIs are utilised in a custom framework described in the next section.

2.1.2 Road Image Collection Framework

To systematically collect diverse road images without duplicates, a Road Image Collection Framework was developed. It takes advantage of described Street View and Directions APIs and provides a convenient way to define routes along

[a] `http://developers.google.com/maps/documentation/streetview/overview`
[b] `http://developers.google.com/maps`
[c] `http://developers.google.com/maps/documentation/streetview/usage-and-billing`
[d] `http://developers.google.com/maps/documentation/streetview/overview#no-image`
[e] `http://developers.google.com/maps/documentation/directions/overview`
[f] `http://developers.google.com/maps/documentation/utilities/polylinealgorithm`

Table 2.1: An example of route definitions in a CSV file viewed as a table.

Start address	End address	Name	Meters between points
383 01 Prachatice	Kvilda	prachatice-kvilda	100
507 82 Pecka	Třebihošť	pecka-trebihost	100
Leontýn, 270 23 Roztoky	330 05 Dobříč	leontyn-dobric	100
208, Krásno	356 01 Březová	krasno-brezova	100

which road images will be collected. Visualization of the predefined routes helps to prevent duplicate images (described more in Subsection 2.1.2.2).

2.1.2.1 Routes Definition

The framework allows user to set routes via a CSV file. It contains four columns: *start address, end address, name* and *meters between points*. The framework utilises the Directions API to generate directions between the addresses[a]. The more precise address, the better, as an uncertain address can lead to the API's misinterpretation. The *name* is primarily for debugging purposes in the visualisation described below. The last column specifies a minimum distance between images to be collected[b] to avoid similar images. We can see an example of the road definitions in Table 2.1.

Each route was manually composed in Google Maps, and their start and end addresses copy-pasted to route definitions. Hence, we had an idea of what directions the API returns. Initially, the routes were based on the author's knowledge, focusing on rural second and third-class roads (i.e. less frequent curvy roads) and later picked specifically to cover enough diversity. The minimum distance between points was mainly set to 100 metres and exceptionally lower, e.g. on rare cobblestones routes.

[a] The address can be in any format; it only must not contain the CSV delimiter – a semicolon.

[b] As mentioned in the Directions API overview, it responds with a smoothed polyline. The coordinates in the encoded polyline are not equally distributed (e.g. every ten meters). The coordinates are only present when necessary to keep a good approximation of some curvature on the road. Therefore, there might be only a few points on a kilometre-long straight segment of a road (at the start and the end). In contrast, there might be points every ten meters on a curvy segment to sufficiently approximate the curvatures. For this reason, the parameter only specifies a minimum distance.

2.1.2.2 Route Visualisation

An interactive map can be generated showing paths obtained via the Directions API for the predefined routes. It is an essential tool in preventing overlapping segments leading to duplicate images. Sometimes, the API generates a wrong path due to a vaguely defined address. Thanks to the visualisation, we can quickly identify the issues and adjust the route definitions to fix them.

Two example visualisations of routes defined for this work can be seen in appendixes B.1 and B.2. The first shows all routes defined in this work, while the second one shows a detailed view of three routes. Each route has a marker at the start and at the end. Markers have a caption containing a letter (*A* represents the start, *B* represents the end), route *name* and route distance.

2.1.2.3 Route processing

When the routes were ready, we could further proceed in the data collection process by utilizing Street View Static API. One of our ideal dataset requirements was for pictures to contain minimum unwanted objects. That is something we can affect by setting proper parameters in the API request, including the field of view, pitch and heading.

The field of view and the pitch were empirically set after several trials to $40°$ and $-30°$, respectively. With these values, the images depict the road right ahead of a car and do not contain unwanted objects along the road – assuming the heading parameter is set in the car's moving direction. As the heading is relative to the compass heading, it cannot be a static value. Luckily, since we take coordinates from the route's polyline, we can calculate the bearing between the two following points using a Haversine formula[a] and use it to compute the compass heading[b] parameter in an API request.

Finally, the Street View API could be utilised to download the images and their metadata. The framework saves the images in a JPG format with a resolution of 640x480 under a unique identifier. The metadata of the images is stored in a CSV file. To wrap up, we defined 85 routes on 1115 kilometres resulting in 7,292 road images.

[a] `https://en.wikipedia.org/wiki/Haversine_formula`

[b] It works well for most cases, but it sometimes fails due to a variable distance between points in polylines. Consequently, some images might not contain road at all or just partially. This issue is addressed during image labelling by putting such images in a particular class.

2.2 Data Labelling

This section depicts the annotation process of collected images. It introduces the classification categories, tools used for labelling and presents statistics of the newly created dataset.

2.2.1 Classification Classes

The classification classes were roughly proposed in Section 1.4. However, with respect to the nature of collected data, it is necessary to extend the categories and provide more thorough description. Therefore, we suggest the following classes:

1. An excellent uniform surface without any repairs, cracks or potholes, having solid curb. Typically a new asphalt surface. The only exceptions are straight fixed longitudinal cracks in the middle of the road to (i.e. close to/instead of the centre line marking, where motorcyclists do not ride.

2. Still very good surface with minor repairs of linear cracks ("snakes") in all directions, tiny unfixed cracks, or slightly rougher surface but smooth and homogeneous. Holes, bumps and potholes of any kind are not allowed.

3. Roads have partially cracked surface, large patches, slight unevenness or bumps. An inhomogeneous surface is allowed but without potholes.

4. Roads have an old rough surface, severe alligator cracks, bumps, potholes. Typically also a diverse surface or multi-layer patch repairs.

5. Any unpaved road which is not suitable for a road motorcyclist due to little grip, i.e. cobblestone, forest or dirt road. It could also be a wooden bridge.

6. Generally pictures that do not contain roads, or only a small fraction of it not allowing to determine the quality. E.g. images with cars, trees, fields, buildings or generic "no imagery" images.

2.2.2 Labelling Framework

Since there were about six thousand images to annotate, we wanted to make the processing time efficient and easy. Hence, a proper tool for labelling was

required. Before conducting a survey, the following required properties were stated:

- free of charge[a],
- simple to set up,
- usable both on computers and mobile devices,
- ability to export labels,
- ability to review and correct labels,
- secure online availability[b].

An ideal tool meeting our requirements would be a simple responsive web app with a login wall, allowing us to upload the images and label them by clicking a class button. The survey showed that there are quite a few available labelling tools online [39]. However, none of them meets our requirements. Many are paid (e.g. LabelBox, Hasty, Kili, Prodigy), and the free hosted services are only session-based without the ability to login from other device and continue (e.g. Make Sense, ImgLab). Also, several open-source projects can be run on-premise. Sadly, they often focused on bounding-box annotation, being too complex to use on mobile devices and to set up (e.g. Annotorious, LOST or CVAT). We tried setting up, e.g. Label Studio, which is a universal labelling tool supporting image classification. It looked very promising. Unfortunately, it is a very heavy-weight tool (not as easy to set up). Images import was only possible via an URL or special CSV/JSON format and without login (hence, if made public, anyone could add or delete images and labels). The last group of tools were locally running desktop applications, which do not meet the online availability requirements. To wrap up, we did not find any suitable labelling tool. Therefore, we designed our own.

Our framework consists of three micro-web apps. The first is used for labelling, the second is used to review labelled images, and the third for making corrections. All apps require a username and a password and are very straightforward to use.

[a] A paid service with sufficient free quota is plausible.

[b] An online availability was required for two reasons – being able to annotate on-the-go and possibly allowing multiple annotators at a time.

The main labelling app (Fig. 2.1) displays an image and seven buttons. The first six buttons are defined by the classification classes. The seventh button is for cases when an annotator is unsure about the class. Images marked as such can be filtered via the radio button "Unknown". When either button is clicked, the label is immediately saved, and the next image is shown. However, when an image has an assigned class, it is never shown again. The images marked as "Unknown" can only be viewed by a user with the *admin* role, as well as the results and corrector app.

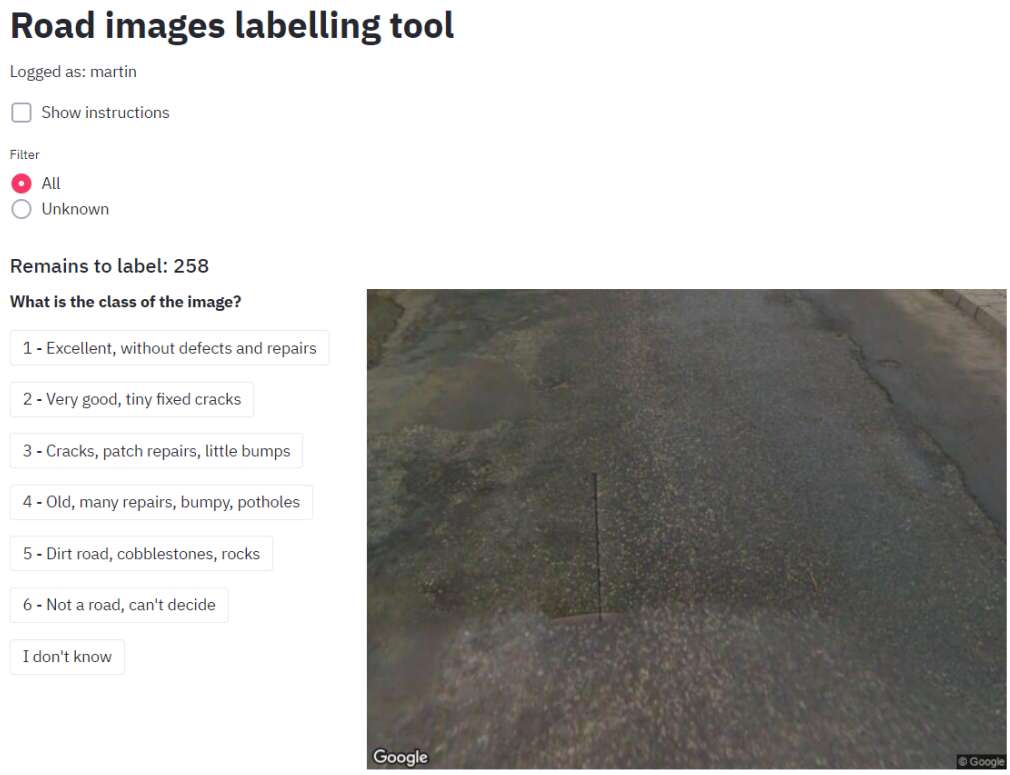

Figure 2.1: This picture shows a screenshot of the labelling web app, allowing us to annotate the images effectively.

The labelling results app (Fig. 2.2) shows a grid with images labelled with a given class. A label id is present for every image, which can be used in the corrector app.

The corrector app (Fig. 2.3) allows the admin user to correct the class of the given label. It only contains a text input and the same seven buttons as in the main labelling app.

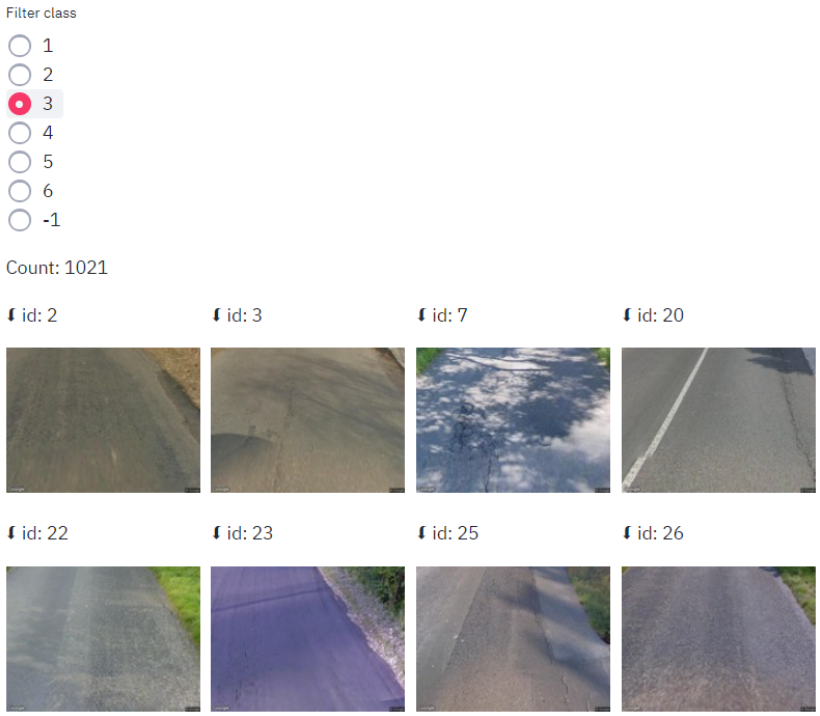

Figure 2.2: This picture shows a screenshot of the labelling results web app, enabling us to view the annotated images.

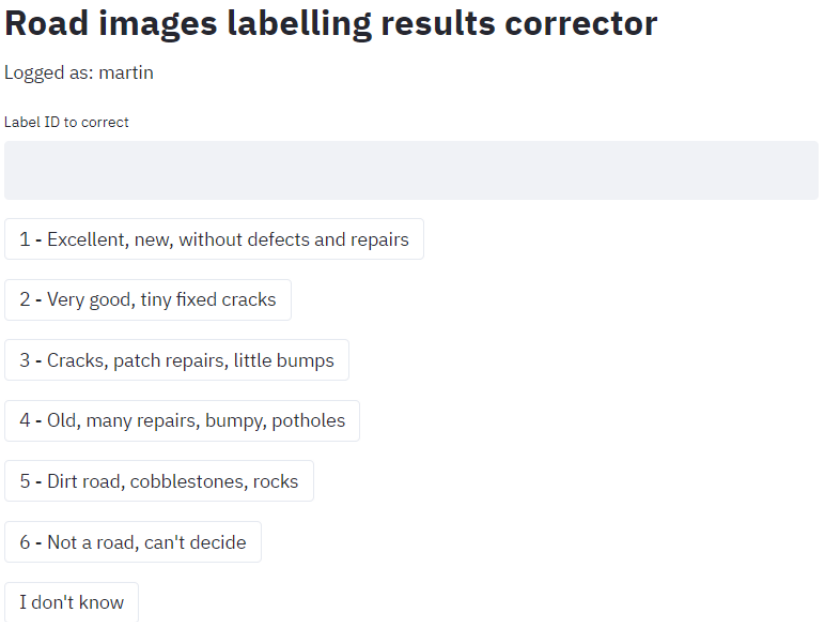

Figure 2.3: This picture shows a screenshot of the results corrector web app, enabling us to correct individual labels.

The apps are built with Streamlit open-source app framework[a], which turns Python script into a web app. Embedded SQLite database was used as data storage. The database scheme allows multiple users to be added with different roles. Every time the web page is loaded, the labelling framework tries to load new image definitions into the database from a specified source folder. The class definitions were hard-coded. To access the Streamlit app online, it needs to be hosted. We utilised the free Deepnote[b] Jupyter-like notebooks available online. However, each app built with Streamlit runs on a local-host port. To make the app publicly accessible, a secure tunnel was made to a public domain using LocalTunnel[c]. Then, our apps were available on custom subdomains of *localtunnel.me*, e.g. *roadlabelling.localtunnel.me*. Everything was for free. When the labelling was done, the apps were shut down. Examples of labelled images can be seen in Figure 2.4.

The next section describes the difficulties and challenges we encountered during labelling and how we dealt with them.

[a] `https://streamlit.io`
[b] `https://deepnote.com`
[c] `https://localtunnel.github.io/www`

Figure 2.4: This picture shows the examples of the labelled dataset. Each row depicts images given class, starting with first class at the top end sixth at the bottom.

2.2.3 Labelling Challenges

Although the class descriptions are relatively detailed, there were quite a few confusing images lying in between some categories. For example, first-class roads with drainage grates, different roads connections, extremely narrow roads or roads under reconstruction, where one lane is new and the other excavated. The general policy in labelling was to assign a label according to the most worsening part of the road in the picture. However, if we recall the purpose and use case of the model being developed in this work, such occasional elements could cause assigning a lower qualitative class than expected. For example, if a new road contains a drainage grate, it is unlikely the road contains another a few meters ahead. In contrast, a road with patches and repairs is likely to have more further on the way because the repairs are a sign of usage. Hence, these occasional elements were ignored.

Similarly, images often contained a seasonal impurity such as mud, oil, gravel, rests of snow or fresh-cut grass. Therefore, the following policy was applied in such cases: if the seasonal element is likely to be flushed by rain or cleaned by a street sweeper, ignore it and decide as if they were not present.

The rules described above had two meanings: enforcing the learning algorithms to focus more on the road surface and reducing subjective labelling. Despite that, we admit that there was still some room for unclear decisions. Which is also why we did not let the public help us with labelling, although it was initially considered and the framework was capable of it.

When all images had an associated class, the labelled dataset was created by exporting images into folders by classes. Table 2.2 depicts per-class statistics of the exported datasets. We can see that there are three dataset versions. The

Table 2.2: There are three datasets in the table. The first was labelled initially. Later during the work, it was extended into *v2* with the focus on classes 4-6. These two were used for training and validation. The *test* dataset was created to evaluate the best models' performance unbiasedly.

Dataset/Class	1	2	3	4	5	6	Total
v1	1712	1360	1021	790	45	148	5076
v2	1850	1518	1393	1067	88	178	6094
test	214	273	272	286	126	27	1198

reason is that we initially collected and annotated the dataset *v1*. However, classes 4-6 were very imbalanced. Therefore, an extension was made to reduce the imbalance, forming the dataset *v2*. Nevertheless, a significant class imbalance is still observable, and the issue is further addressed in the next section. The third version is a *test* dataset, only used for unbiased evaluation of the best models.

2.3 Data Balancing

Class imbalance in datasets is a common issue in the machine learning field. It can lead to biased models and serious overfitting, degrading quality of the models. It is always best to collect more data and make the dataset balanced. However, sometimes it is not possible. There are several methods that can be used for balancing classes – undersampling and oversampling. We focus only on the oversampling technique to generate more data samples. Undersampling is not suitable for our dataset, because it is still relatively small.

The oversampling is done on a *train* set only, not to devalue the purpose of a *validation* set. We set validation split to 25% from original datasets *v1, v2* or their derivates introduced later in this chapter. This section introduces the oversampling techniques used in this work.

2.3.1 Image Transformations

Image Transformation is a common oversampling technique. The idea is to generate new samples by applying some transformation to existing data. The transformation can be geometrical, such as rotation, flip or zoom. Besides, it can be a colour transformation, where we can adjust, e.g. brightness, contrast, or saturation. The transformation usually has several random real parameters specifying the factor of change.

In this work, we created two oversampled datasets with the following transformations:

- random rotation in range 0°-10°,
- random brightness adjustment from range (0.5, 1.3)[a],
- random shear in range 0°-10°,

[a] Values below one means dimming, values above means brightening.

- random horizontal flip.

All the mentioned transformations were applied together. However, the parameters were chosen randomly for every picture during the oversampling process. The oversampling was based on the dataset *v1* and followed two strategies.

The first strategy aimed only to balance the less-frequent classes with the most-frequent. Hence, there were no new samples in the first class, but about 1,300 new samples in the sixth class.

The second strategy involved oversampling in a way that all classes in the oversampled dataset contained about 2,500 samples. Therefore, even the most frequent classes contain augmented data. We call the oversampled datasets *v1_augmented_1* and *v1_augmented_2*. They contain 9,097 and 16,431 samples, respectively.

2.3.2 SMOTE

SMOTE stands for Synthetic Minority Oversampling Technique, which utilizes the KNN algorithm to generate new samples [40]. While it is relatively successful on low-dimensional datasets, the results with high-dimensional datasets are not guaranteed due to a low density of the data in the feature space [41]. Our images have $640 \times 480 \times 3 \approx 9 \times 10^5$ dimensions, which is very high considering we only have about a thousand samples in classes 1-4 and about a hundred in classes 5-6. The dimensionality would be high even if we downsampled the images to half (which would, however, be on the limit in terms of the detail loss and ability to classify). Therefore, we did not expect any stunning results and took it only as an experiment.

We utilised SMOTE using the *imbalanced-learn* Python package [42]. The results were not satisfactory, as suggested by the theory. The images generated using SMOTE have unrealistically adjusted colours and contain a lot of noise and artefacts. Such examples can be seen in Figure 2.5. Based on the results of preliminary experiments, we concluded that an oversampled dataset using this technique is not beneficial. It would only confuse the learning algorithm. Hence, this oversampled dataset using SMOTE was not used in further experiments.

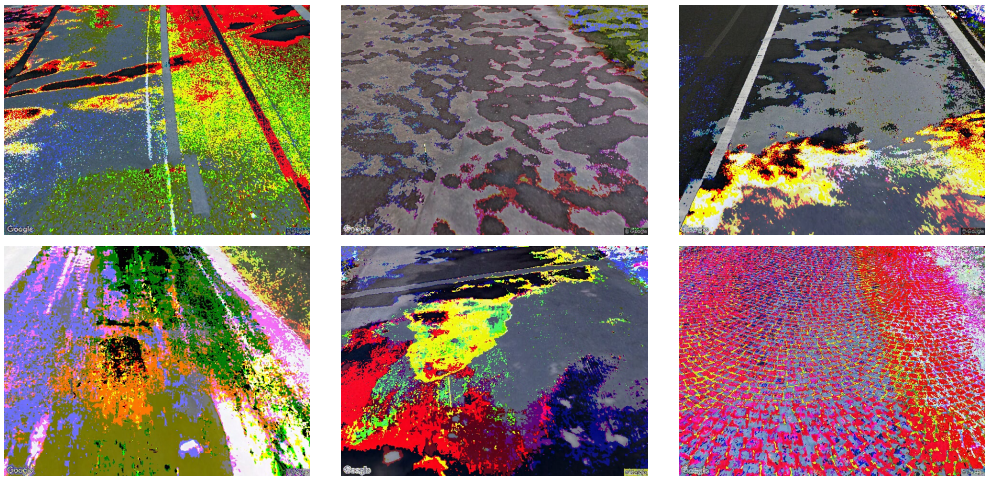

Figure 2.5: This picture shows the examples of SMOTE generated samples. We can see that the images contain unrealistic colours and artefacts, which would only confuse the learning algorithms.

2.4 Other Preprocessing Techniques

In addition to the balancing techniques described in previous section, we experimented several preprocessing techniques with the aim to increase performance of the developed models.

2.4.1 Shadow Removal

Undoubtedly, shadows are a big issue in scene understanding. In our task, shadows might be misinterpreted as a patch repair, pothole or surface homogeneity, causing incorrect class predictions. Since roads often lead through woods and trees cast shadows on the roads, they are also heavily present in our dataset. Hence, we explored the state-of-the-art methods of shadow removal and their possible usages.

Finally, we took advantage of a method proposed by Cun et al. [43], which achieved impressive results on pavements, and their model is publicly available[a]. The model is based on a novel CNN architecture called Dual Hierarchical Aggregation Network and trained on data partially synthesised by their shadow matting generative adversarial network.

[a] `https://github.com/vinthony/ghost-free-shadow-removal`

After generating the shadow-free images, the results were promising. However, we quickly realised certain drawbacks.

While the removal worked reasonably well on images in which the shadow was present, it added artificial defect-looking objects to images without shadows. That is a huge issue, especially in the first two classes, as they would confuse with the third and fourth class. In some cases, the shadow was detected correctly but only partially removed, causing the removed part to look more like a patch repair. Examples of shadow-free images can be seen in Figure 2.6.

Despite the questionable results, we decided to include the shadow-free dataset in our experiments, described more in the next chapter. We refer to this dataset as *v2_shadow_free*.

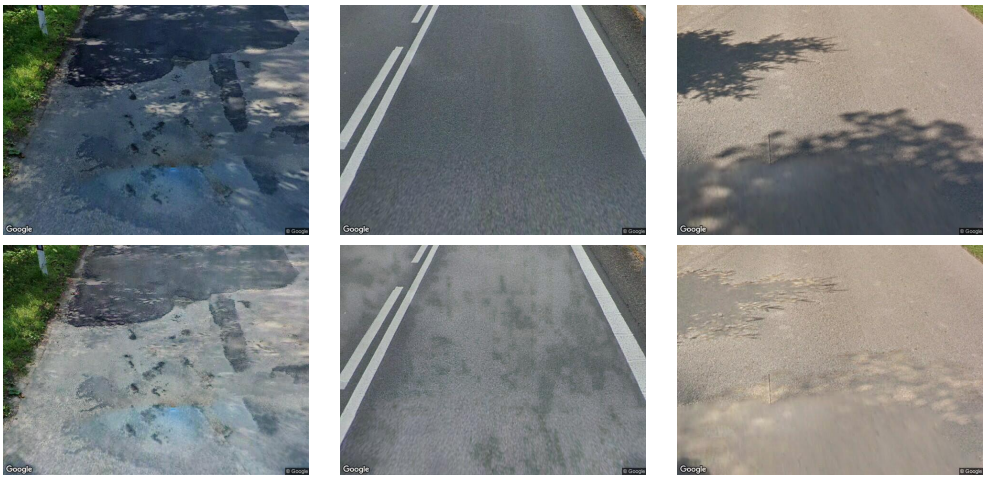

Figure 2.6: This picture shows the examples of shadow-free images using the model proposed in [43]. The top images are the originals, and the shadow-free are at the bottom. We can see that the shadow was removed quite successfully in the first image. Contrastingly, the removal in the other two images generated new defect-looking artefacts.

2.4.2 CLAHE

CLAHE is an improved adaptive histogram equalization with the ability to clip the output limit range, proposed by Zuiderveld [32]. The method is generally used to improve contrast in images and achieves good results.

In contrast to the previous methods, this technique does not cause such dramatic changes to the images, resulting in unrealistic artefacts or patch

repairs. Nevertheless, it is not problem-free either. If the clip limit is too high, the contrast can pull up even the tiniest cracks (noise) from the roads, making them all damaged from the classifier point of view. It is essential to choose the proper clip limit, which would help identify defects and not highlight the surface noise. However, choosing the right clip limit is a matter of trial and error.

We utilised the OpenCV library [44], in which CLAHE is already implemented. Based on several experiments, we generated datasets with the following clip limits: $1, 1.5, 2$ and 3. We refer to them as *v2_clahe_1, v2_clahe_1_5, v2_clahe_2, v2_clahe_3*. Examples of different clip limits on our dataset can be seen in Figure 2.7.

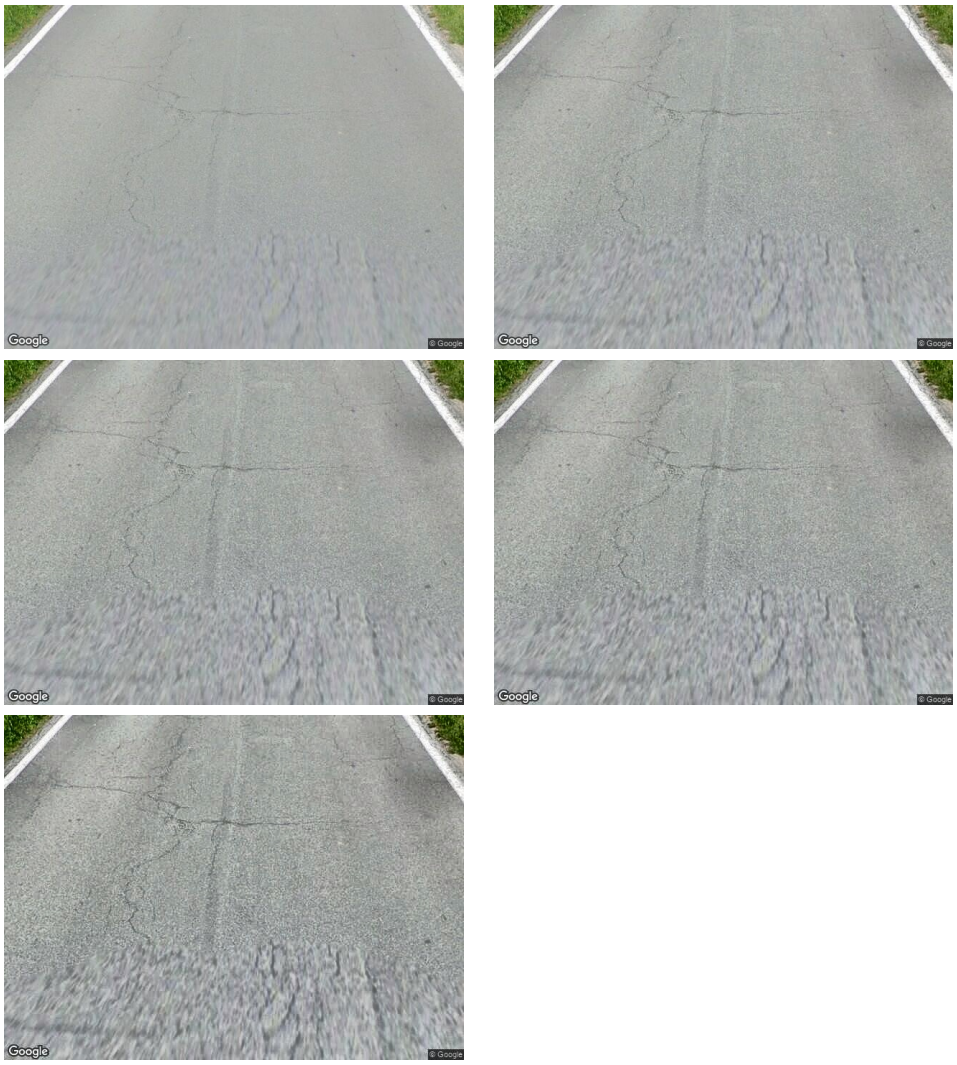

Figure 2.7: This picture shows the examples of CLAHE generated samples. From top left to right and bottom: original, clip limit of 1, 1.5, 2 and 3. We can see that clip limit of 3 noticeably highlights tiny cracks. A higher clip limit might lead to higher confusion, making the classes less separable.

Chapter 3

Experiments

This chapter introduces the implemented technologies used for training and performing experiments. The design of experiments and methodology is also explained along with the evaluation metrics. Finally, our implemented approaches are described.

3.1 Implementation

In this section, we mention the relevant frameworks and libraries used in this work. All the implementations were done on a machine running Windows 10 20H2 equipped with Core i7 CPU, NVIDIA GeForce GTX 1070 GPU and 32 GB RAM.

Developers and data scientists generally have more than one option in terms of technologies they want to implement. Hence, everyone can pick the most suitable tool for their task. We decided to utilise the following technologies based on the related works and our survey.

We used Anaconda Data Science Platform [45] as a base for all other software used for development. This platform allows us to set up a containerised environment with native GPU support, which is crucial for CNN learning due to high computational requirements. Besides, environments isolate libraries and their versions, avoiding conflicts with other software installed on the machine.

As for programming language, we picked Python[a], which is heavily used in the machine learning field for its wide range of support of data science tools and libraries. In addition, its syntax is concise and code well readable.

NumPy[b] is an open-source Python package for effective numerical calculations, including matrix operations or multidimensional array transformations.

TensorFlow 2.3 [46] is a machine learning platform developed by Google. It bundles plenty of tools and algorithms, enabling us to build neural networks suitable for many tasks, such as image classification or audio recognition. TensorFlow itself is a bit low-level. Hence, we utilised Keras library [47], which provides a high-level Python API for TensorFlow, specially designed for easy experimenting with deep neural networks, including CNNs.

Before we set up our experimentation flow described further in text, we used Jupyter[c] notebooks for prototyping in the early stages. It is an interactive web application with the ability to run Python code in cells, print graphs and images or add textual notes for clarification.

Lastly, the Matplotlib library [48] was used to visualise model training runs, which helped us evaluate models' performance.

3.2 Design of Experiments

This section presents our experimentation flow, including our evaluation metrics, and describes how we proceeded with hyper-parameter tuning.

3.2.1 Evaluation Metrics

For the purpose of this work, we used two metrics for evaluating models' performance – accuracy and F1-score. Accuracy is a measure of the model's overall ability to classify images correctly, calculated as a ratio of correct predictions and all predictions.

$$\text{Accuracy} = \frac{\text{Correct predictions}}{\text{All predictions}} \tag{3.1}$$

[a] https://www.python.org
[b] https://numpy.org
[c] https://jupyter.org

F1-score can be interpreted as a harmonic mean of precision and recall. For multi-class classification, precision is the fraction of the sum of all true positives (TP) across all classes and the sum of all TP and false positives (FP) across all classes.

$$\text{Precision} = \frac{\sum_{c}^{\text{all classes}} TP_c}{\sum_{c}^{\text{all classes}} (TP_c + FP_c)} \tag{3.2}$$

The recall is calculated as the ratio of the sum of all TP across all classes and the sum of all TP and false negatives (FN) across all classes.

$$\text{Recall} = \frac{\sum_{c}^{\text{all classes}} TP_c}{\sum_{c}^{\text{all classes}} (TP_c + FN_c)} \tag{3.3}$$

To understand how well the model classifies in each class, we calculated the F1-scores per class. In that case, we do not sum across all labels in the precision and recall formulas, but the c is fixed to the corresponding class.

Since our dataset is not perfectly balanced, the F1-score can be more meaningful for model performance evaluation rather than accuracy.

3.2.2 Methodology

Our experimentation workflow consisted of three components. In the first, we defined individual experiments. The second component was used to load the given experiment definition, run the experiment and save its results. Finally, the third component was used to evaluate the saved models.

Every experiment definition had an associated id and contained a definition of the model's architecture, including its hyper-parameters and desired optimiser[a] to be used during training. Moreover, a dataset (*v1, v2, v1_clahe,...*), validation split, or colour space (RGB/greyscale) could be specified for each experiment.

When we wanted to run a predefined experiment with a given id, we used the second component. It took care of loading specified dataset, compiling the model architecture and setting up callbacks. The callbacks were used for

[a] `https://www.tensorflow.org/api_docs/python/tf/keras/optimisers`

two things – to save intermediate network weights and for early stopping. TensorFlow calculates the model's accuracy after every epoch using a *validation* set. We took advantage of that and saved model weights of the best and latest epoch in every experiment. The early stopping was primarily used for saving time and resources when the model converged before reaching the specified number of epochs.That is when the accuracy had not improved for a given number of epochs, known as patience. We usually set patience from ten to fifteen epochs, depending on the subjective convergence speed.

The third component was used to calculate the models' accuracy and F1-scores, given an experiment id and dataset. We used the *validation* set for comparing model performance among each other. Later, we picked the best-performing models on a *validation* set and evaluated them using the *test* dataset to determine the unbiased performance.

3.2.3 Model Tuning

Building a machine learning model means experimenting with many different architectures, hyper-parameters, and in our case, also with different datasets. Therefore, it is essential to run and evaluate experiments systematically. To achieve that, we were always running several experiments in a batch, each adjusting only one parameter against a referential model. Each experiment was evaluated based on the evaluation metrics. Adjustments leading to improvement formed a new referential model for further experimenting. In total, we ran 164 experiments. The best ones are described in the next section.

3.3 Implemented Approaches

This section introduces the best approaches we found in the experiments. There are two main types of implemented approaches – custom architectures trained from the ground up and a transfer learning approach, where we took advantage of pre-trained networks.

3.3.1 Custom Architectures

In the first phase of the work, we experimented with custom architectures, along with optimisers, optional layers (described in Subsection 1.5.4), activation functions and different real-time augmentation. We used the dataset *v1* for

Table 3.1: Architecture of the *E16* model. Abbreviations: CONV=Convolutional, N=neurons, K=kernel size, S=stride, R=rate. Dense layer is the Fully-Connected layer.

#	Layer
1	CONV (N8, K3, S1), ReLU
2	CONV (N8, K3, S1), ReLU
3	MaxPooling (K2, S2)
4	CONV (N16, K3, S1), ReLU
5	CONV (N16, K3, S1), ReLU
6	MaxPooling (K2, S2)
7	CONV (N32, K3, S1), ReLU
8	MaxPooling (K2, S2)
9	Dropout(R0.2)
10	CONV (N64, K3, S1), ReLU
11	MaxPooling (K2, S2)
12	CONV (N128, K3, S1), ReLU
13	MaxPooling (K2, S2)
14	Dropout(R0.1)
15	CONV (N256, K3, S1), ReLU
16	MaxPooling (K2, S2)
17	Flatten
18	Dense (N128), ReLU
19	Dense (N6)

experimenting with custom architectures. The best architectures we managed to build are described below.

Our first proposed architecture is composed of eight convolutional layers but only six pooling layers. The architecture is depicted in Table 3.1.

There are two convolutional layers right after another at the top of the architecture. Two dropout layers were applied to reduce overfitting. All convolution filters were of size 3×3. After every convolution, the ReLU activation function was applied. Before training, the input pictures were resized to resolution 480×360. Also, real-time augmentation was applied to reduce overfitting, including random horizontal flip, random rotation and random zoom. The random rotation ranged $[-108°; 108°]$, and zoom ranged $[0\%; 30\%]$. Finally, inputs were scaled to the unit range. The model was trained using the

Table 3.2: Architecture of the *E46* model. Abbreviations: CONV=Convolutional, N=neurons, K=kernel size, S=stride, R=rate. Dense layer is the Fully-Connected layer.

#	Layer
1	CONV (N8, K3, S1), LeakyReLU
2	MaxPooling (K2, S2)
3	Dropout(R0.2)
4	CONV (N16, K3, S1), LeakyReLU
5	MaxPooling (K2, S2)
6	CONV (N32, K3, S1), LeakyReLU
7	MaxPooling (K2, S2)
8	CONV (N64, K3, S1), LeakyReLU
9	MaxPooling (K2, S2)
10	CONV (N128, K3, S1), LeakyReLU
11	MaxPooling (K2, S2)
12	Dropout(R0.1)
13	Flatten
14	Dense (N128), ReLU
15	Dense (N6)

RMSprop optimiser [49] with a learning rate of 0.001. The training converged after about 130 epochs. We refer to this model as *E16* since it was the 16th experiment.

The second model's architecture can be seen in Table 3.2. It is composed of five convolutional layers and the same number of pooling layers. There are two dropout layers present. The activation function used in this architecture was LeakyReLU [50], with hyper-parameter alpha set to 0.3. Real-time augmentation was applied as well. In addition to *E16*, random contrast with factor 0.3 was added along with both horizontal and vertical flip. The model was trained using the Adam optimiser [51] with a learning rate of 0.001, converging after about 150 epochs. We refer to this model as *E46*.

Thirdly, we implemented the architecture proposed by Rateke et al. [7], referred to as *E67*. It consists only of three convolutional layers, each followed by a pooling layer, being the simplest architecture among the proposed. We can see the architecture in Table 3.3. However, we did not apply the same augmentation as in [7], but similar to *E46*. Additionally, we resized the input

Table 3.3: Architecture of the *E67* model. Abbreviations: CONV=Convolutional, N=neurons, K=kernel size, S=stride, R=rate. Dense layer is the Fully-Connected layer.

#	Layer
1	CONV (N32, K3, S1), ReLU
2	MaxPooling (K2, S2)
3	CONV (N32, K3, S1), ReLU
4	MaxPooling (K2, S2)
5	CONV (N64, K3, S1), ReLU
6	MaxPooling (K2, S2)
7	Flatten
8	Dense (N128), ReLU
9	Dense (N6)

image down to resolution 240×320. This model was also trained with the Adam optimiser with a learning rate of 0.001. The training converged after about 250 epochs.

3.3.2 Transfer Learning

This subsection describes several pre-trained networks we experimented with. These networks served as a base model, with loaded frozen weights trained on the ImageNet dataset. The only trainable weights were connections to an output layer that we added to fit our classification task. Later, based on the preliminary results, we fine-tuned the best models to increase the performance even more. That was done by training all weights, including those initially frozen.

MobileNet [29] is a an efficient CNN designed with low resources demands and balanced accuracy-latency trade-off. Thanks to architecture using depth-wise separable convolutions [52], it is lightweight with only 4.2M neurons and 88 layers[a]. The authors demonstrated a wide range of applications, including object detection and classification. Each pre-trained network assumes specifically preprocessed data. For MobileNet, input samples must be scaled to the range [-1;1], sample-wise. Since the network was trained on images with resolution 256×256, it is recommended to resize our inputs to that resolution. However,

[a]Including output classification layers for the ImageNet dataset

we consider such resolution too low for our application due to significant detail loss on road surface needed for effective quality classification. Hence, we decided to resize the input images to 360×360, preserving the square aspect ratio of the data on which MobileNet was trained.

Another pre-trained network we experimented with is the successor of MobileNet – MobileNetV2 [53]. It has even less neurons (3.5M), while achieving higher accuracy on the ImageNet dataset. The input preprocessing is the same as for MobileNet.

DenseNet-121 [54] is a deep CNN with 121 layers and 8M neurons. The novelty in this architecture was connecting each layer to every other subsequent layer. That means every layer takes inputs from all preceding layers, which, according to authors, should strengthen feature propagation and stimulate feature reuse. Compared to MobileNet, it achieves 5% higher accuracy on the ImageNet dataset. As for preprocessing, DenseNet assumest input scaled to the range [0;1] and normalised with respect to the ImageNet dataset. Additionally, we resized the images to the same resolution as with MobileNet.

The last model we utilised for transfer learning was the InceptionResNetV2 [55]. It is a hybrid of ResNet [22] and Inception [23] architectures. ResNet introduced so-called residual connections, which are direct connections from each layer to a subsequent layer 2-3 hops away. The idea behind Inception architectures is the use of different kernel sizes in parallel to decrease sensitivity to the size of an object in an image. Therefore, the hybrid network is both deep and wide. With its 572 layers and 55M neurons, it is the most complex architecture we experimented with. Regarding the inputs, it assumes preprocessed data in the same way as MobileNet networks – scaled to [-1;1], and we also resized the images as with other networks.

Chapter 4

Results

This chapter presents our findings and experimental results of the implemented methods and trained models. We describe which preprocessing and data augmentation techniques positively and negatively impacted models' performance. Next, we discuss the results with a potential interpretation. The results are grouped by implemented approaches – custom architectures and transfer learning.

4.1 Custom Architectures

One of the main issues we encountered when experimenting with custom architectures was overfitting. Therefore, we applied real-time data augmentation, which significantly helped. We found the following random transformations to be helpful: contrast, horizontal and vertical flip, rotation and zoom, as used in models *E46, E67*. Oppositely, random shear or random crop to smaller size had a negative impact on the results. Nevertheless, even with the mentioned augmentation techniques, the overfitting was still noticeable. Next, we experimented with balanced datasets *v1_augmented_1, v1_augmented_2*, created by oversampling *v1* dataset using random image transformations. To our surprise, the accuracy was significantly worse (by about 10%) than in the case with the real-time augmentation.

Further experimenting showed that the RMSprop optimiser tends to reduce the overfitting compared to the Adam optimiser on the same architecture. We observed a trade-off between overfitting and the best possible accuracy.

On the one hand, the RMSprop optimiser reduced overfitting, but on the other also decreased maximum training and validation accuracy. Adam optimiser achieved higher validation accuracy but also larger overfitting. We observed that this effect could be decreased, contradictory, by applying fewer augmentation techniques, as in model *E16*.

We also experimented with colour space and optional layers. Converting images to grayscale resulted in a slight decrease in accuracy. Dropout layers positively impacted overfitting with factors around 0.2-0.3. Higher values negatively impacted overall models' performance. Surprisingly, batch normalisation layers, aiming to stabilise the learning, caused the opposite. Figure 4.1 depicts such behaviour.

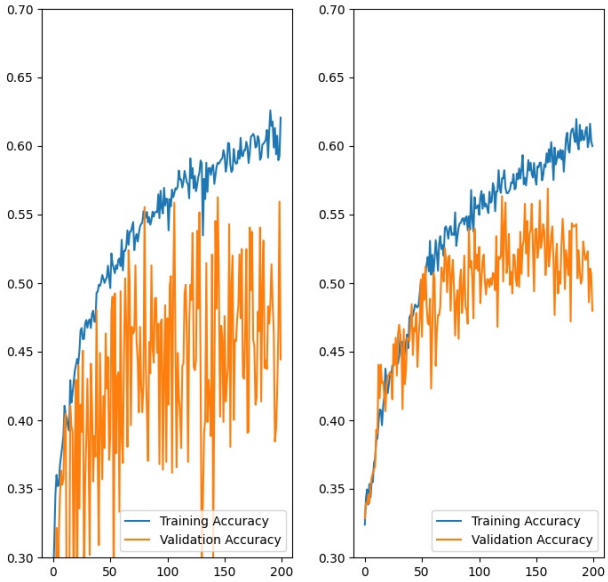

Figure 4.1: An example of batch normalisation negative effects on the same architecture. The plots show training and validation accuracy [$\times 100\%$] over epochs while training. We can see significant instability with batch normalisation layers (plot on the left) compared to architecture without batch normalisation layers (on the right).

We present results of the implemented approaches *E16, E46* and *E67* in Table 4.1. It can be seen that we achieved about the same accuracy with three different architectures. However, their F1-scores differ. The biggest differences are in second and third class, which are also classes with the lowest overall scores. We attribute it to the fact that both categories can contain similar features,

Table 4.1: The table shows the results of models with custom architectures. Results contain validation accuracy (Acc) and per-class validation F1-score.

Model	Acc	F1-score					
		1	2	3	4	5	6
E16	**59.42%**	**71.44%**	40.61%	**52.73%**	61.98%	57.14%	**86.36%**
E46	58.23%	71.41%	36.92%	42.70%	**66.37%**	61.54%	81.25%
E67	59.18%	70.29%	**47.37%**	51.21%	62.11%	**63.16%**	81.82%

e.g. (fixed) cracks, but with different severity. Additionally, we know the manual labelling of these classes was sometimes disputable. The sixth class has the highest F1-score, which we explain by many identical placeholder images for missing imagery on Google Street View. We later discovered that such images accounted for about 17% of all images in the sixth category.

4.2 Transfer Learning

We utilised four different pre-trained networks and trained them with several data augmentation techniques. Initial training was done on the dataset *v1*, to be comparable with custom architectures. Surprisingly, our data augmentation decreased the validation accuracy of transfer learning models by about 5%. While it almost removed the overfitting, we did not consider this a good trade-off and continued experimenting without any augmentation. It showed that DenseNet performed very well in terms of not overfitting (only about 4%, while others 10%+). We can see the results of models trained on the dataset *v1* in Table 4.2. Transfer learning approaches achieved higher accuracy by about 6%. The first class achieved higher F1-score by about 4%, classes 2-4 by about 10% and class 5 by about 20%.

For time reasons, we needed to pick one of these models for further experimenting with dataset *v2*. We picked MobileNet since its overall accuracy is the best.

Experimenting with dataset *v2* involved several variations, including CLAHE and shadow removal preprocessing. All experiments were trained on the previously mentioned MobileNet architecture. The results are depicted in Table 4.3. The model's accuracy on dataset *v2* increased by about 2%. The F1 score of the first and second class did not significantly change, which was

Table 4.2: The table shows the results of models based on the transfer learning approach trained on the dataset *v1*. Results contain validation accuracy (Acc) and per-class validation F1-score.

Model	Acc	F1-score					
		1	2	3	4	5	6
Mobile Net	**66.75%**	75.11%	52.20%	**62.45%**	72.63%	80.00%	83.15%
Mobile NetV2	64.78%	74.91%	47.00%	60.26%	70.46%	82.35%	**86.67%**
Dense Net-121	65.41%	74.42%	54.29%	57.76%	71.43%	87.50%	84.71%
Inception Res-NetV2	65.48%	**75.80%**	**55.83%**	51.10%	**73.51%**	**88.89%**	77.27%

Table 4.3: The table shows the results of models based on the transfer learning approach trained on the dataset *v2* and its preprocessed variations. Results contain validation accuracy (Acc) and per-class validation F1-score.

Dataset	Acc	F1-score					
		1	2	3	4	5	6
v1	66.75%	75.11%	52.20%	62.45%	72.63%	80.00%	83.15%
v2	68.35%	75.66%	51.14%	67.86%	76.89%	65.31%	81.25%
v2 clahe_1	69.34%	75.76%	58.04%	67.06%	77.41%	63.83%	83.33%
v2 clahe_1_5	69.01%	75.85%	54.86%	68.27%	76.02%	65.22%	83.50%
v2 clahe_2	**70.12%**	76.69%	56.69%	**69.46%**	77.64%	63.83%	84.00%
v2 clahe_3	68.29%	74.11%	**58.14%**	63.61%	77.80%	66.67%	**84.78%**
v2 shadow free	69.67%	**76.97%**	55.25%	67.64%	**77.98%**	**72.73%**	82.83%

expected, as the dataset *v1* was extended with the focus on categories 3-5. The F1-score of the third class improved by about 5%. The fourth class achieved gain by about 4%. In contrast, the score of the fifth class dropped by 15%. We explain that by a significant change made by the dataset extension, as the count of samples was doubled, adding new variety to the data. However, the fifth class still contains only 88 samples, which is very few for a model to learn the variety. Hence, we do not consider a 65% score a bad result. Additionally, the model's overfitting decreased to about 5%.

In Chapter 2, we discussed our concerns about CLAHE that only some classes might benefit from higher contrast. The results confirmed that. The clip limit of 3, which is the highest we experimented with, decreased the F1-score of the first class compared to the original data. However, a 1.5% drop is lower than expected. Additionally, all CLAHE clip limits we experimented with increased the score of the second class. The biggest gain was with a clip limit of 3 by 7%. However, in contrast, the clip limit of 3 caused the biggest score drop in the third class – about 4%. Therefore, we consider lower clip limits and lower variation in performance change a better choice. Specifically, the clip limit of 1 seems to perform the best. It has slight F1-score improvement in the first and fourth class, about the same improvement ($\sim$7%) in the second class as with the clip limit of 3, and only a slight drop in the third and fifth class.

The results with shadow-free images were surprising. Although the shadow removal introduced many defect-looking blobs, there was at least slight F1-score improvement in all classes except the third, which had a slight 0.22% drop. Also, while the CLAHE preprocessing does not seem to reduce overfitting, the shadow removal reduced the overfitting to about 2%.

Models using *v2*, *v2_clahe_1* and *v2_shadow_free* datasets were picked for fine-tuning. Table 4.4 shows the results. We can see that fine-tuning increased the accuracy by about 4% on all datasets. We observe that the model with shadow-free dataset only outperformed *v2* in the third and fifth class, while without fine-tuning, it outperformed in all classes, except the third. Also, the F1-score drop is more significant with the shadow-free dataset ($\sim$5%). The model with CLAHE dataset has the best accuracy. Although it does not outperform the model with *v2* in all classes, it does not contain any significant drops compared to the model using the shadow-free dataset.

Table 4.4: The table shows the results of fine-tuned models based on the transfer learning approach trained on the dataset *v2* and its preprocessed variations. Results contain validation accuracy (Acc) and per-class validation F1-score.

Dataset	Acc	F1-score					
		1	2	3	4	5	6
v2	74.00%	78.64%	62.81%	**75.80%**	80.09%	65.22%	82.11%
v2 clahe_1	**74.52%**	**79.83%**	61.60%	75.14%	**81.35%**	72.34%	**82.98%**
v2 shadow free	73.21%	78.99%	**64.40%**	70.90%	78.08%	**77.55%**	81.19%

Table 4.5: The table shows the results of fine-tuned models based on the transfer learning approach trained on full dataset *v2* (or its preprocessed variation). Results contain *test* accuracy (Acc) and per-class *test* F1-score

Model	Acc	F1-score					
		1	2	3	4	5	6
v2	68.28%	75.18%	64.92%	66.55%	76.15%	40.74%	59.74%
v2 clahe_1	**71.04%**	**76.34%**	**68.95%**	**69.85%**	**78.81%**	**47.95%**	**59.77%**
v2 shadow free	64.77%	62.20%	61.25%	63.75%	74.56%	46.43%	72.73%

Next, we trained the fine-tuned models on all the data from given datasets, i.e. without a validation split. Models were then evaluated on the *test* dataset. The results can be seen in Table 4.5. Overall, the results on the *test* datset seem to be relatively consistent with the validation accuracies in Table 4.4, except the fifth and sixth class, where is a noticeable F1-score drop by about 20%. We explain that similarly as the drops occurring after extending the *v1* dataset. There was a lot of new data variety in the *test* dataset in these two categories. Additionally, there are more samples for the fifth class in the *test* dataset than in the *v2* dataset, which the model was trained on.

Table 4.5 also shows that the performance of the model using the full shadow-free dataset dramatically dropped and is behind both the original *v2* and CLAHE model. The sixth class is the only exception, which achieved inexplicable 10%

improvement. The CLAHE model achieved the highest accuracy again, and it does not contain any significant drops as the model trained on the shadow-free dataset.

We conclude that the evaluation on the *test* set did not show significant overfitting issues in classes 1-4. Nonetheless, there is an overfitting issue with classes 5-6, which we believe is caused by the lack of training samples.

Conclusion

Finding high-quality roads for driving pleasure can be challenging and time-consuming, even when we utilise modern street view services. This thesis focused on an automatic road quality classification, which could streamline such a process. In this chapter, we would like to review and summarise our contributions and propose a direction for future improvement. Also, potential ethical issues linked with the usage of this work are briefly discussed.

Contributions

In the introductory part of this work, we stated the motivation for automated road quality evaluation and defined our goals. This section outlines their fulfilment.

The first objective was to survey the related state-of-the-art works and available datasets. The survey shown our use is particular, and none of the available datasets fits our needs.

Hence, a new road quality image dataset focusing on driving pleasure was collected from Google Street View and manually annotated into six classes[a]. That is our first contribution. Our second contribution is a lightweight labelling tool we developed while creating the new dataset.

Another objective was to implement at least two approaches. We realised several different CNNs with custom architectures and adopted several pre-trained

[a]Dataset available at: `https://github.com/lenoch0d/road-quality-classification`

networks. Additionally, we experimented with numerous augmentation and preprocessing techniques. The best models were evaluated on a *test* dataset to get unbiased results. Our third contribution is a model achieving 71% accuracy and F1-scores (rounded) 76%, 69%, 70%, 79%, 48% and 60% for classes 1-6, respectively. Before concluding on these scores, we would like to mention the challenges of our data domain we faced.

Our classification classes do not describe a single feature in the images, but an overall appearance, making it challenging to label images unbiasedly without fluctuations (i.e. a similar image is labelled differently due to uncertainty). Additionally, the images in our dataset contain various lighting conditions, including shadows and sharp sunlight. These elements lead to confusion between classes having smooth, coherent surface and classes with potholes and patches. We experimented with two preprocessing techniques to mitigate the problem – shadow removal and CLAHE. The results showed each technique likely improves and decreases the performance of different categories.

Despite the F1-scores of the proposed model not being dramatically high and classes 5-6 significantly underperforming others, we conclude our model is satisfactory for use in practice. Particularly for the use case introduced in this thesis. We claim that based on a visualisation of the model's predictions created beyond the scope of this work, which can be seen in Appendix B.3.

To sum up, we fulfilled all the established objectives, and our work may serve as a basis for additional work.

Future Work

We see two ways in extending this work. One way involves improving the current classifier. Extending the *v2* dataset with new samples in all classes, especially in classes 5-6, would undoubtedly increase the performance. Next, we skipped further experimenting with the DenseNet-121 network due to limited time and computational resources. However, DenseNet-121 had the least overfitting issues. Hence it might be insightful to explore its capabilities on our dataset. Also, removing the generic "no imagery" images from the sixth class might lead to a more relevant evaluation of models' qualities. Since the files are always the same, they could be filtered automatically even without the classifier.

The second direction in further development involves practical usage of the proposed classifier. In Appendix B.3 we demonstrated a possible visualisation enabling a user to identify high-quality and low-quality roads quickly. The model could be wrapped into an application taking waypoints as an input and producing such visualisation as an output. Further, this application could automatically suggest directions between two places using only high-quality roads.

Ethical Issues

Every machine learning model brings potential risks and ethical issues when used in a real-world application. Therefore we should consider its negative impacts.

The motivation for this work was to make it easier for people who drive for pleasure to find high-quality roads. What if such functionality was implemented into a navigation software, similarly to traffic jam signalling? It would become effortless for everyone to drive on high-quality roads and avoid the bad ones, like avoiding a traffic jam. Consequently, there would likely be much higher traffic on these preferred roads. Inevitably, higher traffic means more load, possibly exceeding the designed limits and leading to faster surface degradation. When a given road would become low-quality, the traffic would move to another road with higher qualities. Taking into account that less-frequent roads take longer to be repaired for reasons mentioned in the introduction, it could lead to systematic road damaging. This potential effect would vastly differ from the intended positive motivation of this thesis.

Although the negative impacts may seem absurd, we should be aware of them when designing an application based on our work.

Bibliography

1. ŠÁRKÖZI, Tibor. *Dopravní přestupky MHMP 2020*. Magistrát hl. m. Praha, 2021-02. Available also from: https://opendata.praha.eu/dataset/dopravni-prestupky-mhmp-2020.
2. SAMUEL, Arthur L. Some studies in machine learning using the game of Checkers. *IBM Journal of Research and Development*. 1959, vol. 3, no. 3, pp. 71–105. Available from DOI: 10.1147/rd.33.0210.
3. MURPHY, Kevin P. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012. Adaptive Computation and Machine Learning series. ISBN 9780262018029. Available also from: https://books.google.cz/books?id=NZP6AQAAQBAJ.
4. TIWARI, Saurabh; BHANDARI, Ravi; RAMAN, Bhaskaran. RoadCare: A Deep-Learning Based Approach to Quantifying Road Surface Quality. In: *Proceedings of the 3rd ACM SIGCAS Conference on Computing and Sustainable Societies*. Ecuador: Association for Computing Machinery, 2020, pp. 231–242. COMPASS '20. ISBN 9781450371292. Available from DOI: 10.1145/3378393.3402284.
5. AFENIKA, Adhelinia; GUNAWAN, P. H.; TARWIDI, D. Classification of Road Surface Quality Based on SVM Method. *Journal of Physics: Conference Series*. 2020, vol. 1641, p. 012064. Available from DOI: 10.1088/1742-6596/1641/1/012064.

6. HOFFMANN, M.; MOCK, M.; MAY, M. Road-quality classification and bump detection with bicycle-mounted smartphones. *CEUR Workshop Proceedings*. 2013, vol. 1088, pp. 39–43. ISSN 1613-0073.

7. RATEKE, Thiago; JUSTEN, Karla Aparecida; WANGENHEIM, Aldo von. Road Surface Classification with Images Captured From Low-cost Cameras — Road Traversing Knowledge (RTK) Dataset. *Revista de Informática Teórica e Aplicada (RITA)*. 2019. Available from DOI: https://doi.org/10.22456/2175-2745.91522.

8. NOLTE, M.; KISTER, N.; MAURER, M. Assessment of Deep Convolutional Neural Networks for Road Surface Classification. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018, pp. 381–386. Available from DOI: 10.1109/ITSC.2018.8569396.

9. DOSHI, Keval; YILMAZ, Yasin. *Road Damage Detection using Deep Ensemble Learning*. 2020. Available from arXiv: 2011.00728 [cs.CV].

10. PEI, Zixiang; ZHANG, Xiubao; LIN, Rongheng; SHEN, Haifeng; TANG, Jian; YANG, Yi. Submission of DD-VISION team in Global Road Damage Detection 2020. In: 2020. Available also from: https://pan.baidu.com/s/1VjLuNBVJGS34mMMpDkDRGQ. Password: xzc6.

11. HEDGE, Vinuta; TRIVEDI, Dweep; ALFARRARJEH, Abdullah; DEEPAK, Aditi; KIM, Seon Ho; SHAHABI, Cyrus. *Ensemble Learning for Road Damage Detection and Classification, Article In Press*. 2020. Available also from: https://github.com/USC-InfoLab/rddc2020. Accessed: 2020-03-17.

12. CHACRA, David; LEOPOLD, Henry; PINTO, Jeremy; LUNSCHER, Norman; YOUNES, Georges; ZELEK, John. Road Defect Detection in Street View Images using Texture Descriptors and Contour Maps. *Journal of Computational Vision and Imaging Systems*. 2016, vol. 2, no. 1. Available from DOI: 10.15353/vsnl.v2i1.94.

13. LEI, X.; LIU, C.; LI, L.; WANG, G. Automated Pavement Distress Detection and Deterioration Analysis Using Street View Map. *IEEE Access*. 2020, vol. 8, pp. 76163–76172. Available from DOI: 10.1109/ACCESS.2020.2989028.

14. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

15. MIKOŁAJCZYK, A.; GROCHOWSKI, M. Data augmentation for improving deep learning in image classification problem. In: *2018 International Interdisciplinary PhD Workshop (IIPhDW)*. 2018, pp. 117–122. Available from DOI: 10.1109/IIPHDW.2018.8388338.

16. GEIGER, Andreas; LENZ, Philip; STILLER, Christoph; URTASUN, Raquel. Vision meets Robotics: The KITTI Dataset. *International Journal of Robotics Research (IJRR)*. 2013. Dataset available from: http://www.cvlibs.net/datasets/kitti/raw_data.php.

17. MADDERN, Will; PASCOE, Geoff; LINEGAR, Chris; NEWMAN, Paul. 1 Year, 1000km: The Oxford RobotCar Dataset. *The International Journal of Robotics Research (IJRR)*. 2017, vol. 36, no. 1, pp. 3–15. Available from DOI: 10.1177/0278364916679498. Dataset available from: https://robotcar-dataset.robots.ox.ac.uk/datasets/.

18. PEZZEMENTI, Zachary; TABOR, Trenton; HU, Peiyun; CHANG, Jonathan K.; RAMANAN, Deva; WELLINGTON, Carl; BABU, Benzun P. Wisely; HERMAN, Herman. *Comparing Apples and Oranges: Off-Road Pedestrian Detection on the NREC Agricultural Person-Detection Dataset*. 2017. Available from arXiv: 1707.07169 [cs.CV].

19. SMITH, Mike; BALDWIN, Ian; CHURCHILL, Winston; PAUL, Rohan; NEWMAN, Paul. The New College Vision and Laser Data Set. *I. J. Robotic Res.* 2009, vol. 28, pp. 595–599. Available from DOI: 10.1177/0278364909103911.

20. GIUSTI, Alessandro; GUZZI, Jérôme; CIREŞAN, Dan C.; HE, Fang-Lin; RODRÍGUEZ, Juan P.; FONTANA, Flavio; FAESSLER, Matthias; FORSTER, Christian; SCHMIDHUBER, Jürgen; CARO, Gianni Di; SCARAMUZZA, Davide; GAMBARDELLA, Luca M. A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots. *IEEE Robotics and Automation Letters*. 2016, vol. 1, no. 2, pp. 661–667. Available from DOI: 10.1109/LRA.2015.2509024.

21. PAN, S. J.; YANG, Q. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*. 2010, vol. 22, no. 10, pp. 1345–1359. Available from DOI: 10.1109/TKDE.2009.191.

22. HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. *Deep Residual Learning for Image Recognition*. 2015. Available from arXiv: 1512.03385 [cs.CV].

23. SZEGEDY, Christian; VANHOUCKE, Vincent; IOFFE, Sergey; SHLENS, Jon; WOJNA, ZB. Rethinking the Inception Architecture for Computer Vision. In: 2016. Available from DOI: 10.1109/CVPR.2016.308.

24. SHINZATO, P. Y.; SANTOS, T. C. dos; ROSERO, L. A.; RIDEL, D. A.; MASSERA, C. M.; ALENCAR, F.; BATISTA, M. P.; HATA, A. Y.; OSÓRIO, F. S.; WOLF, D. F. CaRINA dataset: An emerging-country urban scenario benchmark for road detection systems. In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. 2016. Available from DOI: 10.1109/ITSC.2016.7795529. Dataset available from: http://www.lrm.icmc.usp.br/web/index.php?n=DataSet.Home.

25. MA, Ke; HOAI, Minh; SAMARAS, Dimitris. Large-scale Continual Road Inspection: Visual Infrastructure Assessment in the Wild. In: *Proceedings of British Machine Vision Conference*. 2017. Dataset available from: https://www3.cs.stonybrook.edu/~cvl/pavement.html.

26. CIMPOI, Mircea; MAJI, Subhransu; KOKKINOS, Iasonas; VEDALDI, Andrea. *Deep filter banks for texture recognition, description, and segmentation*. 2015. Available from arXiv: 1507.02620 [cs.CV].

27. SIMONYAN, Karen; ZISSERMAN, Andrew. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. Available from arXiv: 1409.1556 [cs.CV].

28. ARYA, Deeksha; MAEDA, Hiroya; GHOSH, Sanjay Kumar; TOSHNIWAL, Durga; MRAZ, Alexander; KASHIYAMA, Takehiro; SEKIMOTO, Yoshihide. *Transfer Learning-based Road Damage Detection for Multiple Countries*. 2020. Available from arXiv: 2008.13101 [cs.CV]. Dataset available from: https://github.com/sekilab/RoadDamageDetector.

29. HOWARD, Andrew G.; ZHU, Menglong; CHEN, Bo; KALENICHENKO, Dmitry; WANG, Weijun; WEYAND, Tobias; ANDREETTO, Marco; ADAM, Hartwig. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR*. 2017, vol. abs/1704.04861. Available from arXiv: 1704.04861.

30. ARYA, Deeksha; MAEDA, Hiroya; GHOSH, Sanjay Kumar; TOSHNIWAL, Durga; OMATA, Hiroshi; KASHIYAMA, Takehiro; SEKIMOTO, Yoshihide. *Global Road Damage Detection: State-of-the-art Solutions*. 2020. Available from arXiv: 2011.08740 [cs.CV].

31. SHANMUGAM, Divya; BLALOCK, Davis; BALAKRISHNAN, Guha; GUTTAG, John. *When and Why Test-Time Augmentation Works*. 2020. Available from arXiv: 2011.11156 [cs.CV].

32. ZUIDERVELD, K. Contrast Limited Adaptive Histogram Equalization. *Graphics Gems*. 1994, vol. IV, pp. 474–485. Available also from: https://ci.nii.ac.jp/naid/10031105927/en/.

33. REN, Shaoqing; HE, Kaiming; GIRSHICK, Ross; SUN, Jian. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. Available from arXiv: 1506.01497 [cs.CV].

34. LOWE, D. G. Object recognition from local scale-invariant features. In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. 1999, vol. 2, 1150–1157 vol.2. Available from DOI: 10.1109/ICCV.1999.790410.

35. DOLLÁR, P.; ZITNICK, C. L. Structured Forests for Fast Edge Detection. In: *2013 IEEE International Conference on Computer Vision*. 2013, pp. 1841–1848. Available from DOI: 10.1109/ICCV.2013.231.

36. ZANIN, Michele; MESSELODI, Stefano; MODENA, Carla. DIPLODOC road stereo sequence. 2013. Available from DOI: 10.13140/RG.2.1.3681.9929. Dataset available from: https://tev.fbk.eu/databases/diplodoc-road-stereo-sequence.

37. IOFFE, Sergey; SZEGEDY, Christian. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. Available from arXiv: 1502.03167 [cs.LG].

38. SRIVASTAVA, Nitish; HINTON, Geoffrey; KRIZHEVSKY, Alex; SUTSKEVER, Ilya; SALAKHUTDINOV, Ruslan. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* 2014, vol. 15, no. 1, pp. 1929–1958. ISSN 1532-4435.

39. Heartexlabs. *awesome-data-labeling* [online]. 2021. Available also from: https://github.com/heartexlabs/awesome-data-labeling/tree/70e400ccdad2bffea96d7c3b85e48180ab0e92ca.

40. CHAWLA, N. V.; BOWYER, K. W.; HALL, L. O.; KEGELMEYER, W. P. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*. 2002, vol. 16, pp. 321–357. ISSN 1076-9757. Available from DOI: 10.1613/jair.953.

41. BLAGUS, Rok; LUSA, Lara. Evaluation of SMOTE for High-Dimensional Class-Imbalanced Microarray Data. In: 2012, vol. 2. Available from DOI: 10.1109/ICMLA.2012.183.

42. LEMAÎTRE, Guillaume; NOGUEIRA, Fernando; ARIDAS, Christos K. Imbalanced learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. *Journal of Machine Learning Research*. 2017, vol. 18, no. 17, pp. 1–5. Available also from: http://jmlr.org/papers/v18/16-365.

43. CUN, Xiaodong; PUN, Chi-Man; SHI, Cheng. *Towards Ghost-free Shadow Removal via Dual Hierarchical Aggregation Network and Shadow Matting GAN*. 2019. Available from arXiv: 1911.08718 [cs.CV].

44. BRADSKI, Gary. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. 2000, vol. 25, no. 11, pp. 120, 122–125. ISSN 1044-789X. Available also from: http://www.ddj.com/ftp/2000/2000_11/opencv.txt.

45. *Anaconda Software Distribution*. Anaconda Inc., 2020. Vers. 2-2.4.0. Available also from: https://docs.anaconda.com/.

46. ABADI, Martín; AGARWAL, Ashish; BARHAM, Paul; BREVDO, Eugene; CHEN, Zhifeng; CITRO, Craig; CORRADO, Greg S.; DAVIS, Andy; DEAN, Jeffrey; DEVIN, Matthieu; GHEMAWAT, Sanjay; GOODFELLOW, I.; HARP, Andrew; IRVING, Geoffrey; ISARD, Michael; JIA, Yangqing; JOZEFOWICZ, Rafal; KAISER, Lukasz; KUDLUR, Manjunath; LEVENBERG, Josh; MANÉ, Dandelion; MONGA, Rajat; MOORE, Sherry; MURRAY, Derek; OLAH, Chris; SCHUSTER, Mike; SHLENS, Jonathon; STEINER, Benoit; SUTSKEVER, Ilya; TALWAR, Kunal; TUCKER, Paul; VANHOUCKE, Vincent; VASUDEVAN, Vijay; VIÉGAS, Fernanda; VINYALS, Oriol; WARDEN,

Pete; WATTENBERG, Martin; WICKE, Martin; YU, Yuan; ZHENG, Xiaoqiang. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Available also from: https://www.tensorflow.org/.

47. CHOLLET, Francois et al. *Keras*. GitHub, 2015. Available also from: https://github.com/fchollet/keras.

48. HUNTER, J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*. 2007, vol. 9, no. 3, pp. 90–95. Available from DOI: 10.1109/MCSE.2007.55.

49. TIELEMAN, Tijmen; HINTON, G. Divide the gradient by a running average of its recent magnitude. COURSERA Neural Netw. *Mach. Learn.* 2012, vol. 6, pp. 26–31. Available also from: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

50. MAAS, Andrew L; HANNUN, Awni Y; NG, Andrew Y. Rectifier nonlinearities improve neural network acoustic models. In: *Proc. icml*. 2013, vol. 30, p. 3. No. 1. Available also from: https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf.

51. KINGMA, Diederik P.; BA, Jimmy. *Adam: A Method for Stochastic Optimization*. 2017. Available from arXiv: 1412.6980 [cs.LG].

52. CHOLLET, Francois. Xception: Deep Learning With Depthwise Separable Convolutions. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

53. SANDLER, Mark; HOWARD, Andrew; ZHU, Menglong; ZHMOGINOV, Andrey; CHEN, Liang-Chieh. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. Available from arXiv: 1801.04381 [cs.CV].

54. HUANG, Gao; LIU, Zhuang; MAATEN, Laurens van der; WEINBERGER, Kilian Q. *Densely Connected Convolutional Networks*. 2018. Available from arXiv: 1608.06993 [cs.CV].

55. SZEGEDY, Christian; IOFFE, Sergey; VANHOUCKE, Vincent; ALEMI, Alex. *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*. 2016. Available from arXiv: 1602.07261 [cs.CV].

Appendix A

List of Acronyms

API	Application Programming Interface
CLAHE	Contrast Limited Adaptive Histogram Equalization
CNN	Convolutional Neural Network
FC	Fully connected
KNN	K-Nearest-Neighbour
R-CNN	Region Based Convolutional Neural Network
ROI	Region of Interest
SIFT	Scale-invariant feature transform
SVM	Support vector machine
TTA	Test Time Augmentation

Appendix B

Routes visualisation

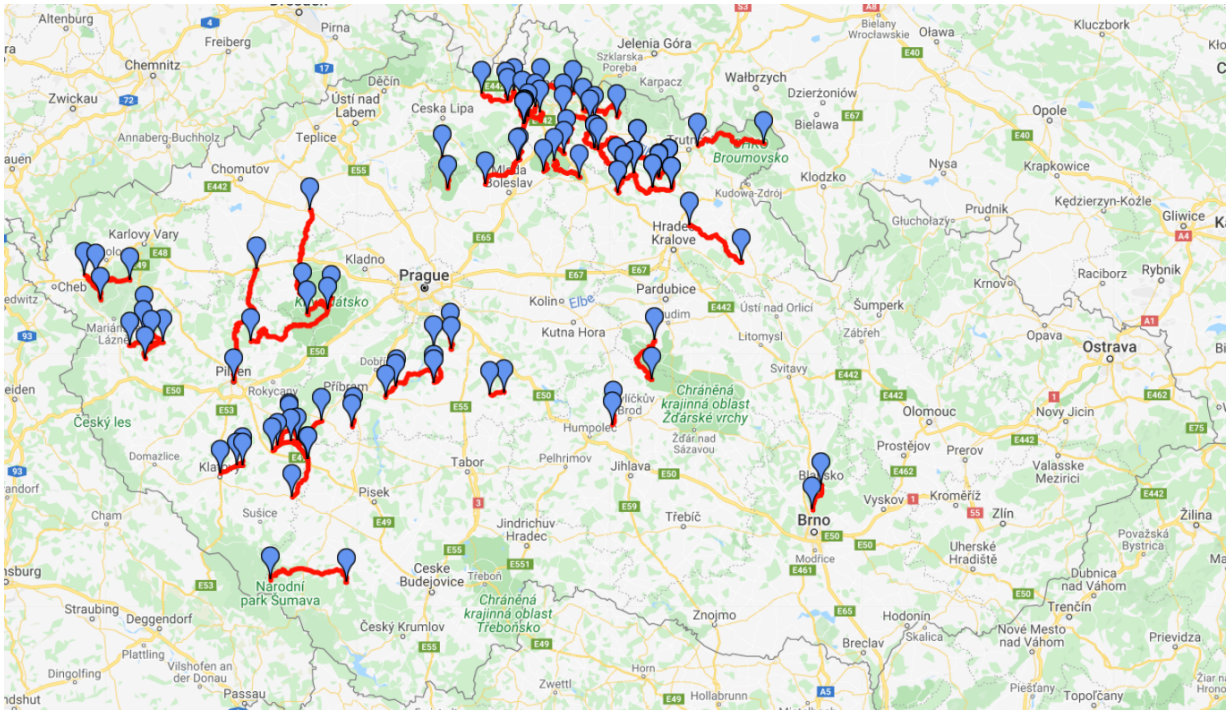

Figure B.1: Visualisation of all routes defined in this work with the total distance of 941 km. Map base: Google Maps

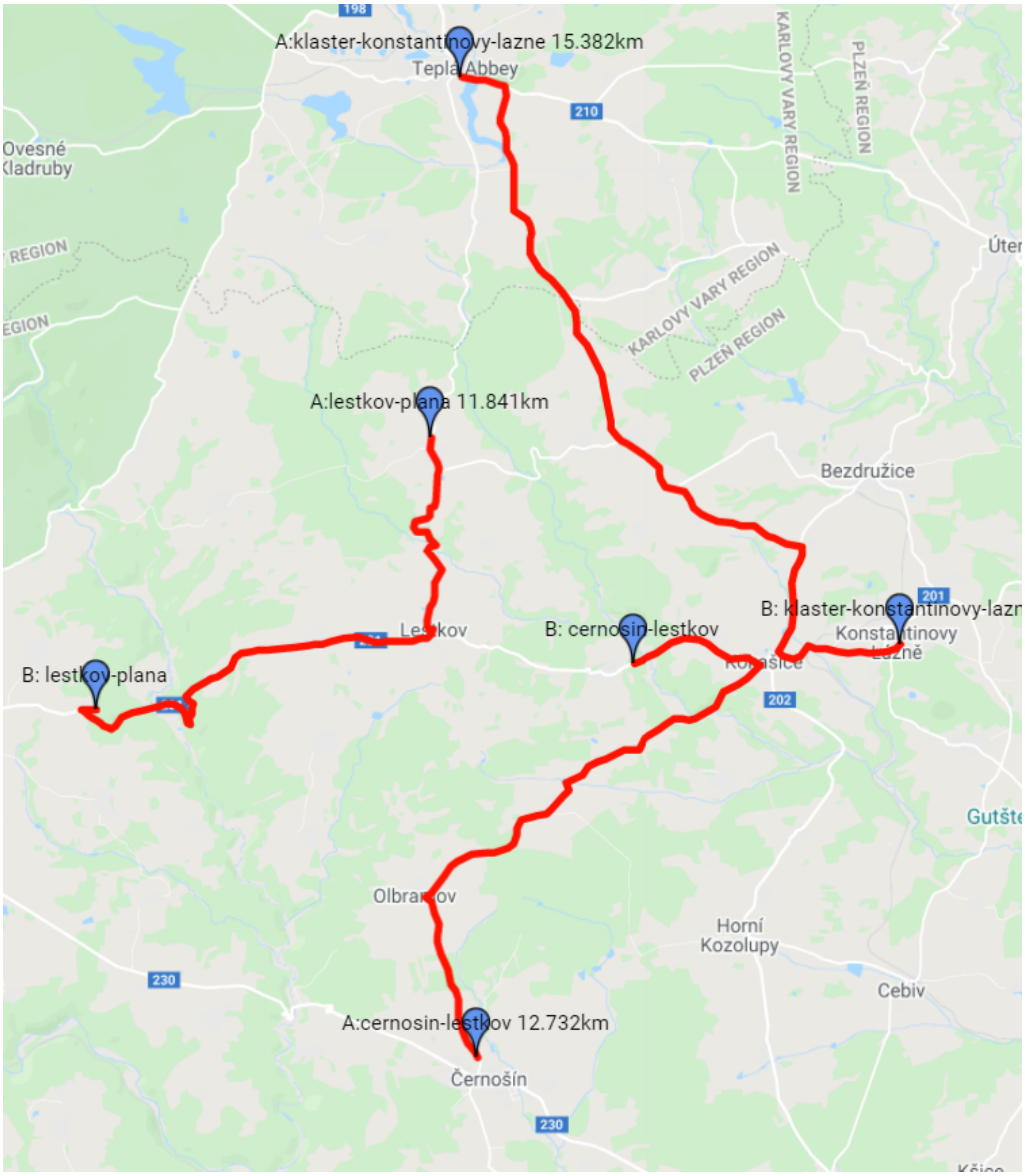

Figure B.2: Close-up visualization of three routes. We can see that each route has a marker at the start and at the end. Markers have a caption containing a letter (*A* represents the start, *B* represents the end), route *name* and route distance. Map base: Google Maps

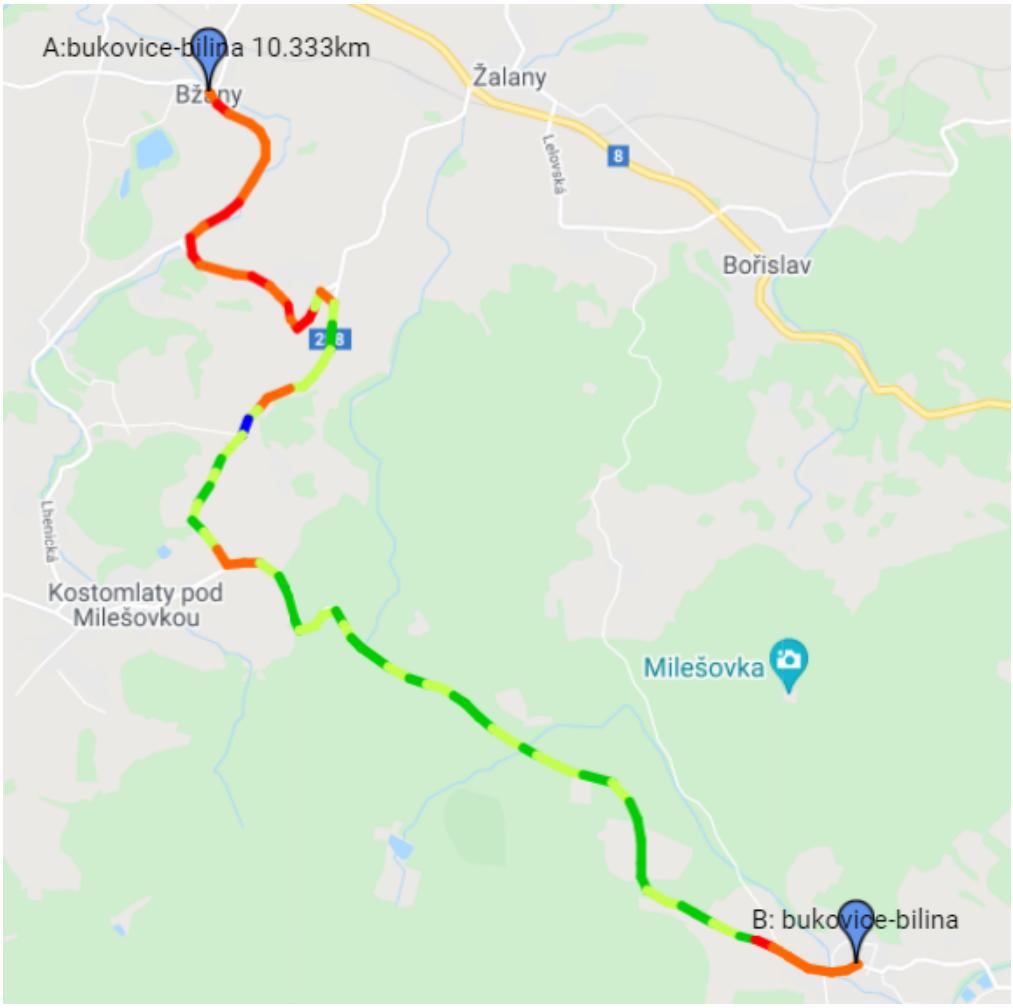

Figure B.3: This picture visualises our proposed model's predictions on a map using coloured segments. The colors represents classes as follows: green=1, green-yellow=2, orange=3, red=4, blue=6. Although the segments change their colours frequently, the quality trend is visible. Map base: Google Maps

Appendix C

Enclosed Material

- `README.md` ..... sources documentation in Markdown format
- `README.pdf` ..... sources documentation in PDF format
- `src` ..... directory with implementation files
  - `environment.yml` ..... the file with Anaconda environment definition
  - `dataset` ..... raw image data and data collection sources
    - `route_definitions` ..... directory with route set definitions
    - `routes` ..... directory with raw image files
  - `labeling` ..... labelling framework and exported datasets
  - `modeling` ..... implemented models
  - `text` ..... directory of LaTeX source codes of the thesis
- `text` ..... directory with text of the thesis
  - `assignment.pdf` ..... the assignment in PDF format
  - `BP_Lank_Martin_2021.pdf` ..... this thesis text in PDF format