Sparsely Annotated Semantic Segmentation of Weather-Related Road Surface Conditions

A Quantitative Study on the Efficiency of Different Machine Learning Models and Label Enhancement Methods

Master's thesis in Physics

Johan Rumar Karlquist

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Master's thesis 2024

Sparsely Annotated Semantic Segmentation of Weather-Related Road Surface Conditions

A Quantitative Study on the Efficiency of Different Machine Learning Models and Label Enhancement Methods

JOHAN RUMAR KARLQUIST

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2024

Sparsely Annotated Semantic Segmentation of Weather-Related Road Surface Conditions
A Quantitative Study on the Efficiency of Different Machine Learning Models and Label Enhancement Methods
JOHAN RUMAR KARLQUIST

Supervisor: Pontus Andersson, Klimator
Examiner: Mohsen Mirkhalaf, Department of Physics, University of Gothenburg

Master's Thesis 2024
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A road sign warning for slippery road ahead. Image retrieved from link to image source

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2024

Sparsely Annotated Semantic Segmentation of Weather-Related Road Surface Conditions
A Quantitative Study on the Efficiency of Different Machine Learning Models and Label Enhancement Methods
Johan Rumar Karlquist
Department of Physics
Chalmers University of Technology

Abstract

This project aims to build upon a machine learning pipeline for semantic segmentation of road conditions, focusing on classifying weather-affected surfaces such as dry, wet, slush, snow, and ice. Accurate detection of road surface conditions is crucial for autonomous driving systems and advanced driver-assistance systems as it directly influences vehicle control strategies, safety measures, and overall driving experience. Unlike most research in semantic segmentation, which relies heavily on densely annotated datasets that require significant manual labor to generate, this project utilises sparsely annotated data. These sparse labels, though containing less information, substantially reduce the need for manual annotation. Additionally, the provided data uses soft labels, representing a probability distribution over class conditions, which differs from the commonly used hard labels representing a single class. Data collection involves vehicles equipped with a front-facing camera recording the road and two laser detectors that gathers information about the road surface conditions.
Two pre-processing approaches were explored: one crops the original input image, and the other performs an image transformation to simulate a bird's-eye view of the road. Multiple new machine learning models were implemented, but it was observed that the choice of model did not significantly affect performance, indicating possible limitations in the provided data. Consequently, various approaches for augmenting data and methods to extract further information from unlabeled pixels were explored, some of which marginally enhanced performance. The pipeline's performance was evaluated using conventional metrics such as accuracy and mean intersection over union, as well as through visualisation of the resulting semantic segmentation.

Keywords: Machine learning, Computer vision, Semantic segmentation, Road climatology, Neural networks, Python

Acknowledgements

I would like to thank the collaboration company Klimator for allowing me to work on this project. Also, special thanks to my supervisor Pontus Andersson, who has helped me greatly by providing suggestions, assistance in interpreting the results, and answering questions, even though they were asked at unreasonable times of the day.

Johan Rumar Karlquist, Gothenburg, July 2024

Use of AI tools

This project has used AI as a tool to improve the writing. In particular, the thesis used the AI tool ChatGPT [1] to generate text. However, the tool was exclusively used for inspiration. None of the text generated by the AI was trusted blindly, and the thesis did not simply copy-paste these texts. Instead, the content of AI-generated text was analysed and then rewritten with sources added to support claims made. Furthermore, the AI tool Grammarly [2] was used to refine already written text with the hopes of improving the overall quality of the written report.

ChatGPT [1] was also used as a tool to generate code. In this case, the same principles were applied as with any generated text. That is, it was used for inspiration and then typically modified to suit the project's needs. Typically the AI tool was used for dull and trivial code snippets, such as code for creating a plot, to save time.

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

CNN	Convolutional Neural Network
RGB	Red Green Blue
ReLU	Rectified Linear Unit
PReLU	Parametric Rectified Linear Unit
PPM	Pyramid Pooling Module
GFLOP	Billion Floating-point Operations
PID	Proportional-Integral-Derivative
Pag	Pixel-attention-guided fusion
Bag	Boundary-attention-guided fusion
B-Head	Boundary Head
B-Loss	Boundary Loss
S-Head	Semantic Head
S-Loss	Semantic Loss
FAM	Flow-based Alignment Module
FPN	Feature Pyramid Network
ASPP	Atrous Spatial Pyramid Pooling
SGD	Stochastic Gradient Descent
Adam	Adaptive Moment Estimation
TP	True Positive
TN	True Negaitve
FP	False Positive
FN	False Negative
IoU	Intersection over Union
mIoU	mean Intersection over Union
AGMM	Adaptive Gaussian Mixture Model
GMM	Gaussian Mixtures Model
TEL	Tree Energy Loss
MST	Minimum spanning tree
bev	bird's-eye view
AI	Artificial Intelligence

Contents

List of Figures

List of Tables

1

Introduction

The road surface condition is an important variable to consider while driving. Different road surfaces, such as snowy or icy roads, increase the risk of accidents and thus compromise the safety of the driver, potential passengers, and pedestrians. According to the US Department of Transportation, weather-related road surface conditions accounted for an annual average of 1.2 million crashes, 420000 injuries, and 5400 fatalities in the US between 2007-2016 [15]. By being aware of the condition of the road, the driver can adjust their speed and driving behaviour accordingly to minimise danger.

Currently, autonomous vehicles are on the rise, and as vehicles become more and more autonomous, the importance of driver awareness has decreased. Features such as adaptive cruising control, lane-keeping assistance, and automated emergency braking have reduced the strain on the driver's concentration on the road. Consequently, the driver may overlook hazards like adverse road surface conditions. In such a case, the vehicle needs to obtain information about the driving hazards and be able to act accordingly, with or without human intervention. It is, therefore, beneficial and in the interest of road safety to develop a system that provides information about the road surface condition. This information could, for example, be utilised to alert the driver of a semi-autonomous car that there may be a need for human intervention due to the road condition. Additionally, information like that will be crucial in fully autonomous vehicles as they will need to recognise driving hazards such as adverse road surface conditions and adapt their driving behaviour by, for instance, reducing speed and increasing following distance.

Semantic segmentation refers to pixel-wise classification of an input image and is currently one of the more popular computer vision tasks [16]. Most research on the subject utilises deep learning methods trained on large datasets with accurate pixel-wise annotations, also known as dense annotations. However, such data is cumbersome to obtain as it requires a significant amount of tedious manual labour and thus limits the amount of data to be found.

To address this challenge, researchers have developed sparsely annotated semantic segmentation methods. Instead of dense annotations, which provide pixel-level labelling for every image pixel, sparse annotations utilise sparsely labelled points or scribbles. Although such data is substantially cheaper and easier to obtain than dense annotations, it still contains the least necessary information for semantic segmentation tasks [14].

The advantage of sparsely annotated semantic segmentation lies in its potential to balance information and costs effectively. By utilising sparse annotations, researchers can reduce the manual labour required for annotation while still achieving satisfactory segmentation results. This trade-off makes sparsely annotated semantic segmentation a promising research direction in computer vision.

This project is conducted in collaboration with Klimator, a world-leading company in the field of road climatology. Klimator combines machine learning, rule-based algorithms, and vast amounts of data to provide the most accurate road weather intelligence available, wherever and whenever. The company is interested in developing a machine learning pipeline, which includes data pre-processing, augmentations, a machine learning model, training, and evaluation, capable of creating semantic segmentation of road surface conditions using a sparsely annotated dataset. Although Klimator has already developed such a pipeline, they are not entirely satisfied with its performance and believe it can be optimised further.

1.1 Aim

This project aims to design, optimise, and apply a machine learning pipeline capable of predicting the condition of the road surface in front of a moving vehicle. In particular, the pipeline will utilise a sparsely annotated dataset to perform semantic image segmentation, classifying various road surface conditions. Additionally, the project aims to propose a suitable evaluation method that can assess the predictive ability of the pipeline.

1.2 Objectives

The project's initial objective is dedicated to thorough preparation, subdivided into three key aspects. Firstly, an extensive literature review will be conducted to explore existing methods employed for similar tasks. Secondly, a thorough investigation of the provided dataset is needed to understand its characteristics. Lastly, the company's existing pipeline will be reviewed to identify flaws and areas for optimisation. This objective aims to gather a substantial amount of helpful information that can later be utilised in the design and optimisation of the final pipeline.

The second and most important objective is the development of a machine learning pipeline capable of predicting the road surface condition in front of a moving vehicle. This will involve a quantitative study of how applying different data pre-processing methods, label enhancement techniques, and implementing various machine learning models affect the pipeline's performance. Furthermore, the effect of different data augmentation approaches and hyperparameters for optimisers will be studied, albeit to a lesser extent than that of machine learning models and label enhancement techniques.

The final objective aims to find an evaluation method that accurately assesses the developed pipeline's performance. This includes a comparative study employing the proposed evaluation method to analyse the effectiveness of various data manip-

ulation techniques and machine learning models. Additionally, an in-depth study of potential limitations and flaws will be carried out to ensure a comprehensive understanding of the pipeline's strengths and areas for improvement.

1.3 Limitations

The data in this thesis comes with annotations from road surface condition estimates made using laser detectors and motion estimation. Some faulty classifications in the annotations may arise due to several estimates being made in the process of generating said annotations. However, there will not be any thorough attempt to correct the faulty classifications, and all annotations will be assumed to have a correct classification. The thesis may, however, study how such faulty classifications affect the pipeline's predictive ability and identify the conditions under which the estimates of road surface conditions are likely to be incorrect.

This project will limit itself to the data provided by the company. There will not be any gathering of new data in any form, not from the internet or any own recordings. This will likely constrain the robustness of the finished pipeline, as the data may not be sufficiently generalised, potentially negatively affecting the performance when introduced to other data. However, the given data will be subjected to data augmentation, which will artificially increase the amount of available data and help mitigate some of the disadvantages introduced by this limitation.

The project will be limited to the classes introduced in the method section: dry, wet, slush, snow, and grey ice. In other words, the pipeline will only be able to predict how the weather has affected the road surface condition. It will not be capable of learning or predicting other road surface conditions, such as uneven, damaged, or pothole-ridden roads.

A significant project limitation is the computing power and the computing time. The project will test the effects of many different hyperparameters, such as data augmentation, label enhancement techniques, models, and learning rates. Testing these hyperparameters will require numerous training runs, which is time-consuming given the limited computing resources.

As such, there will only be a narrow study on the effects of the learning rate and data augmentation parameters, focusing on identifying functional values rather than performing extensive testing to find optimal values. Additionally, certain hyperparameters, such as data augmentation, will be assumed to be independent of the other parameters. Consequently, there will be limited cross-parameter testing where, for instance, each new model would entail another study to find the best data augmentations for that specific model.

Furthermore, the training of machine learning models such as these is inherently stochastic, leading to variability in results between training sessions. Ideally, multiple training sessions should be conducted, and the results of those should be averaged for a more conclusive estimate of performance. However, performing multiple tests for each parameter is not feasible due to constraints in computing power, time, and the extensive number of hyperparameters involved. Instead, the results presented

in the report will be based on one or a few training sessions, and these results will be assumed to represent the overall behaviour of the tested parameters.

2

Theory

This chapter briefly introduces machine learning, focusing on semantic segmentation. It also introduces the machine learning models and methods that will be used in the project.

2.1 Semantic segmentation

This project's central focus revolves around machine learning, more specifically, the subset of machine learning, computer vision. Computer vision encompasses various subtasks, including object detection, image classification, and semantic segmentation. While object detection and image classification are pretty self-explanatory, semantic segmentation is a bit less intuitive and serves as the project's primary task.

Semantic segmentation involves pixel-wise classification, meaning that each pixel in an image is classified [14]. In contrast, image classification consists of classifying the entire image as a single entity. For example, consider the famous MNIST dataset [3], which contains images representing handwritten digits between 0-9; see Figure 2.1 for reference. In image classification, the output would be the predicted digit. However, in semantic segmentation, each pixel in the image would be classified into a specific class or object.

To illustrate, in the context of MNIST, semantic segmentation would involve classifying which pixels belong to the digit and which belong to the background. However, this would be a trivial task since the MNIST dataset is essentially already a semantic image segmentation, where the digit pixels are white, and the background pixels are black or vice-versa.

Figure 2.1: The figure showcases example images from the MNIST dataset [3]. Image acquired from [4].

2.2 Datasets

A dataset is the main component of machine learning and the material from which the machine learns. In computer vision, datasets often consist of images with corresponding labels, typically referred to as the ground truth. One can draw parallels between schoolwork, where the images represent the tasks given to the machine, and the labels are the correct answers to those tasks.

Within semantic segmentation, there are several types of datasets. The most common type is the densely annotated dataset, such as the Cityscapes [5] and the CamVid [6] dataset. In such datasets, every pixel in the ground truth is accurately classified; see Figure 2.2. These datasets provide a lot of information for supervision, enabling machine learning models trained on them to achieve remarkable performance. However, collecting dense annotations requires significant manual labour, which limits the development of semantic segmentation. To address this problem, recent research has been done towards sparsely annotated datasets [14], [7]. These datasets use annotations such as block, scribble, or point annotations, which are cheaper to obtain while still containing the least necessary information for semantic segmentation; see Figure 2.3 for reference. The main challenge with sparsely annotated semantic segmentation is supervision, as the available information for training is relatively small compared to the densely annotated datasets.

There are also different types of annotations. Most research within the subject utilises datasets with hard labels, each corresponding to a single class. However, there are also soft labels, which represent a probabilistic distribution of possible classes. The output of a machine learning model typically comes out as soft labels but is later converted into hard ones by taking the class with the highest probability.

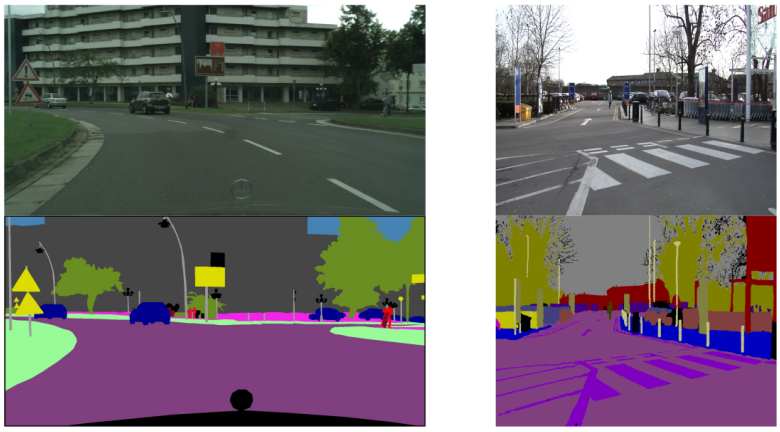


Figure 2.2: The figure shows examples retrieved from the Cityscapes [5] (left) and Camvid [6] (right) datasets. The upper image of each example is the original image used as input while training, while the lower image of each example is the ground truth used for training supervision. Do note that the ground truth images are densely annotated, where each pixel corresponds to one class represented by an individual colour. Images retrieved from [5, 6]

Figure 2.3: Different types of annotations used for semantic segmentation. b) would be classified as densely annotated data, while c), d), and e) are different types of sparsely annotated data. Image acquired from [7].

2.2.1 Data Augmentation

Data augmentation is a precious and commonly used tool while training a machine learning model. This tool artificially extends the dataset, effectively generating more data from thin air. More data is often desirable as it can significantly increase the performance of a machine learning model depending on its complexity. Data augmentation also helps prevent the model from memorising specific characteristics present only in the training data but not necessarily in new data. Consequently, it lowers the risk of overfitting to noise in the training data and enables the model to generalise new data better.

There are numerous ways to augment a dataset. Standard methods are flip, rotate, scaling, translation, colour, noise injection, and blur [17]. The choice of augmentation methods largely depends on the dataset, as the augmentations should reflect real-world factors. For instance, flipping and rotating images mimic variations in viewing angles, while adjusting brightness simulates changes in lighting. By incorporating these realistic variations into the training data, the model becomes more robust and better prepared to handle diverse real-world scenarios.

2.3 Machine learning model architecture

The architecture of machine learning models is a critical component that determines their ability to learn and, consequently, their performance and efficiency. This section briefly overviews key elements regarding machine learning models and their architecture.

2.3.1 Operations

Every machine learning model is built by combining various operations that transform input data into output data through a series of computations. The type of operation dictates what features and patterns it captures in the data. Although this section only introduces a subset of possible operations, it should sufficiently cover the machine learning models used in this project.

2.3.1.1 Convolutional operation

Convolutional operations are the fundamental components of convolutional neural networks (CNN), which are the primary focus of this report. Convolutional operations are applied on the input and allow the network to detect local patterns, such as edges, textures, and shapes [18].

The input of a convolutional operation will have the form (height)x(width)x(channels) [19]. For example, with the input of an RGB image, the height and width would correspond to the image resolution, while the channels would represent the colour channels, which for an RGB image is 3. The convolution operation is then performed using a kernel, with the kernel's size determining the convolution's receptive field [18]. A typical kernel size is 3x3xC, where C represents the number of feature channels or kernels. Each kernel element has a corresponding weight. These weights are among the learnable parameters in the model.

The kernel is applied to the input image area equal to its size during the convolution operation, as illustrated in Figure 2.4a. The operation involves computing the dot product between the pixels and the kernel's weights, resulting in an array with the length of the number of feature channels. These arrays essentially represent the summarisation of what the kernel observed within its receptive field and are known as features. The kernel shifts by a stride value and repeats the process until it has traversed the entire input. Another element in convolutional operations is padding. Padding is often applied to the input [18] to maintain spatial dimensionality between operations. The most common padding technique is zero padding, which adds a border of zeros to the input matrix. As such, an input with the spatial dimensions 3x3x1 and a padding value of 1 would become a 5x5x1 matrix where each border element is set to 0.

The final output of the convolution operations is known as a feature map, and its spatial dimensions are determined according to the following equation:

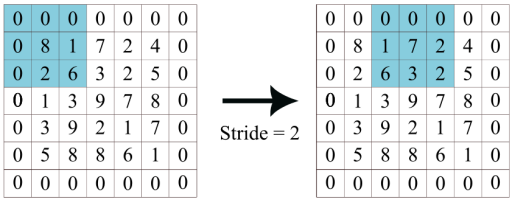
$$H_{out} = \frac{H_{in} - K_H + 2p}{s} + 1,$$
$$W_{out} = \frac{W_{in} - K_W + 2p}{s} + 1,$$

where p is the padding value s is stride value, K_H and K_W are kernel height and width.

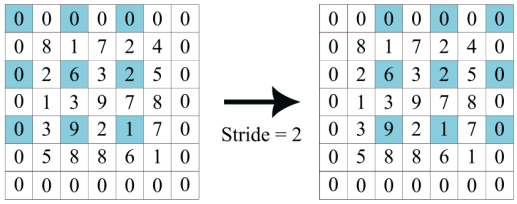
A convolutional neural network is built up by a multitude of different layers, which in turn consists of a combination of various operations, including, but not limited to, convolutional operations. There is a hierarchical function of convolutions depending on which layer they are applied in. The primary function of the convolutional operations in earlier layers is to acknowledge what is referred to as low features. These features are anything between edges, orientations, or blotches of colour. Convolutional operations in layers placed deeper within the network architecture are instead responsible for detecting more complex features, such as objects or faces [20].

Apart from the standard convolution operation introduced above, there are a few more specialised ones, each utilised for specific purposes. Dilated, or atrous, con-

volutions introduce another parameter called a dilation rate. The dilation rate determines the spacing between the values in a kernel. A dilation rate of one would indicate a standard convolutional operation. In contrast, a dilation rate greater than one would refer to a convolutional operation where the kernel parameters are spaced by a gap of dilation rate minus one, as illustrated in Figure 2.4b. Effectively, this increases the receptive field of view of the kernel without increasing the amount of parameters and computation. As such, it is a popular operation for real-time applications where one requires a wide field of view but cannot afford the computation a larger kernel would entail [21].



(a) Representation of a step in a 3x3 kernel in a convolution.



(b) Representation of a 3x3 kernel in a dilated convolution with the dilation rate of 2.

Figure 2.4: a) showcases how a 3x3 kernel is applied in a standard convolution. In b), the standard convolution is replaced by a dilated convolution with a dilation rate of 2. Both a) and b) are applied on the same 5x5 input image. The kernel is represented as light blue in the figure. In this figure, a zero padding of 1 has been applied, which can be identified by the surrounding border of zeros. Both convolution operations are also applied with a stride of 2 and a step has been done to show how the stride of an operation affects which pixels of the input image the kernel is applied on. Once the kernel has slid over the input in one direction, it will start over one stride step down. The output of the kernel is computed by a dot product of the kernel's weight and the input values. Note that in actual neural networks there is usually a depth to the input, which is not shown in the figure. The outputs of said kernels are also excluded, but the output size here would be 3x3 for the standard convolution and 2x2 for the dilated convolution. Images created by the author.

A transposed convolutional operation, also known as deconvolution or fractionally stridden convolution, is another type of convolution. In contrast to the standard convolutional operation that, assuming padding is not applied, performs downsampling, i.e reduces the spatial dimensionality of the input, the transposed convolution is utilised for upsampling, generating an output of greater spatial dimensionality than the input.

To achieve this upsampling, a transposed convolution inserts zeros into the input, effectively expanding its spatial dimensionality. This requires two new parameters z and p', where z stands for zero padding and p' for effective padding. These are calculated from the kernel size k, padding p and stride s according to: $z = s - 1$, $p' = k - p - 1$. The z value determines the amount of zeros inserted between the

rows and columns of the input, while p' represents the padding used during standard convolutions. The kernel is then applied to the adjusted input with a stride value equivalent to 1 for standard convolutions. This results in an output with greater spatial dimensionality than the input, where the output dimensionality is dependent on the values of z and p' [22, 21]. See Figure 2.5 for a visual representation.

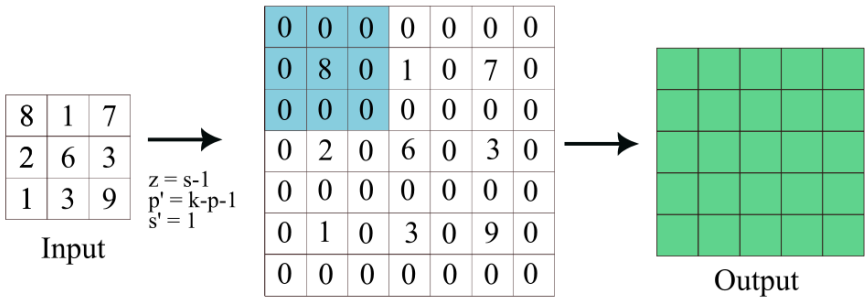


Figure 2.5: Illustration of a transpose convolution operation with kernel size 3x3, stride $s = 2$ and padding $p = 1$. Note that zeros are inserted between each input value according to the value of $z = s - 1 = 2 - 1 = 1$, and a border of zeros is added according to $p' = k - p - 1 = 3 - 1 - 1 = 1$. In transpose convolutions, the step size s' of the kernel is always 1. As such, the input with a spatial dimensionality of 3x3 is upsampled to an output with a spatial dimensionality of 5x5. Image created by the author.

2.3.1.2 Batch normalisation

A batch normalisation [23] is a widely used operation in convolutional neural networks that normalises the input of each neuron to have zero mean and unit variance [20, 18]. This serves as a stabilising factor during the learning process and prevents the problem of internal covariate shift, which refers to the phenomenon where the distribution of each layer's input changes during training as the parameters of the previous layers change [23]. Batch normalisation typically occurs between convolutional operations and activation functions, which will be discussed in the next subsection. Batch normalisation contributes to several beneficial factors when training CNNs, such as accelerated training process, better regularisation, improved generalisation, and less sensitive hyperparameters [20, 18, 23].

2.3.1.3 Activation functions

Activation function's main purpose is to help the network learn and approximate continuous and complex relationships between variables of the network [24, 20, 18]. Activation functions are such an integral part of CNNs that they are sometimes omitted from network architectures as it is assumed that an activation follows a convolutional operation after normalising it. Commonly used activation functions are rectified linear unit (ReLU), parametric rectified linear unit (PReLU) [25], sigmoid and hyperbolic tangent [24, 18, 9]. The activation function is what transforms

the input into a meaningful representation of the data and, as such, is a fundamental component in neural networks [20].

2.3.1.4 Dropout

Dropout is introduced as a form of regularisation with the intent to mitigate overfitting the model on the provided training data. Drodout forces some neurons in the input to randomly disconnect, as in setting them to zero, with a certain probability. This ensures that the model learns multiple independent representations of the input and does not become too reliant on specific neurons, which encourages the network to learn more robust and generalised features. Dropout is typically applied after the fully connected layers on a CNN [20, 18, 24].

2.3.1.5 Pooling operation

The primary function of pooling operations is to reduce spatial dimensionality, which reduces the parameter number of the input and, as such, the computation needed in the network. Similarly to the convolutional operation, the pooling operation uses a kernel to process the input. However, unlike the kernel in convolutions, the pooling kernel does not have any weights attached to it. Instead, the kernel represents an aggregation function that is applied to the values found within its receptive field.

Pooling comes in two different types, each corresponding to a unique aggregation function: max pooling and average pooling. The max pooling function simply uses the maximum pixel value as output, while the average pooling outputs the average pixel value of the input. In addition to dimensionality reduction, the max pooling function serves as a noise suppressant as it discards noisy activations. Max pooling is typically employed in the middle layers of a CNN, whereas average pooling is usually used in the final layers of the network. Note that while the pooling operation inflicts a lot of lost information, it contributes to reduced model complexity and improved efficiency and limits the risk of overfitting, making it an essential factor in convolutional neural networks [18, 20, 19, 24].

Another notable pooling method within semantic segmentation is the pyramid pooling module (PPM). This module aims to incorporate features from multiple scales, providing a richer representation of the input image. The PPM takes feature maps as input and performs multiple pooling operations in parallel with different kernel sizes to capture information at various scales. The outputs of each pooling operation are subjected to a 1x1 convolution and then upsampled via bilinear interpolation, such that each pooled feature map has the same spatial dimensionality as the original feature map. Subsequently, all feature maps, including the original one, are concatenated to form the output of the module as depicted in Figure 2.6. This pooling technique enhances the network's ability to capture contextual information across different scales, thereby improving semantic segmentation performance [8].

2.3.1.6 Fully connected layer

As the name suggests, the fully connected layer is a type of layer where each neuron from the previous layer is connected to each neuron of the current layer, making it

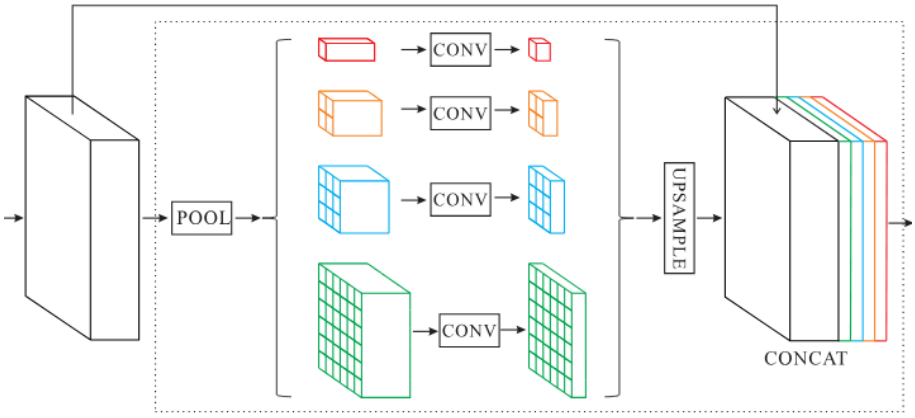


Figure 2.6: An overview of the pyramid pooling module. The original feature map, the leftmost block, is subject to multiple pooling operations at different scales. A 1x1 convolution is applied to the resulting pooled feature maps, which are then upsampled through bilinear interpolation to retain the spatial dimensionality of the original feature map. All feature maps are then concatenated to create the module's output. The image accessed from [8].

fully connected. These layers are typically found towards the end of a convolutional neural network and are responsible for generating the final output predictions.

The output of a fully connected layer is computed by applying a weight matrix and a bias vector to the input, followed by an activation function, such as average pooling used in the ResNet neural network [12]. A softmax function is commonly applied to the output of this layer, and consequently to the output of the entire neural network, to receive a probability distribution across classes [19, 18, 20, 24].

2.3.2 Encoders, decoders, backbones and segmentation heads

Some neural networks adopt an encoder-decoder structure. These networks are utilised for their ability to learn sequence-to-sequence. Their architecture consists of two parts: the encoder and the decoder.

The primary function of the encoder is to capture features in the input data by downsampling it and encoding these features into feature maps. This process typically involves the application of several consecutive convolutional operations with corresponding batch normalisation and activation functions. The term "encoder" refers to its role in encoding observed features into numerical representations within the feature maps.

In contrast, the decoder upsamples the feature maps generated by the encoder to match the spatial dimensionality of the original input data [26]. This is often achieved by applying transpose convolution operations, as seen in models like ENet [9], or bi-linear interpolations, as seen in models like PIDNet [10]. Essentially, the decoder uses the feature maps to reconstruct the original image and, in the context of semantic segmentation, classifies each pixel according to its features.

The encoder and decoder terms are closely related to the terms "backbone" and "segmentation heads", where a backbone network is often used as the encoder in encoder-decoder architectures. The backbone's primary task is to extract features, much like an encoder's primary task. Conversely, the segmentation head is designed explicitly for semantic segmentation tasks and takes the extracted features from the backbone network and processes them to produce pixel-wise predictions.

2.4 Machine learning models

2.4.1 ENet

The already existing pipeline provided by the company utilises the machine learning model ENet [9] due to its simplicity and, perhaps more importantly, the small number of parameters and its intuitive design. This model will serve as a baseline for the project. As such, the performance of any other models and methods employed will be compared to the performance of ENet with the hopes of achieving better results.

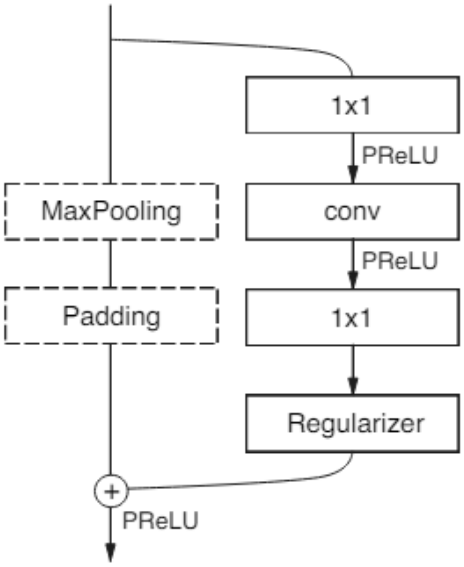
As previously stated, the ENet model's most significant advantage is its relatively small size. The model contains merely 0.37M parameters and requires only 3.83 GFLOPs (billion floating point operations). These are significantly lower numbers than those of other machine learning models, such as SFNet-R18 and PIDNet-S, which will also be used in the project. These models require 247 and 47.6 GFLOPs and 12.87M and 7.6M parameters, respectively.

The architecture of ENet can be seen in Figure 2.7a. The initial block applies max pooling and a 3x3 convolution with a stride of 3 on the input, which is then concatenated to create the output. The subsequent blocks, or bottlenecks as they are referred to, each consist of three different convolutional operations depending on the type of bottleneck and come in 5 different stages. The first convolutional operation is a 1x1 projection that reduces the dimensionality, the second is the main convolutional operation that depends on the bottleneck type, and the third is a 1x1 expansion. The structure of the bottlenecks can be seen in Figure 2.7b. Batch normalisation and PReLU [25] are applied between each convolution. Spatial dropout is used as a regulariser, where the parameters depend on which stage the bottleneck is in. For the downsampling bottleneck type, the first projection convolution is replaced by a 2x2 convolution with a stride of 2 in both dimensions.

A max pooling operation is added to the main branch, as seen in Figure 2.7b. The main convolutional layer is either a regular, dilated or transposed convolution with 3x3 kernels. These are sometimes replaced with asymmetric convolutions where the kernel size is 5x1 and 1x5, resulting in a 5x5 convolution. Padding is added to maintain the spatial dimensionality within each stage. Lastly, a full convolution, which is the equivalent of a fully connected transpose convolution, is applied to receive the final output.

Name	Type	Output size
initial		$16 \times 256 \times 256$
bottleneck1.0	downsampling	$64 \times 128 \times 128$
$4\times$ bottleneck1.x		$64 \times 128 \times 128$
bottleneck2.0	downsampling	$128 \times 64 \times 64$
bottleneck2.1		$128 \times 64 \times 64$
bottleneck2.2	dilated 2	$128 \times 64 \times 64$
bottleneck2.3	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.4	dilated 4	$128 \times 64 \times 64$
bottleneck2.5		$128 \times 64 \times 64$
bottleneck2.6	dilated 8	$128 \times 64 \times 64$
bottleneck2.7	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.8	dilated 16	$128 \times 64 \times 64$
Repeat section 2, without bottleneck2.0		
bottleneck4.0	upsampling	$64 \times 128 \times 128$
bottleneck4.1		$64 \times 128 \times 128$
bottleneck4.2		$64 \times 128 \times 128$
bottleneck5.0	upsampling	$16 \times 256 \times 256$
bottleneck5.1		$16 \times 256 \times 256$
fullconv		$C \times 512 \times 512$

(a) The overall architecture of the ENet machine learning model.



(b) Insight into the structure of a bottleneck.

Figure 2.7: a) The model consists of seven different sections. The initial stage processes the input and the final stage creates the output. The five stages in between consist of bottleneck blocks that, in turn, consist of three different convolutional layers with batch normalisation and PReLU applied between each layer. b) An overview of the structure of each bottleneck. The ”conv” layer is bottleneck type dependent, and the 1x1 layers correspond to projection and expansion convolutional operations. Padding is added to ensure a maintained spatial dimensionality within a stage, and max pooling is added for the downsampling bottleneck type. Additionally, while downsampling, the first 1x1 projection layer will be replaced with a 2x2 convolution with stride 2 in both dimensions. Images acquired from [9].

2.4.2 PIDNet

Another machine learning model implemented in the project is the PIDNet [10]. The name is inspired by PID controllers, which are Proportional-Integral-Derivative controllers used in modern dynamic systems. The model is mainly designed for real-time semantic segmentation tasks and achieves state-of-the-art results on the CamVid [6] and Cityscapes [5] datasets, which are commonly used in semantic segmentation.

The PIDNet network consists of three branches: P, I, and D, each with different responsibilities. The P branch focuses on parsing and preserving detailed information in high-resolution feature maps. The I branch parses long-range dependencies by aggregating context information locally and globally. The D branch predicts boundary regions by extracting high-frequency features. The depths of the P, I, and D branches are set to moderate, deep, and shallow, respectively, contributing to an efficient implementation. There are three different models in the PIDNet family: small, medium, and large, which are generated by adjusting the depth and width of

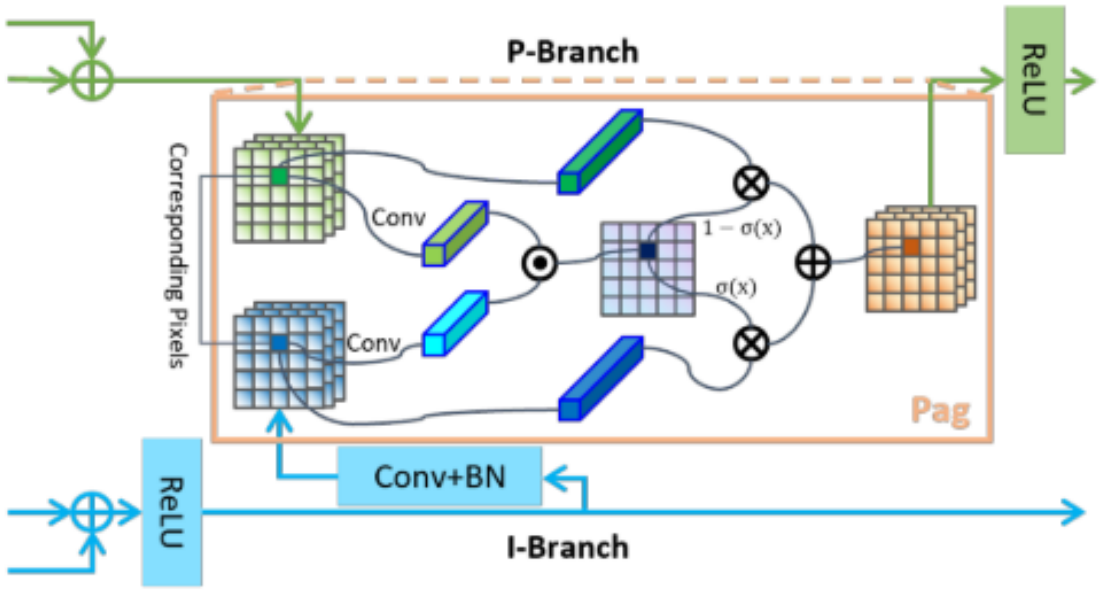


Figure 2.8: The overall structure of the Pag module. The input feature maps from the P and B branches are initially processed to have matching spatial dimensionality. Then, the feature vectors of corresponding pixels from the feature maps are processed via convolutions and subjected to a sigmoid function introduced in equation (2.2). The final output of the Pag module depends on the outputs of the sigmoid function according to equation (2.3). All depicted convolutions in the figure are 1x1. The image is retrieved from [10].

the model.

PIDNet also introduces two additional modules called Pixel-attention-guided fusion (Pag) and Boundary-attention-guided fusion (Bag). The overall structure of the Pag module can be seen in figure 2.8. It is used to fuse information from the P and I branches by convolutions and sigmoid activators. The sigmoid function and the output of the Pag module can be represented mathematically as:

$$Sigmoid(x) = \frac{1}{1 + e^{-x}}, \tag{2.1}$$

$$\sigma = Sigmoid(f_p(\vec{v_p}) \cdot f_i\vec{v_i}), \tag{2.2}$$

$$Out_{Pag} = \sigma\vec{v_i} + (1 - \sigma)\vec{v_p}, \tag{2.3}$$

where f indicates feature maps from the I and P branches denoted by the subscript i, p respectively. $\vec{v}$ represents the feature vectors for corresponding pixels between feature maps of the I and P branches. This module effectively leverages which of the branches provides the most reliable information. A high value of σ indicates that the model trusts the information given by the I branch, while a low value suggests that the model puts more trust in the information provided by the P branch.

The Bag module is placed towards the end of the network, and its primary function is to fuse the high-frequency and low-frequency areas from the D branch with detailed features from the P branch and context features from the I branch, respectively. Similar to the Pag module, this is achieved through a series of convolutions and

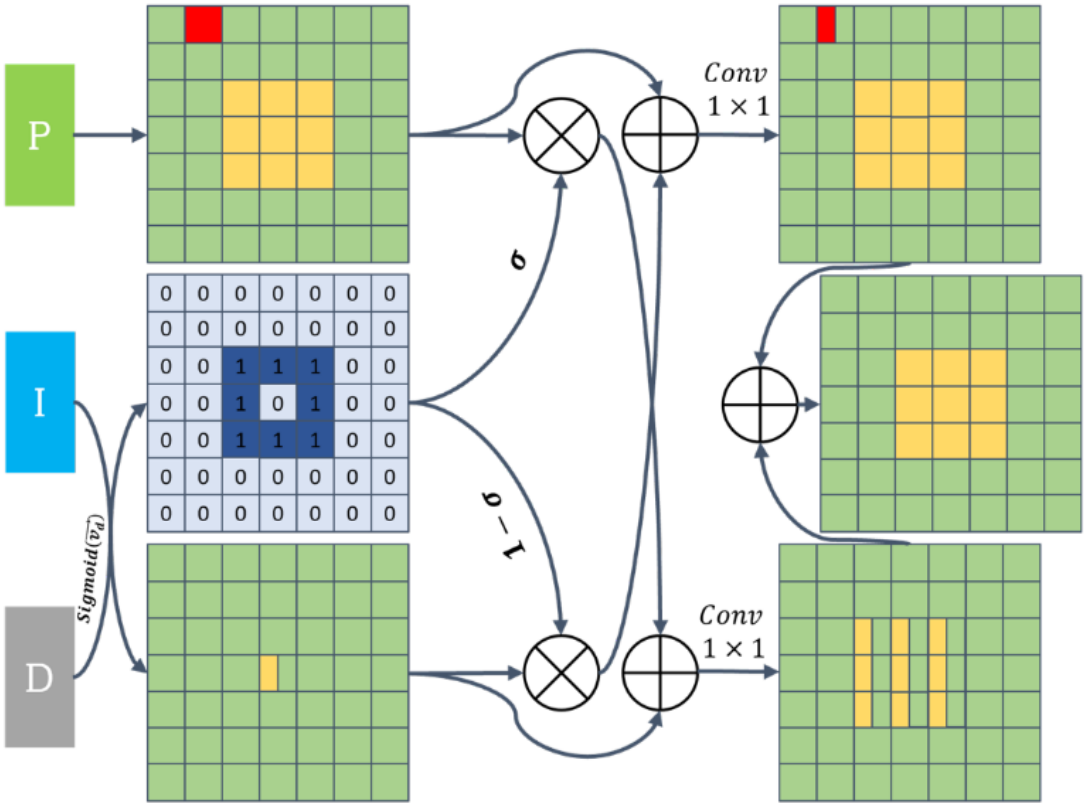


Figure 2.9: The overall structure of the so-called Light-Bag module implemented in the smaller version of PIDNet. The figure shows how bad semantics, coloured in red, from the P branch, are eventually singled out by putting a larger trust in the context information provided by the I branch. The procedure is done through a series of sigmoid activations and convolutional operations mathematically represented in Equation (2.5). The image is retrieved from the original authors of PIDNet [10].

sigmoid activations. The structure of the Bag module varies depending on the size of the Network. The small and medium-sized networks use what is referred to as the light Bag module, whose structure is depicted in Figure 2.9. The module's function can be expressed mathematically as:

$$\sigma = Sigmoid(\vec{v_d}), \tag{2.4}$$

$$Out_{light_{bag}} = f_p((1-\sigma) \otimes \vec{v_i} + \vec{v_p}) + f_i(\sigma \otimes \vec{v_p} + \vec{v_i}), \tag{2.5}$$

where f refers to the composition of convolutions, batch normalisation, and ReLU activation functions. The value of σ indicates the amount of trust the model places in the detailed features compared to the context information. If $\sigma > 0.5$, the model will the detailed features more. This fusion mechanism allows the Bag module to effectively combine the strengths of both detailed and contextual information, leading to a better overall segmentation performance by enhancing feature representation and improving boundary accuracy.

The overall architecture of PIDNet is shown in Figure 2.10. Compared to the previously introduced Enet architecture [9] and the other architectures that will be

introduced, PIDNet is more complex due to the separation and cooperation of three distinct branches. Additionally, PIDNet employs S/B-Heads (Semantic-/Boundary-Heads) to create outputs used for loss calculations. The structure of these can also be found in the top right corner of Figure 2.10.

The network can be divided into six different stages apart from the three different branches. The first two stages are the initial stages, where feature maps are created from the input image. In stages 2-4, the network branches out. In each stage, the input is processed by a series of convolutional operations and corresponding batch normalisation and function activations, which together generate feature maps. The I branch's feature map is provided to both the P and D branches via lateral connections. In the P branch, a Pag module processes the feature maps from both the I and P branches. The corresponding feature maps are merged in the D branch through element-wise summation. The output of the stage 2 P branch is also processed through an S-Head, which calculates the semantic loss (S-Loss) l_0. Stage 3 is implemented similarly to stage 2, except that instead of the S-head processing the output of the P branch, there is a B-head that processes the output of the D branch, which is used to calculate the boundary loss (B-Loss) l_1. Stage 4 similarly continues feature extraction to the previous stages, without any added S/B-head. In stage 5, the final output of each branch is processed. While the output of the P and D branches are sent directly to the Bag module, the output of the I branch first undergoes a PPM. The output of the Bag module is processed by an S-Head, which results in the final semantic segmentation.

The PIDNet model divides their loss function into four parts: l_0, l_1, l_2 and l_3. l_0 and l_2 represent the commonly used cross-entropy loss but appear at two different stages. l_0 is the semantic loss from the output of the first Pag head, while l_2 is the semantic loss of the final output of the model. The l_1 and l_3 losses originate from the boundaries, where l_1 represents a weighted binary cross-entropy loss for boundary detection and l_3 is a boundary-awareness cross-entropy loss. The total loss L is then calculated by:

$$L = \lambda_0 l_0 + \lambda_1 l_1 + \lambda_2 l_2 + \lambda_3 l_3, \tag{2.6}$$

where the parameters are empirically set by the authors as $\lambda_0 = 0.4$, $\lambda_1 = 20$, $\lambda_2 = 1$, $\lambda_3 = 1$ [10]. Figure 2.10 give more insight into how It is important to note that the PIDNet models are designed for densely annotated datasets, such as CamVid [6] and Cityscapes [5], which differ from the provided sparsely annotated dataset used in this report. Therefore, boundary regions are completely irrelevant in this project because there are none, making the D branch of the network less impactful. As a result, the l_1 and l_3 losses orginating from boundary regions will not be used in this project. Instead, the total loss will be fully dependent on l_0 and l_2, where the initial proportions suggested by the developers of PIDNet will be used, namely $L = 0.4 \cdot l_0 + l_2$. These proportions may be adjusted if a different ratio achieves better results.

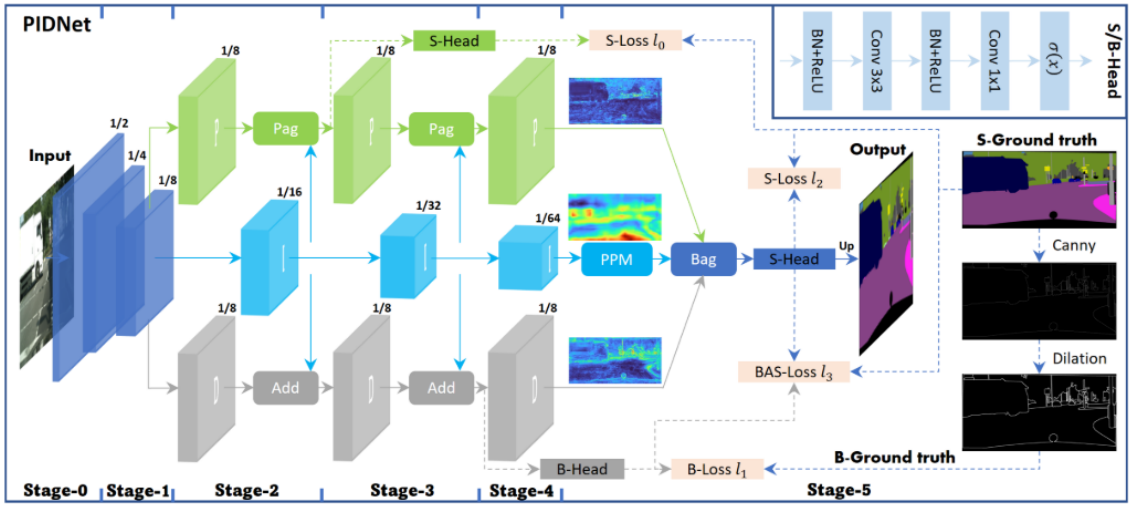


Figure 2.10: The figure presents an overview of the architecture of PIDNet. Each "block" consists of a series of convolutional operations with corresponding batch normalisation and activation functions. The spatial dimensionality of the feature maps in each block is represented as a fraction of the spatial dimensionality of the input image above each block. The blocks in stages 2-4 are labelled according to which branch they are a part of. Lateral connections between the I branch and the two other branches are represented as blue arrows. In the figure, S and B denote semantic and boundary, while Add and Up refer to element-wise summation and bilinear upsampling, respectively. Image acquired from [10]

2.4.3 ResNet

In this thesis, the ResNet [12] model was extensively used as a backbone network for several different segmentation models. ResNet utilises residual connections, which mitigate the vanishing gradient problem, enabling the training of very deep networks. However, due to computing efficiency, this project will focus most on one of the shallower versions of ResNet, namely ResNet-18, where 18 represents the number of residual blocks in the model.

The fundamental building block of ResNet is the residual block, which can be represented by the following equation:

$$\boldsymbol{y} = F(\boldsymbol{x}, W_i) + \boldsymbol{x}, \tag{2.7}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ represent the input and output vectors of the block, respectively, and F represents the residual mapping to be learned.

The architecture of ResNet-18 is shown in table 2.1. The initial layer applies a 7x7 convolution with 64 feature channels, stride 2 and 3 zero padding, followed by batch normalisation, ReLU activation, and a 3x3 max pooling layer with stride 2. After the initial layer comes the residual layers consisting of residual blocks arranged in pairs, each with two 3x3 convolutions. The first layer has 64 feature channels, and the amount of feature channels doubles for each subsequent layer, resulting in a final layer with 512 feature channels. An average pooling layer is applied to the output of the residual block layers, reducing the spatial dimensions to 1x1. The results are then passed to a fully connected layer that provides the model predictions. Note that batch normalisation is applied after each convolutional operation, and the ReLU activation function is applied at the end of each layer. Additionally, downsampling with a stride of 2 is performed in each layer's first residual block.

Table 2.1: Architecture of ResNet-18. The network is divided into six layers: The initial layer, the four residual block layers and the final layer. In each layer, the input is downsampled according to the output size in the table. The table also presents the operations performed in each layer, where A x A, B represents a convolutional operation with B kernels, with kernel sizes of A. Note that layers 1-4 include batch normalisation and activation functions, which are excluded in the figure. Table created by the author with information from [12]

Layer Name	**Type / Operation**	**Output Size**
Input	Image	$224 \times 224 \times 3$
conv1	7×7, 64, stride 2	$112 \times 112 \times 64$
bn1	Batch Normalisation	$112 \times 112 \times 64$
relu	ReLU Activation	$112 \times 112 \times 64$
maxpool	3×3 Max Pooling, stride 2	$56 \times 56 \times 64$
layer1	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$56 \times 56 \times 64$
layer2	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$28 \times 28 \times 128$
layer3	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$14 \times 14 \times 256$
layer4	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$7 \times 7 \times 512$
avgpool	Global Average Pooling	$1 \times 1 \times 512$
fc	Fully Connected (num_classes)	$1 \times 1 \times$ num_classes

2.4.4 SFNet

The developers of SFNet [11] have thoroughly investigated the concept of optical flow to align two adjacent video frame features in the video processing task. This idea led to the proposal of a flow-based alignment module (FAM), which predicts the flow field inside the network to align feature maps of adjacent levels. This concept, referred to as "Semantic Flow", is generated between different levels in a feature pyramid.

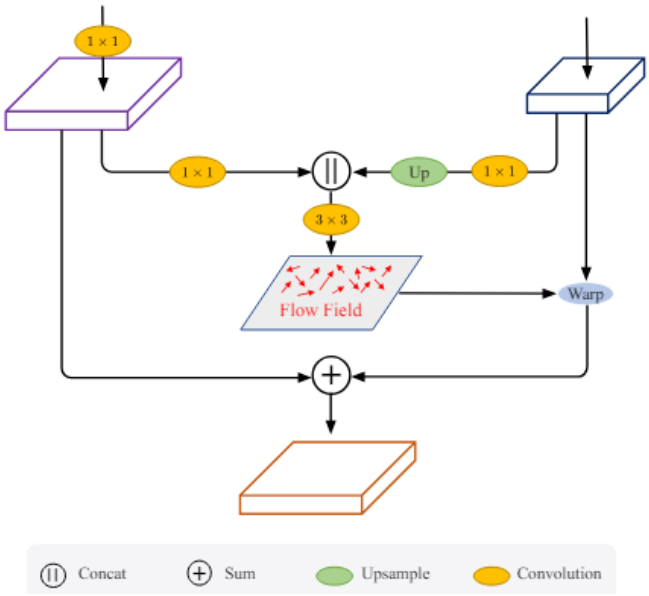


Figure 2.11: An overview of the underlying operations done in the FAM module developed by [11]. Two adjacent feature maps, represented by purple and blue blocks for upper and lower levels, are convoluted and upsampled to a matching spatial dimensionality and then concatenated. The concatenated feature map is further convoluted to create the semantic flow field, which is used to perform a warping operation according to equation (2.9) and (2.10). The feature map of the warped points and the upper-level feature map are then summed together to create the output of FAM. Image acquired from [11].

FAMs are placed within the decoder of the network. It is integrated between feature map levels, where the feature maps are compressed into the same channel depth as the upcoming level via two 1x1 convolutions. Given two adjacent feature maps F_n and F_{n-1} with the same channel number, a bi-linear interpolation layer is used for upsampling the feature map F_n to the same size as F_{n-1}. The feature maps are then concatenated together, and the result is used as an input for a sub-network with two 3x3 convolutional layers, as shown in Figure 2.11. The sub-network's output is the predicted semantic flow field, which can mathematically be written as

$$\Delta_{n-1} = conv_n(cat(F_n, F_{n-1})), \tag{2.8}$$

where Δ_n is the semantic flow field of level n, $conv_n(\cdot)$ stands for the convolution operation, $cat(\cdot)$ represents the concatenation. The semantic flow field Δ_{n-1} is then

used to map each position p_{n-1} on the spatial grid Ω_{n-1} to a warped point p_n on the upper level n via the following operation:

$$p_n = \frac{p_{n-1} + \Delta_{n-1}(p_{n-1})}{2}. \tag{2.9}$$

The final output of the FAM $\tilde{F}_n(p_{n-1})$ is then approximated by linearly interpolating the 4-neighbours (top-left, top-right, bottom-left, bottom-right) of p_n with the following bi-linear sampling mechanism:

$$\tilde{F}_n(p_{n-1}) = F_n(p_n) = \sum_{p \in N(p_n)} w_p F_n(p), \tag{2.10}$$

where $N(p_n)$ represents the neighbours of the warped points p_n in feature map n, and w_p are bi-linear kernel weights estimated by the distance of the warped grid. Much like ENet [9], SFNet is subdivided into an encoder and a decoder. The encoder consists of a backbone network, such as the ResNet series [12], with their final fully connected layer removed. Additionally, a pyramid pooling module (PPM) [8] is adopted at the end of the backbone network.

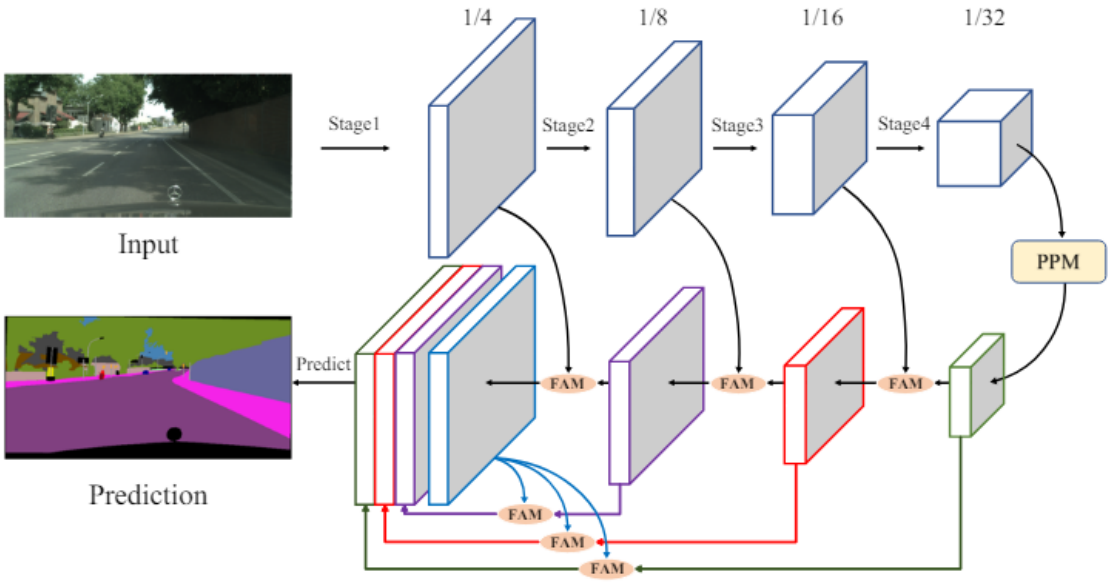


Figure 2.12: An overview of the SFNet architecture. The top part represented by stages represents the encoder, or bottom-up pathway, where each stage corresponds to the layers within the used backbone network, which would be the residual blocks if ResNet is used as the backbone [12]. A PPM module is applied to the output of the backbone network, and the result is sent as input into the decoder, or top-down pathway, represented as the bottom part in the figure. The decoder upsamples the input feature maps with the help of FAM modules and lateral connections, which provide the feature maps from the corresponding stages of the encoder. Image retrieved from [11].

The decoder follows the feature pyramid network (FPN) architecture [27]. An FPN consists of two main pathways and lateral connections between them. The bottom-

up pathway extracts feature maps at different levels and is essentially the encoder of the whole architecture. The top-down pathway upsamples feature maps from the final output of the encoder and merges the result with corresponding feature maps from the bottom-up pathway using the lateral connections. However, in SFNet, the FPN is modified by replacing bi-linear upsampling with the proposed FAM. Figure 2.12 provides an overview of the network's architecture while also representing an FPN.

2.4.5 Deeplabv3plus

The Deeplabv3plus model [13] is a semantic segmentation model that has achieved state-of-the-art performance on datasets such as [5]. This model extends the fundamental deeplabv3 model [28] by employing an encoder-decoder structure. The encoder of deeplabv3plus consists of a backbone network that extracts low-level features and an atrous spatial pyramid pooling (ASPP) module to extract higher-level features. The ASPP module performs parallel atrous, or dilated, convolutions at different dilation rates and concatenates their outputs to extract high-level features.

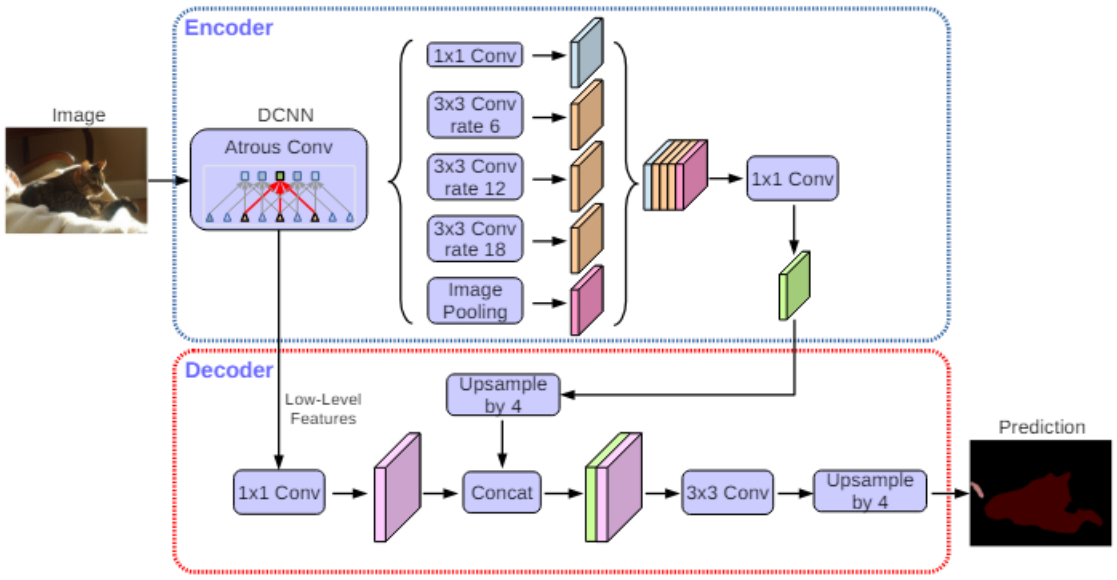


Figure 2.13: The figure showcases the architecture of the encoder-decoder structure of Deeplabv3plus. Note that the backbone network that creates the low-level features is omitted in this figure. The backbone network extracts low-level features that are further processed through an ASPP module to extract high-level features. 1x1 convolutions are applied to both the low- and high-level features, where the latter is upsampled to match the dimensionality of the low-level features. The low-level and high-level features are then concatenated and go through another 3x3 convolution. The output of this convolution is further upsampled to create the output prediction image. Image retrieved from [13].

The high- and low-level features are sent to the decoder, where they are upsampled to share the same spatial dimensions and concatenated. A convolution operation

is then performed on the concatenated features, and the result is upsampled to match the spatial dimensionality of the original input image, resulting in semantic segmentation of the input image. For reference, see Figure 2.13. Note that in Deeplabv3plus, upsampling does not entail a transposed convolution; instead, the upsampling is done through bi-linear interpolation.

2.5 Training

This section will introduce how training is performed in machine learning. However, there are various sub-branches within machine learning, and in this project, the focus will specifically be on supervised learning.

2.5.1 Supervision

In the training phase of machine learning, the dataset is used to teach the chosen model. The dataset consists of images with corresponding labels. These images are fed as input to the model, and the functions in the model's structure extract features from the image that are then used for classification. In semantic segmentation, the output would be pixel-wise classification over the whole image used as input.

The model's output is compared to the labels in the dataset to determine how well the model performed. This comparison is done by applying different loss functions that calculate the so-called loss of the model. A high loss would indicate that the output does not correspond well with the labels. One of the most common loss functions is cross-entropy loss. In multi-class classification, the cross-entropy loss in each pixel is formulated as

$$Loss = -\sum_{i=1}^{N} y_i \cdot log(p_i), \tag{2.11}$$

where N is the total amount of classes, y_i is the ground truth probability for each class, and p_i is the predicted probability for class i, ranging from 0 to 1. Note that for hard labels, y_i would be 1 for the correct class and 0 for the rest. The $Log(x)$ function, where x ranges from 0 to 1, returns a negative number approaching 0 when $x = 1$ and $-\infty$ when $x = 0$. Cases where the predicted probability of the correct class is low would, therefore, result in a large loss, while a large predicted probability for the correct class would result in a small loss.

This loss is then propagated back to the model by an optimiser, such as gradient descent or similar, where the model updates its parameters in a direction that minimises the loss function. The parameter update is also proportional to the learning rate. The learning rate is usually set up to decrease over time, which means that the model updates its parameters less aggressively over time. A decaying learning rate allows the model to learn quickly but coarsely in the beginning and slowly but precisely towards the end. A learning rate that is too high would lead to the parameter overshooting the optima in each update, while a learning rate that is too low would cause the model to never converge to the optima.

2.5.1.1 Optimisers

Optimisers are the algorithms that handle the gradients created from the loss functions, which are then used to adjust the parameters in the model. The most common optimisers, which this project will stick to, are stochastic gradient descent (SGD) and adaptive movement estimation (Adam).

Stochastic gradient descent is an extension of the well-known gradient descent algorithm. Instead of computing gradients over the entire dataset, like the gradient descent algorithm, SGD randomly selects a smaller subset of the dataset and only computes the gradient upon the data in this smaller subset. This procedure significantly reduces the computing power required for larger datasets and thus substantially decreases the time needed for each iteration, but also results in a slightly slower convergence rate. To accelerate the convergence rate it is common to use momentum. Momentum uses information from the previous time step to smooth out variations and thus accelerate convergence rate [29].

Adam is another stochastic optimisation algorithm that works much in the same way as SGD. However, a key difference between the two optimisers is that Adam adopts an adaptive learning rate based on the first and second moments of the gradients [30]. In comparison, SGD uses either a fixed learning rate throughout the training or a learning rate scheduler that adjusts the learning rate during the training phase according to some equation. Due to the adaptive nature of Adam's learning rate, it is less sensitive to the choice of learning rate and other hyperparameters, making it more user-friendly. However, according to Karpathy [31], the SGD will almost always slightly outperform the Adam optimiser if one manages to find a learning rate that is, at the very least, close to optimal.

Another common method used in optimisers is weight decay, a form of L2 regularisation that limits the size of weight updates. This method helps prevent the model from overfitting the training data by adding a penalty term to the loss function. This penalty term is proportional to the square of the magnitude of the weights and encourages the model to learn simpler patterns by penalising large weights, thereby reducing its complexity and preventing it from fitting the training data too closely [32].

Both Adam and SGD compute the gradients using smaller subsets of the dataset, typically called batches. The most common approach is to divide the whole dataset into batches and then sequentially compute each batch's loss and corresponding gradients until the entire dataset is processed. One iteration of this is called an epoch and is usually repeated for a fixed amount of times or until a satisfactory result has been achieved. Between each epoch, the data in the dataset is shuffled to avoid order biases that could hurt the model's learning.

2.5.1.2 Test and validation

In machine learning, datasets are typically divided into two or three parts: training, validation, and test data. As the name suggests, the training data is used to train the model and assist the learning through supervised techniques, as explained in

section 2.5.1. Validation and test data, on the other hand, are used to evaluate how well the model has learned. These datasets are separate from the training data. They are not used during the training process, meaning the model receives no feedback and does not perform any parameter updates based on its performance on validation and test data. This separation helps ensure the model does not overfit the training data by learning undesired characteristics, such as noise.

Training progress is often monitored by comparing the loss curves from both the training and validation data. Other loss-dependent metrics, such as accuracy, may also be used for this purpose. The expected behaviour of the training loss curve is to decrease monotonically over epochs in a negative exponential manner. The validation curve, however, usually decreases until it reaches a minimum and then starts increasing. This pattern indicates two phases: the phase before the validation loss increases and the phase after. The optimal point to stop training is just before the validation loss begins to increase. Stopping too early would result in underfitting, while stopping too late would lead to overfitting. See Figure 2.14 for reference.

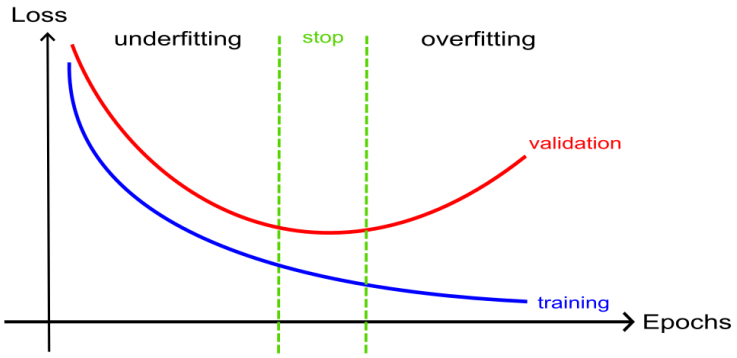


Figure 2.14: The figure displays a common comparison of the training and validation loss curves. The y-axis represents loss, and the x-axis epochs. The optimal point to stop training would be between the two green dashed bars. Stopping before the first green bar would result in an underfit model, while stopping after the second green bar would result in an overfit model. Image created by the author.

Underfitting occurs when a model is too simple to capture the underlying characteristics of the dataset, leading to poor performance on both the training data and any unseen data. The model fails to learn enough from the dataset, resulting in inadequate predictions. In contrast, overfitting occurs when the model is too complex for the training data and begins to learn undesired characteristics, such as noise. While this results in excellent performance on the training data, it causes poor performance on new, unseen data. It is, therefore, crucial to divide the dataset into training and validation data, allowing evaluation of the model's performance on both seen and unseen data.

The model should be tested on validation data at intervals to assess its performance on unseen data, for instance, once every epoch. The model should be trained while the training and validation dataset loss decreases. However, suppose the loss for the

validation dataset starts increasing. In that case, that is a sign that the model has begun to overfit the data, and the training should be terminated lest the model starts to learn noise in the training dataset instead of the desired underlying patterns.

2.5.2 Evaluation

The performance of a machine learning model is evaluated on data that it has not yet encountered during training, typically on validation or test data.

There are several different evaluation methods that are each suitable for various tasks, but most of them are dependent on the concept of true and false positives and negatives. True positives and negatives are cases where the machine model has correctly predicted a class. In contrast, false positives and negatives are cases where the model has incorrectly predicted a class. In the context of this project, false positives would, for example, be instances where the model predicts the dry class, but the ground truth indicates another class. False negatives occur when the model fails to predict a dry class, even though the ground truth is dry. True positives are instances where the model's prediction matches the ground truth, and true negatives are cases where the model correctly predicts a class other than the dry class when the ground truth is also not dry.

In this project, there will be five different classes, and in this setting, the true negatives are less impactful than in binary classification. One can derive many different evaluation methods from the basic concept of true and false positives and negatives. Arguably, the most common metric is accuracy, which measures the percentage of correct predictions out of the total amount of predictions. For a specific class i, the accuracy can be calculated as follows:

$$Accuracy_i = \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i}, \tag{2.12}$$

where TP, FP, and FN are true positives, false positives, and false negatives, respectively.

Other common evaluation methods in machine learning are precision, recall, and F1-score. These are built similarly using the concept of true and false positives and negatives. For instance, precision for class i would be calculated by

$$Precision_i = \frac{TP_i}{TP_i + FP_i}, \tag{2.13}$$

and recall for class i is

$$Recall_i = \frac{TP_i}{TP_i + FN_i}. \tag{2.14}$$

The F1 score is then a combination of both precision and recall and can be calculated by

$$F1_i = \frac{2 \cdot Precision_i \cdot Recall_i}{Precision_i + Recall_i}. \tag{2.15}$$

Precision indicates the accuracy of the positive predictions, and recall measures the model's ability to find all positives. The F1 score is a metric that combines recall and precision and can be seen as a trade-off between the two [33].

A confusion matrix is a common method to visualise true and false positives and negatives. In binary classification, the confusion matrix has four elements corresponding to each of these parameters see Figure 2.15. However, this project will focus on multi-class classification, and in this context, the confusion matrix has a slightly different form.

In multi-class classification, the confusion matrix becomes more complex due to the presence of multiple classes rather than a single one. The diagonal elements of the matrix represent true positives for each class, indicating the number of correct predictions for each specific class. The matrix rows correspond to false positives, indicating how often instances of other classes are incorrectly predicted as a particular class. Conversely, the columns correspond to false negatives, showing how often instances of a specific class are incorrectly predicted as other classes. This structure helps in understanding the model's performance across all classes, as illustrated in Figure 2.16.

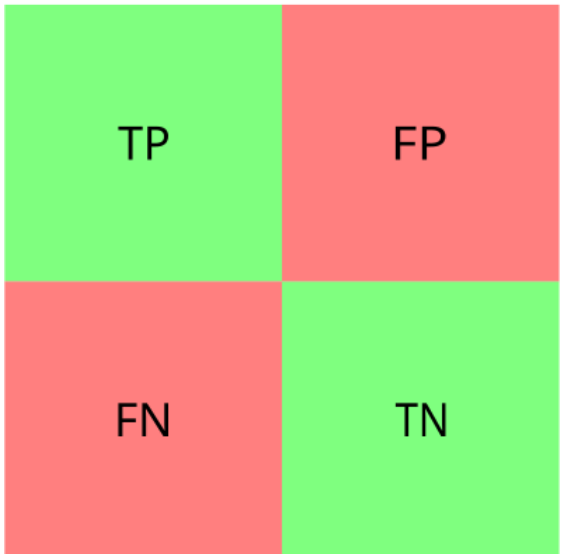


Figure 2.15: Confusion matrix for binary classifications. TP, FP, FN, and TN stand for true positives, false positives, false negatives and true negatives, respectively. Image created by the author.

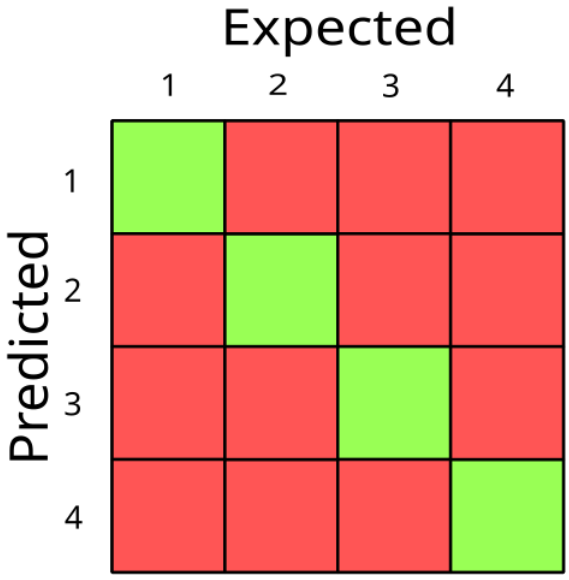


Figure 2.16: Confusion matrix for multi-class classifications. The green diagonal represents correct classifications, that is, true positives, while the red squares represent incorrect classifications, that is, false negatives and positives. In this example, there are four different classes. The numbers on the left-hand side and on top of the matrix represent the classes. That is, the element in row and column one of the matrix corresponds to the true positive value of class 1. The element 12 of the matrix would represent cases where the ground truth is class 2, but the model predicted class 1. As such, red squares in row i can be interpreted as false positives of class i and red squares in column i can be interpreted as false negatives of class I. Image created by the author.

The project aims to develop a model that can accurately perform semantic segmentation of the road surface condition in an input image. In semantic segmentation, a commonly used evaluation metric is the intersection over union (IoU).

IoU is calculated by dividing the area of overlap between the predicted classes and the ground truth by the area of union [34], as illustrated in Figure 2.17. A high IoU indicates a strong agreement between the predicted and ground truth classes, with a value of 1 indicating perfect overlap. Figure 2.18 provides examples of IoU measurements in a binary classification scenario. However, This project revolves around multi-class classification, which requires the calculation of the mean IoU when evaluating model performance. To calculate the IoU, one can utilise the aspects of true and false positives or equivalently, as they are based on those aspects, confusion matrices. The IoU would then be calculated by:

$$IoU_i = \frac{TP_i}{TP_i + FP_i + FN_i}. \tag{2.16}$$

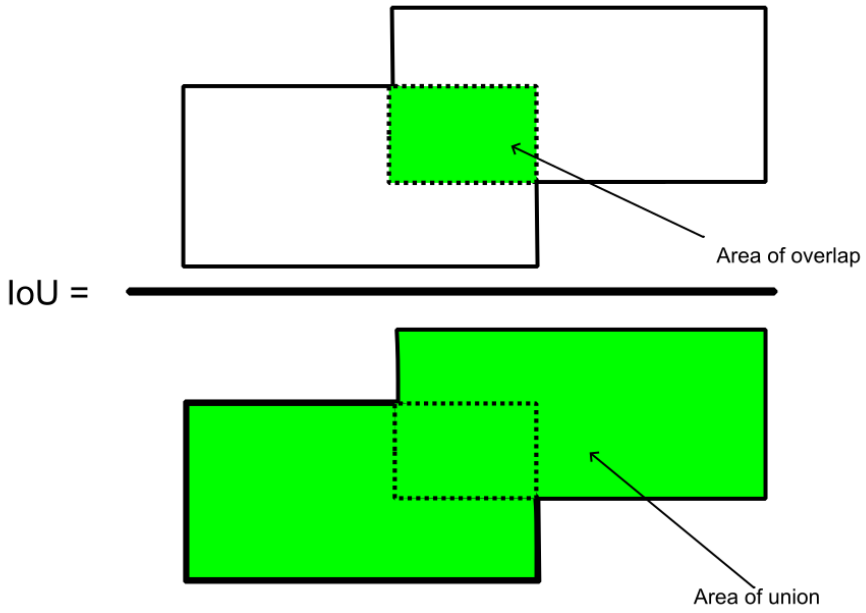


Figure 2.17: The figure graphically shows how IoU is calculated. That is, the IoU is the area of overlap divided by the area of union. Image created by the author.

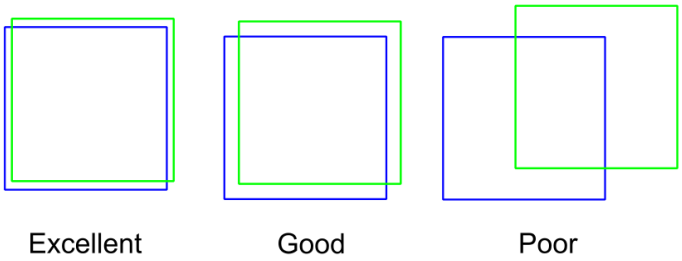


Figure 2.18: The figure displays examples of IoU calculations and evaluates them as excellent, good or poor. In the leftmost example, the area of overlap is almost the same as the area of union, and therefore, the IoU is high. In the rightmost example, the area of overlap is relatively small compared to the area of union, which results in a poor IoU. Image created by the author.

2.6 Sparsely annotated semantic segmentation methods

2.6.1 AGMM

AGMM, the Adaptive Gaussian Mixture Model, is a method developed to bridge the gap between a densely labelled dataset and a sparsely annotated one for supervision [14]. The method's main idea is to leverage the similarity between labelled and

unlabelled pixels in an image to generate predictions for the latter. These predictions can then be used to add supervision to the unlabelled regions by incorporating them into the loss function rather than assigning pseudo labels, which is otherwise a popular method [14].

The generated predictions of the unlabelled pixels should be probabilistic since generated hard one-hot predictions are inappropriate due to the added noise that will harm performance. Therefore, [14] proposes using Gaussian mixtures to represent the class distribution and uses the learned features of labelled pixels as centroids of the generated Gaussian mixtures. In simpler terms, all pixels in the image are divided into clusters in feature space, where the mean features of each labelled class act as the centroids of said clusters. Probabilistic predictions are then generated for the unlabelled pixels depending on how far away they are from the centroids created by the labelled pixels, see Figure 2.19 for reference.

The method uses several additional loss functions apart from the ordinary cross-entropy. Firstly, the mean features μ_i of the i_{th} class are calculated by:

$$\mu_i = \frac{1}{|x_{li}|} \sum_{\forall x \in x_{li}} f(x), \tag{2.17}$$

where $f(x)$ are the features of pixel x and x_{li} is a vector of all pixels labelled by the class i. The GMM predictions G are then calculated using Gaussian mixtures as such:

$$G = \sum_{i}^{K} g_i(x, \mu_i, \sigma_i) = \sum_{i}^{K} e^{-\frac{d^2}{2\sigma_i^2}}, \tag{2.18}$$

where K represents the classes in an input image, d the distance between $f(x)$ and the mean μ_i and σ_i the variance of class i. However, the main article found that the value of σ did not impact the results in a significant way and put it as 1, which is what this report will do as well. The GMM predictions G are then used for self-supervision with the prediction P in a cross-entropy manner to form the first loss L_{self}:

$$L_{self} = -\frac{1}{|x|} \sum [Glog(P) + (1 - G)log(1 - P)]. \tag{2.19}$$

The sparse labels y_l are also used to supervise the GMM predictions as:

$$L_{spar} = \frac{1}{|y_l|} \sum_{\forall y \in y_l} ylog(G). \tag{2.20}$$

The paper [14] also proposes a contrastive loss to enlarge the distance between the centroids of different Gaussian mixtures:

$$L_{con} = \frac{2}{K(K+1)} \sum_{\forall i,j \in K, i \neq j} e^{-(\mu_i - \mu_j)^2}. \tag{2.21}$$

All in all, we have a total GMM loss formulated by $L_{GMM} = L_{self} + L_{spar} + L_{con}$. Consequently, the total loss becomes $L = L_{seg} + L_{GMM}$, where L_{seg} is the ordinary cross-entropy loss between the model's output and the ground truth in labelled pixels.

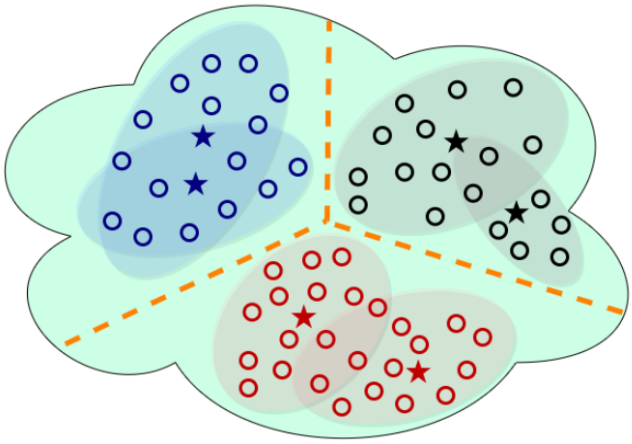


Figure 2.19: A visualisation of how pixels are assigned to their corresponding Gaussian mixtures using AGMM. The cloudlike object represents feature space, the stars are labelled pixels, and the circles are unlabelled pixels. Both unlabelled and labelled pixels are coloured according to their corresponding class. What is not shown in the figure is that the mean features of the labelled pixels are used as centroids of each Gaussian mixture. The unlabelled pixels are assigned to a class represented by a Gaussian mixture with a probability distribution depending on the distance to the mixture's centroid. Image retrieved from [14]

2.6.2 Tree Energy Loss

Tree Energy Loss (TEL) is another method intended to bridge the gap between densely and sparsely annotated datasets by providing supervision from unlabelled pixels. The main concept of TEL is to create pseudo labels for the unlabelled pixels in the image in an effort to increase the amount of supervision available while training. This is done by dividing the sparsely annotated image into two parts, the labelled set of pixels Ω_L and the unlabelled set Ω_U. While one can use the ground truth labels for training with the labelled set, that is impossible for the unlabelled ones. Instead, TEL uses the fact that pixels corresponding to the same object often share similar patterns at various feature levels. By leveraging these patterns, the method is able to provide additional supervision via generated soft pseudo labels for the unlabelled set. These pseudo labels are utilised to calculate a tree energy loss L_{tree}, which can be combined with the segmentation loss acquired from the labelled set,

$$L_{tot} = L_{seg} + \lambda L_{tree}, \tag{2.22}$$

where λ is used as a balance factor. The segmentation loss is calculated by the usual cross-entropy loss discussed in Section 2.5, specifically Equation (2.11). The tree energy loss can be divided into three substeps: (1) A tree affinity generation step to model the pair-wise relation. (2) A cascaded filtering step to generate the pseudo. (3) A soft label assignment step to assign pseudo labels for unlabelled pixels.
The main concept of the tree affinity generation is to represent the image as an undirected graph $G = (V, E)$, where all pixels make up the vertice set V, and the edge set E corresponds to the edges between two adjacent vertices. In the TEL method, one uses the architecture of a 4-connected planar graph, where each pixel

is adjacent to up to 4 neighbouring ones. Low- and high-level weight functions can then be calculated using

$$\omega_{i,j}^{low} = \omega_{j,i}^{low} = |I(i) - I(j)|^2,$$
$$\omega_{i,j}^{high} = \omega_{j,i}^{high} = |F(i) - F(j)|^2,$$

where $I(i) \in \mathbb{R}^{3\times h\times w}$ and $F(i) \in \mathbb{R}^{3\times h\times w}$ are the RGB and the semantic features of pixel i respectively. h and w are the height and width of the downsampled input image.

The semantic features are extracted by applying a 1x1 convolution on the features before the classification layer of the decoder. By sequentially removing the edge with the largest weight from E, one can create a minimum spanning tree (MST) for both the low- and high-level. These MSTs are constructed by utilising the Borůvka algorithm. Based on the topology of MST, it is possible to determine whether vertices share similar feature representations, as such vertices would appear within the same object and tend to interact with each other preferentially. The distance between two vertices is then calculated by summing the weights of their connected edges. The distance map D of the MST consists of the distance of the shortest path between vertices, i.e., the hyper-edge $\mathbb{R}$,

$$D_{i,j}^{*} = D_{j,i}^{*} = \sum_{(k,m)\in\mathbb{E}_{i,j}^{*}} \omega_{k,m}^{*}, \tag{2.23}$$

where i, j, k and m are vertice indexes and $* \in low, high$. The distance maps are then projected into positive affinity matrices,

$$A^{low} = \exp(-D^{low}/\sigma),$$
$$A^{high} = \exp(-D^{high}/\sigma),$$

where σ is a preset constant parameter that modulates the colour information. The low-level affinity matrix A^{low} contains object boundary information, while the high-level affinity matrix A^{high} maintains semantic consistency.
A cascaded filtering strategy F is introduced to generate the pseudo labels $\tilde{Y}$ from the network predictions P and the affinity matrices:

$$\tilde{Y} = F(F(P, A^{low}), A^{high}). \tag{2.24}$$

The network prediction P is the network output after a softmax operation. The filtering operation F is defined as:

$$F(P, A^{*}) = \frac{1}{z_i} \sum_{\forall j\in\Omega} A_{i,j}^{*} P_j, \tag{2.25}$$

where $\Omega = \Omega_L \cup \Omega_U$ is the full set of pixels and $z_i = \sum_j A_{i,j}$ is a normalization term. Note that the pseudo labels generated with cascaded filtering can preserve sharper semantic boundaries, which are significant in semantic segmentation but are lacking in sparse annotations. This could lead to a boost in semantic performance.
The soft labels are then assigned by a simple function that measures the distance

between a pseudo label $\tilde{Y}$ and the predicted probability P. The authors of TEL have opted for an L1 distance, which results in the following equation for the tree loss L_{tree}:

$$L_{tree} = -\frac{1}{|\Omega_U|} \sum_{\forall i \in \Omega_U} |P_i - \tilde{Y}_i|. \tag{2.26}$$

3

Methods

3.1 Preparation

The first few weeks of the project were used as a preparation stage. This stage was subdivided into three key aspects: literature review, study of the current pipeline, and inspection of the dataset.

3.1.1 Literature Review

The aim of the extensive literature review was to explore existing methods and models that have been used for similar tasks. Consequently, the focus was laid on articles that had performed semantic segmentation. In particular, the project explored research of machine learning models designed for semantic segmentation, some of which were subsequently implemented into the project's pipeline. Additionally, the literature review extended to articles containing different data augmentation methods, an essential tool for a better generalised pipeline and a common approach to artificially extend a dataset and increase the performance of machine learning models.

A significant limitation of the project is the provided sparsely annotated dataset. While sparse annotations require less manual labour than dense annotations, they also contain less information, which constrains the model's learning process. Additionally, most machine learning models for semantic segmentation are developed with densely annotated datasets in mind and, therefore, contain features that exclusively work for such datasets, causing further challenges. Consequently, a portion of the literature review revolved around finding research on methods that bridge the gap between sparsely and densely annotated datasets by providing supervision from unlabeled pixels as well.

3.1.2 Inspection of the data

The data consists of images recorded by a front-facing camera mounted on moving vehicles. These images are annotated using laser detectors placed underneath said vehicle. The laser detectors are based on infrared spectroscopy to provide point-wise estimated probabilistic distribution over the road surface condition underneath the vehicle. The estimates are generated by processing the output of the laser detectors with a laser model developed by Klimator, the collaboration company. The road

surface condition estimates are located via motion estimation to annotate them into the images recorded by the front-facing cameras, resulting in sparsely annotated images as illustrated in Figure 3.1. In this project, these images will be subjected to data pre-processing and then serve as input to the machine learning model, where the point-wise road condition estimates will be used as labels for supervision. The image frame has a resolution of 2880x1860.

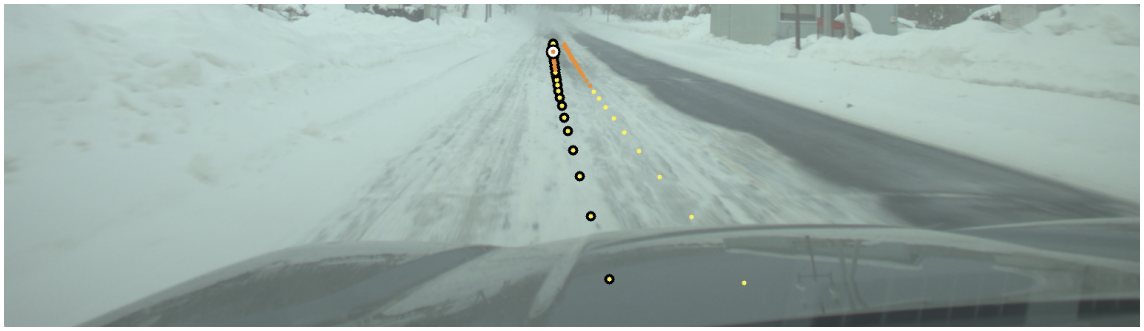


Figure 3.1: A part of an example image in the dataset that will be used in the project. Note that the image is cropped so that it only shows the road with annotations. The conditions with the highest probability in the point-wise road surface condition estimations are annotated as coloured circles in the image. The yellow and orange colours correspond to the road surface conditions of snow and grey ice. Image by author.

The data provided by the company are divided into sessions, typically containing a few thousand consecutive frames, corresponding to 10 minutes of recording time. These recordings come from various collaboration companies based in different countries, with different vehicles, on different parts of the day, and sometimes even varying laser detector setups. All of these factors affect the expected road surface condition. It was, therefore, essential to perform a comprehensive study of the provided data with the aim of finding a combination of sessions that can act as a dataset for the pipeline.

The most crucial factor in this study was to study the distribution of road surface conditions in the data sessions so that the resulting dataset would contain a well-balanced representation of different road surface conditions under varying circumstances. Another factor was to acknowledge flaws in the data and reflect on how such flaws can affect the performance of the final pipeline.

In the end, two different datasets were selected. The first dataset is relatively small and contains sessions exclusively recorded by a single collaboration company. No additional manual session selection was used to create this dataset. Instead, it included all sessions recorded by the collaboration company. The main advantage of the first dataset is that it consists of data deemed to be of the highest quality.

In contrast, the second dataset is a combination of sessions recorded by a multitude of different companies, including sessions present in the other dataset. The main advantage of the second dataset is that it is larger than the first and provides a greater diversity of images and conditions, which should result in a more robust

machine learning model. However, the overall quality of the second dataset is lower than that of the first. Even though it contains more frames recorded in more diverse scenarios, the individual frames are less interesting and often contain homogeneous data where all annotations belong to a single condition.

The first dataset contained roughly 16000 training frames and 2080 evaluation frames, while the second dataset contained roughly 32000 training frames and 4000 evaluation frames.

3.1.3 Flaws in data

3.1.3.1 Motion estimation

While the annotations in Figure 3.1 are essentially perfectly made, there are cases where the motion estimations fail, resulting in annotations that are significantly worse. This can happen when the camera-equipped vehicle is either driving upwards or turning, especially in small turns. These cases result in annotations that are way off the vehicle's actual path, and sometimes, the annotations are not even on the road at all. These cases are showcased in Figures 3.2 and 3.3 for driving upwards and turning, respectively.

Figure 3.2: The figure shows two cases where the equipped vehicle is driving up a small hill/bridge. In both figures, some road surface condition annotations are located in the air. The motion estimation does not take vertical motion into account and could potentially cause a lot of problems as the pipeline uses these annotations to learn classification. That is, it may associate a dry road surface condition with the characteristics of trees or sky. Image by author.

3.1.3.2 Traffic

Another flaw in the dataset is data collected in traffic. The motion estimation does not account for obstacles such as vehicles in front of the camera. As a result, annotations are made as if the road is completely clear of traffic, leading to annotations appearing on top of vehicles ahead instead of on the road. Figure 3.4 illustrates two examples of where this anomaly occurs. To address this issue, another machine learning model is employed that detects vehicles in the image, and any annotations within the bounding boxes of detected vehicles are removed. However, this results in

Figure 3.3: The figure shows annotation problems while the vehicle is turning. In the left figure, the vehicle is driving in an urban landscape, which includes a lot of smaller turns. The motion estimation utilises the vehicle's yaw rate. Therefore, smaller turns with a temporarily big yaw rate will cause the motion estimation to assume a large change of direction. As seen in the figure, situations like these can cause the annotations to be completely off the vehicle's actual path. In the right figure, the vehicle is performing a larger turn. The annotations close to the vehicle are reasonable, but the annotation method still struggles to locate the annotations further away. Image by author.

data frames that contain considerably fewer labels, which further limits the amount of information given to the model and, as such, limits the model's ability to learn. Additionally, images with traffic in them will still occur in validation images. They will sometimes result in weird classifications, where the model tries to classify road conditions on top of vehicles.

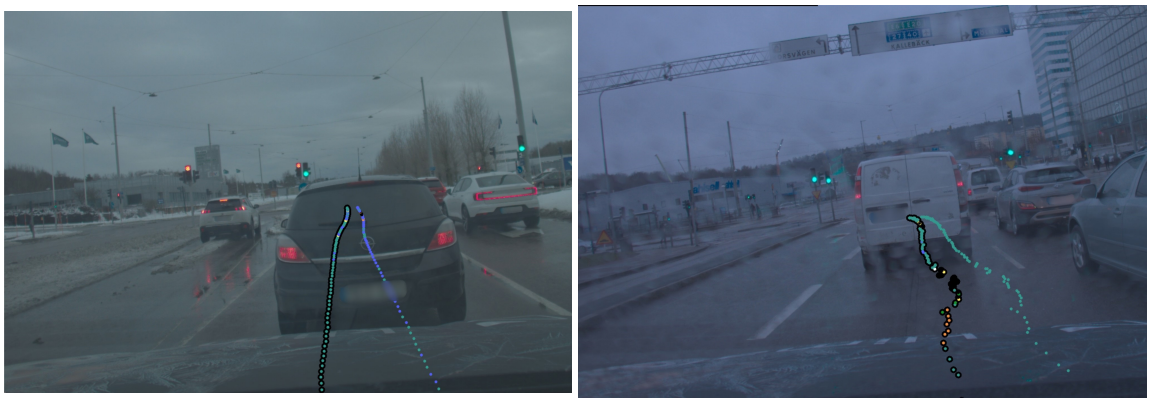


Figure 3.4: The figures show two images where vehicles are in front of the camera, blocking the view of the road. As such, annotations are mistakenly located on the vehicles ahead instead of on the road. Image by author.

The motion estimations are also based on GPS data, which works well most of the time, but when the vehicle is in a tunnel, the laser annotations are way off the actual driving path, see Figure 3.5.

Figure 3.5: The figure showcases a scenario where the recording vehicle is driving under a tunnel. The estimations used to create laser annotations are based on GPS data, and the tunnel severely limits the ability to receive such data. Consequently, this results in laser annotations incorrectly placed outside the recording vehicle's driving path. Image by author.

3.1.3.3 Diversity

Naturally, some weather-related conditions occur more often than others. This affects the diversity of road surface conditions in the data and, therefore, the class diversity in the provided dataset. The most common road surface conditions are dry and snow. However, all conditions except slush appear relatively often, which is quite reasonable since slush is a relatively short-lived road condition that occurs somewhere in between a snowy and a wet road.

To create a more balanced dataset, one can discard data that does not contain the underrepresented road surface conditions. However, it is also desirable to have a large dataset that contains as much information as possible, and therefore, discarding collected data is wasteful. As such, the low number of slush conditions creates a trade-off between wanting a balanced dataset and keeping it large.

3.1.4 Study of current pipeline

As mentioned, the company had already developed a pipeline to perform the task of semantic segmentation of road surface conditions. This pipeline implements functions for data pre-processing, data parsing, training, and evaluation of a machine learning model, which will be introduced later in this section. The project opted to study the functions of the provided pipeline and evaluate which should be retained and where adjustments would be beneficial.

3.1.4.1 Data pre-processing

The company has provided a substantial amount of data to build datasets for this project. As previously stated, the data is organised in sessions, each containing between a few hundred and a few thousand consecutive frames. To address the problem of traffic blocking the annotations in the road discussed in section 3.1.3.2, the company has, for some sessions, applied another neural network to filter out sparse labels that are located on top of vehicles, as in Figure 3.4. The project will adhere to these sessions to avoid any unnecessary problems that will undoubtedly arise from this flaw.

Additionally, the company has implemented a function to remove images with only a few or no annotated road surface conditions. Due to how the motion estimation works, this can occur when the recording vehicle is standing still. Since the vehicle is not moving, it will try to annotate labels underneath it that are not within the image frame. However, this filtering function is applied before any of the other data parsings, and therefore, the final input to the machine learning model may still contain few labels.

The company has also developed a function to split the provided sessions into a training and a validation dataset in a balanced manner. The split is done evenly, so the training and the validation dataset should contain the same number of sessions. This function is based upon simulated annealing [35] and considers night and year, such that the training and validation dataset should contain the same fraction of sessions recorded in each year and sessions recorded during the night. The reasoning behind night is relatively self-explanatory. One dataset should not exclusively contain night frames and the other day frames, as that would severely limit the model's learning capability. The year is a little more subtle, but it is essential to understand that data recorded during the same year likely has more characteristics in common than data recorded in different years. Therefore, the recording year is also a parameter used while splitting sessions to ensure a more balanced dataset that contains as many different characteristics as possible. The split function also ensures that each session is exclusive to either the training or validation dataset. This split prevents frames from the same sessions, which likely share similar characteristics, from being in both datasets, ensuring that the validation dataset contains new data.

After the sessions are attributed to either the training or validation dataset, they are further pre-processed by considering time. Two consecutive frames probably share a lot of the same information and, therefore, contain a lot of redundant data. Thus, the company has implemented a function with a criterion that the frames in the dataset should be some time apart to avoid frames that are essentially the same.

Furthermore, the provided pipeline has an implemented balancing function to ensure a more balanced dataset in terms of road surface condition diversity. Regrettably, the balancing is achieved by discarding frames, which results in some information loss. However, to save as many frames as possible, the function is biased towards protecting frames containing the least common road surface condition while discarding frames with the most common road surface condition. The function is run until a balance according to a balance ratio has been achieved.

3.1.4.2 Parsing of data

The provided pipeline uses two different ways of parsing data. The first method involves cropping the original image to remove unnecessary background elements such as the sky, the recording vehicle and the roadside. Apart from a function that randomly crops away up to 15% of the image's left side, this cropping was done in a static manner. After cropping, the original image is rescaled to a lower fixed resolution, significantly speeding up the training process. The final image resolution after parsing using this method is 816x224.

The other approach to data parsing uses the known camera parameters to perform an image transform that produces a bird's-eye view of the road. This transformation targets a 50x8 meter section of the road in front of the vehicle. This method almost guarantees that the input image for the model consists exclusively of the road. However, it also introduces noise into the data due to estimations made in the transformation process. The final image resolution using this method is 288x816.

Both approaches are valid and reasonable, and it is therefore desired to study their performances to determine which is most suitable for the task at hand. The data generated from these methods will henceforth be referred to as "non-bev data" and "bev data" for the first and second methods, respectively, where "bev" refers to a bird's-eye view.

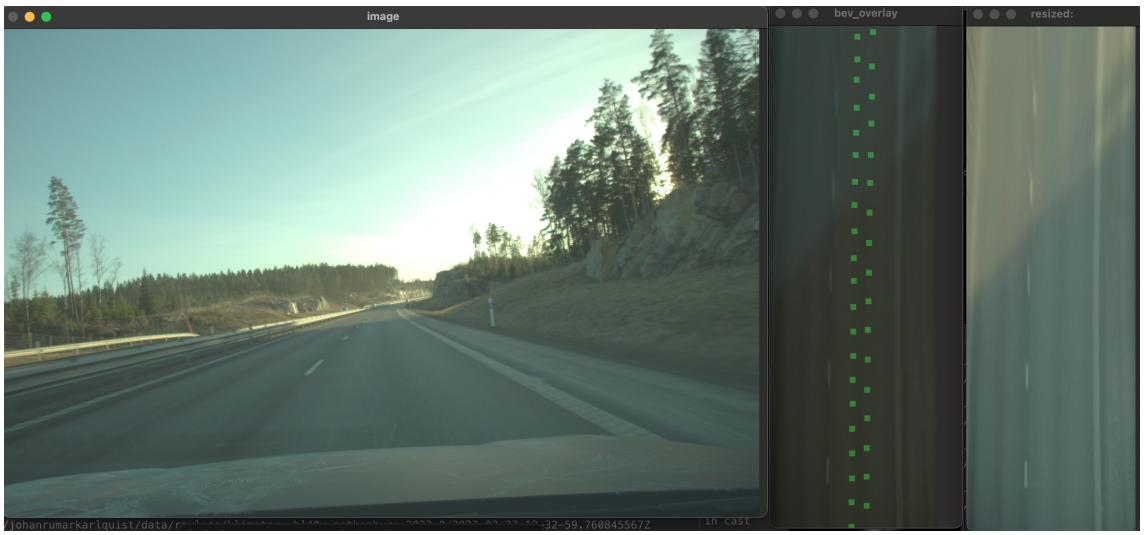


Figure 3.6: The figure showcases three different settings. The leftmost image is the original image taken by the front-facing camera in the vehicle and contains no annotations. The rightmost image is the original image after the transformation, resulting in a bird's eye view of the most relevant part of the road. The middle image is an augmented version of the transformed image containing the annotations. In this case, the road is correctly annotated as dry. Image by author.

3.1.4.3 Label enhancement

The main problem with sparsely annotated datasets is the lack of information available for supervision. The annotated pixels represent a significant minority, and there

are typically merely 20-50 annotated pixels out of more than 150000 total pixels in a frame. This causes a lot of problems that are more thoroughly discussed in section 5. To counteract this problem, the company has implemented some straightforward methods to extend supervision to the unlabeled pixels.

For bev images, the company employs Gaussian kernels to propagate annotation values to unlabelled pixels around each labelled one. These kernels retain the probabilistic distribution of the labelled pixel. The kernels have a radius of 40 pixels, meaning that any unlabelled pixel within 40 pixels of a labelled pixel inherits the same probabilistic distribution as the labelled pixel. However, the unlabelled pixel's contribution to the loss depends on the distance to the labelled pixel in a Gaussian manner.

For non-bev images, the company has opted to create squares centred on the labelled pixel. These squares have a side length of 10 pixels, and all unlabelled pixels within these squares are assigned the same probabilistic distribution as the labelled pixel.

Both approaches aim to extend supervision to a broader set of pixels, which is achieved through the assumption that the road's surface condition does not change much close to a labelled pixel. If the assumption holds true, it significantly increases the amount of information available for supervision. However, if it does not, it could lead to the model receiving faulty supervision, potentially deteriorating the model's ability to accurately connect features to road surface conditions.

3.1.4.4 Data augmentation

The company has also implemented some methods of data augmentation into the provided pipeline. These are vertical and horizontal flips, translation (and mirror) in the x-direction, quality, and brightness. These augmentations are designed explicitly for bev data, with which the company had the most success. The augmentations are intended to counteract the flaws found in the dataset and, as such, mitigate the effect these flaws will have on the pipeline's performance.

The position of the sparse labels in the dataset is almost the same for all images, that is, in two straight lines directly in front of the vehicle. The translation in the x-direction is intended to diversify the position at which these sparse labels appear, lest the model starts learning that only the prediction result of the pixels in a similar position as the sparse labels matter to reduce the loss while training. This augmentation also entails a mirror function, where if the image is translated by 20% to the left, 20 % of the image will be mirrored to the right side.

Another observation in the dataset is that the sparse labels further away are naturally of lower resolution, especially for bev images. Consequently, the input image will have a lower resolution on the top part of the image compared to the bottom part. The vertical flip counteracts this phenomenon by making the lower resolution appear on the bottom of the input image instead, while the horizontal flip introduces variability in the orientation of the road.

The augmentation for quality is intended for the same purpose and has two choices with equal chance: lower resolution and sharpening. A lower resolution is simulated

by resizing the image, first down, then up again. The sharpening is performed by using an image enhancer function implemented by the Python Imaging Library [36]. The augmentation of brightness is implemented to counteract the fact that images have different lighting due to time, date and weather. In the optimal case, the pipeline's performance should be independent of the lighting in the image. The brightness is adjusted by multiplying the RGB values of the image with a brightness value.

These augmentations are applied by a fixed chance, originally 50% for each flip, 60% for quality, 30% for brightness and 80% for translation. The magnitude of the augmentations, excluding the flips, are also decided in a randomised manner where the translation is between 40% of the image size to either direction and the quality and brightness augmentations are based on a random value between 0 and 1.

The company did not use any augmentations for non-bev data except for the small variable translation introduced in the cropping function while parsing the data.

3.1.4.5 Machine learning model and training

The company's original pipeline utilises the machine learning model ENet. ENet is trained with the Adam optimiser with a learning rate of 0.01, momentum of 0.9 and a weight decay of 1e-4 over 40 epochs.

For bev data, the loss is computed by a cross-entropy loss function that takes the soft labels into account. The total loss from cross entropy is multiplied by a loss mask pixel value. Labelled pixels will have a pixel value of 1; pixels within a Gaussian kernel have a value between 0 and 1, depending on the distance to the nearest labelled pixel. The remaining pixels will have a value of 0. This pixel-wise loss is then summed and divided by the total number of pixels.

For non-bev data, the loss is also computed by a cross-entropy loss function, but without any weights attached to the pixels within each label square. That is, each pixel within a label square will have a pixel value of 1 in the loss mask, equally contributing to the loss, while the other pixels will have a value of 0 and will not have any contribution to the total loss.

3.1.4.6 Evaluation

The provided pipeline saved the progress made while training using tensorboards, a part of the tensorflow library [37]. Tensorboard is a helpful visualisation tool with which one can save important parameters in suitable training steps that are visualised as graphs over the steps. It is also possible to save step-specific images through the tensorboard.

The evaluation metrics used are accuracy, confusion matrices and manual evaluation on visualised images. These values of the metrics and the images are saved in customisable steps. On top of this, the tensorboard is also utilised to track the training and validation loss. Several different accuracies are of interest. The most prominent are the individual accuracies of the condition classes. However, the company has also decided to use a "slip" accuracy. The slip condition is divided into

two classes and can be seen as a binary classification. The dry and wet conditions are considered non-slip conditions, while slush, snow and grey ice are considered slip conditions. The model may not be able to predict each condition class individually, but it should, at the very least, be able to predict whether a road poses a risk of slipping.

The accuracies are calculated batch-wise, and the mean batch accuracy is used as the final result. The accuracies are also based on hard labels. These hard labels are generated by taking the condition with the highest probability in the soft labels and the predictions of the machine learning model. The same procedure of hard labels applies to the created confusion matrices. However, unlike accuracy, these are not generated on a batch-wise basis and then averaged. Instead, all labels and their corresponding predictions over the entire dataset are saved and used to create the confusion matrix.

The evaluation is done on both non-augmented training data and data from the validation set. However, it does not utilise the entire dataset during each evaluation cycle. Instead, it calculates the evaluation metrics based on a maximum number of batches. The maximum number of batches was originally 15 for evaluation on non-augmented training data and 25 for evaluation on validation data.

3.2 Development of the machine learning model

This section will introduce the functions, methods, and models that are implemented by the project and used to build upon the pipeline further in an effort to optimise its performance.

3.2.1 Data pre-processing

The project has decided to retain the provided pipeline's approach to data pre-processing, except for an adjustment in the function that splits sessions into training and validation data. This new adjustment ensured that data recorded on the same day by the same company does not appear in both the training and validation data as they likely share similar characteristics. This adjustment aimed to further differentiate the training and validation data, such that the validation data represented new data that the model had not yet encountered. However, this implementation was only introduced for the second dataset as most of the training runs for the first dataset were already finished at the time of its implementation.

3.2.2 Class weights

Class weights were introduced to mitigate the class imbalance found in the datasets. This imbalance is most noticeable in the first dataset, where slush is by far the least common, making it challenging for the model to learn the features of that condition. While it would have been possible to balance the data until each road surface condition is equally represented, this would significantly reduce the number

of frames available for training and evaluation, causing considerable harm to the model's ability to generalise and would likely have led to overfitting.

The class weights were inversely proportional to the occurrences of each road surface condition. For instance, the first dataset was balanced with a balance ratio of two, resulting in roughly equal amounts of dry, wet, snow and grey ice conditions but merely half as many slush conditions. Consequently, the class weight for slush were roughly double that of the class weights for the other conditions. These class weights were used while calculating loss, where the network was punished twice as much for incorrectly classifying slush conditions.

The second dataset was based on a significantly larger number of sessions than the first. Consequently, a lot more data was available, making it less susceptible to a harsher balance criterion without the risk of ending up with too few frames. This dataset aimed to have roughly the same number of occurrences for each class condition with a balance ratio of 1.05. Even though the balance ratio was less harsh than for the first dataset, it still led to a non-negligible discrepancy between occurrences of class conditions. Consequently, class weights were still necessary, even though their effect was less noticeable than for the other dataset.

3.2.3 Data parsing

The project utilised the same data parsing as the provided pipeline, with a few minor adjustments. That is, the project will take two cases into account, bev and non-bev images. Each of these cases had advantages and disadvantages, which are more thoroughly discussed in the discussion section 5. It was of interest to gauge the performance between both cases and, as such, determine which of them was more suitable for the task at hand.

The project chose to discard the function that randomly cropped away parts of the left side of the non-bev images, initially implemented in the provided data. Additionally, minor adjustments were made in the cropping values for non-bev data. Specifically, in the x-direction, 15% of the image was removed from both the left and right sides to discard the roadside. In the y-direction, the 200-400 bottommost pixels were cropped out to remove the vehicle's engine head. Finally, 30-50% of the original image height was retained from the bottom up, ensuring that the focus of the image remained on the road ahead and not on unnecessary background. The variation in the y-direction crop accounts for different camera angles and recording vehicles used by different collaboration companies.

While it may seem that the chosen cropping method removes a significant portion of the road by only saving such a small fraction of the original image height, the cropped-out road sections would be further away, where the labels are less reliable and more prone to the flaws of the dataset than those closer to the vehicle. Additionally, since the cropping is static, it leaves a non-negligible amount of background for some images, but that should still not cause any trouble. The sparse labels were usually located in the centre of the road, far away from any background, and the model did not receive supervision for pixels that were far away from the labels. Additionally, it is possible that the background provided some contextual information

that the model may have been able to utilise in its predictions.

3.2.4 Data augmentation

The project replaced the brightness and quality augmentations with new versions of augmentations, which share the same purpose as the previous ones. The brightness augmentation was extended to include other colour aspects, such as contrast, saturation, and hue. The quality augmentation was replaced by introducing a random Gaussian blur to the image instead. This augmentation added a Gaussian noise to the image, which simulates a lower resolution, which seemed more reasonable than the previous method of resizing the image for a similar effect. The new augmentations introduced in this project were implemented using pre-built functions developed by the Python Imaging Library [36]. Additionally, the aggressiveness of all augmentations was reduced slightly, as there were indicators that they augmented the data too much and, consequently, did not resemble real-life possible scenarios, resulting in lower performance.

Initially, the project also applied data augmentation on non-bev data, which was not done in the provided pipeline. However, this led to some undesired behaviours, presented in section 4. Consequently, it was decided that no augmentations were to be applied to non-bev data for the first dataset, while on the second dataset, only colour and Gaussian blur augmentation were applied.

3.2.5 Restructuring of code and quality of life improvements

The project also had to majorly restructure the code for the provided pipeline to accommodate the newly implemented functions and machine learning models. The original pipeline was limited to the ENet machine learning model, and the restructuring aimed to make the system more flexible and convenient for integrating different models.

While the primary inputs of all implemented models were similar, such as the chosen optimiser and loss masks that represent the labels, they often differed in design. For instance, the Adam and SGD optimisers both required an input of learning rate but were otherwise initialised differently. Consequently, the project implemented abstract classes for each training model. This allowed every model used in the project to have common function calls, such as initialising the machine learning model, setting up model-specific optimiser, parsing the data, calculating the total loss, and many more.

These function calls typically required the same type of parameters but used different procedures to satisfy each model's individual demands. For instance, different models had different ways of calculating loss, which could have been solved using if-else statements. However, that approach would clutter the main training file and quickly become unmanageable as new models and methods are implemented. By implementing an abstract class named "trainingmodel" in this project, common functions can be streamlined. For instance, calling trainingmodel.optimiser(A,B) correctly sets up the optimiser for the specific model that "trainingmodel" is set to.

The value of ”trainingmodel” is determined by command-line arguments provided when starting the training.

Another crucial implementation was the integration of PyTorch's dataloader class [38]. It was noticed that the loading of training data and evaluation was inefficient and caused a significant reduction in training speed. The dataloader class, commonly used because of its intuitive and simple yet effective functionalities, was adopted to address this issue. The class introduces a lot of flexibility and allows for easy customisation of parameters, such as augmentation and batch size, in the training process. However, the most desired part of the dataloader class is the built-in multiprocessing functionality. The integration of dataloaders allowed for sub-processes to prepare upcoming batches while the current batch was being processed, effectively eliminating the loading time of datasets. Unfortunately, it did come at the price of initialising each sub-process before loading, which was not free. The sub-processes did not inherit information from the main process and had to gather that themselves. The initialisation included independently importing Python packages and functions from other files. The initialisation of each sub-process took up to 10 seconds. However, since the multiprocessing eliminated loading time, the cost was easily covered when using a sufficiently large dataset.

For instance, training with the original ENet model on bev data took approximately 0.5-0.6 seconds for each batch with the original implementation of loading data. With the Dataloader class, this time was reduced to roughly 0.3 seconds, almost halving the training time. Now, this also came with the additional cost of initialising each subprocess. However, when using eight subprocesses, the initialisation took approximately 40 seconds, and training on the smaller first dataset with 1000 batches for each epoch effectively reduced the time for one training epoch from 500-600 seconds to 340 seconds.

The dataloader class also included other convenient features, such as shuffling, which randomised the order of batches while training, distributing the propagation of loss in a desired way. For example, suppose some batches contain primarily dry conditions. In that case, it is not desirable for them to always come in the same order since the model will adjust its parameters similarly in each epoch. This behaviour leads to the model learning patterns related to the order rather than the actual features found in the data.

3.2.6 Training

The training is done in epochs, where one epoch signifies that each training frame has been processed. The amount of epochs varied depending on discoveries made while training. The original base amount was 60 epochs, but it was noticed that some models saturate way before getting to the 60th epoch, especially for non-bev data. Therefore, the epochs for these training sessions were reduced to 40. While training, the training dataset was divided into smaller batches. The size of these batches depended on the model's size and was limited by the GPU's memory. For Enet, SFNet and PIDNet, the batch size used was 16, while the Deeplabv3plus model required a reduced batch size of 8.

3.2.6.1 Machine learning models

This project used all models that were introduced in the Theory section 2. In particular, SFNet [11] and Deeplabv3plus [13] utilised the ResNet-18 [12] as a backbone network. The relatively shallow ResNet was chosen to reduce the number of parameters and, as such, the computation time. Furthermore, it was uncertain whether more model complexity would be helpful, considering the flaws and characteristics of the provided data. The ResNet backbone was also pre-trained on the ImageNet dataset [39]. Note that the ImageNet dataset does not necessarily contain data similar to the data used in this project. However, according to Karpathy [31], using a pre-trained network rarely hurts performance and is generally a good idea.

Furthermore, this project implemented the smallest PIDNet model, which was pre-trained on ImageNet [39] as well. PIDNet comes with several different losses introduced more thoroughly in the Theory section 2. The calculation for l_1 and l_3 required boundary detection in the ground truth, which works well for densely annotated datasets, but for sparsely annotated datasets, like the one provided for the project, there are no boundaries in the ground truth. Consequently, these losses were deemed unimportant in this context and were not used, lessening the impact of the D branch in the network. The report used the same prefactors for the remaining losses l_0 and l_2, that is $\lambda_0 = 0.4$ and $\lambda_2 = 1$, even though the other losses l_1 and l_3 were discarded.

3.2.6.2 Optimisers

The project initially adhered to the optimisers used by the original developers of each model. Consequently, all models except for ENet [9] used the SGD optimiser, while ENet used Adam. According to Karpathy [31], SGD has more potential than Adam, especially when hyperparameters such as learning rate are properly adjusted. Unfortunately, this project did not do such an extensive study of the impact between different values of the hyperparameters. However, from a study of a smaller scale, the learning rate for SGD was empirically determined to be 0.01 for SFNet and PIDNet, while Deeplabv3plus used a learning rate of 0.001. A polynomial learning rate scheduler was employed for the SGD optimiser according to:

$$\eta_n = \eta_0 * (1 - \frac{n}{m})^{0.9}, \tag{3.1}$$

which is also the scheduler used by the original developers of the used machine learning models.

However, Adam is a bit less sensitive to hyperparameters such as the learning rate, making it highly attractive since computing power and time are significant limiting factors due to the large amount of hyperparameters to be studied. Therefore, the fewer tweaks needed for the learning rate, the better. Consequently, the performance of the Adam optimiser was implemented even on models whose developers opted to use the SGD optimiser. However, Adam was the only optimiser used for the ENet model since the developers of ENet [9] used Adam. As such, it was assumed that the model was well adapted to this optimisation algorithm, and since this optimiser comes with less hassle, there was no real reason to test any other optimiser for ENet.

For those models with multiple parameter groups, such as PIDNet, the second parameter group had a learning rate ten times that of the base learning rate discussed earlier, following the procedure implemented by the developers of PIDNet [10]. The momentum and weight decay for all models, independent of the optimiser, were set to 0.9 and 0.001, respectively. When using the Adam optimiser, the learning rate was set to 0.001, and there was no learning rate scheduler since the Adam optimiser inherently adapts the learning rate depending on the first and second moments of the gradients.

3.2.6.3 Label enhancement

The data provided for the project was sparsely annotated, severely limiting the amount of information that could be used for supervision. In comparison, most machine learning models used for similar tasks in semantic segmentation are developed for densely annotated datasets, such as Cityscapes [5]. In an effort to artificially increase the amount of supervision available, the report has implemented multiple ways to extract further information from the sparse labels. Two of those ways are the ones originally used in the provided pipeline. That is, Gaussian kernels and larger square labels, which assume that all pixels within a square around the labelled pixels have the same condition distribution as the labelled pixel.

The project also implemented the AGMM [14] and the Tree Energy Loss (TEL) [7] methods. These methods are specifically designed to provide supervision even from the unlabeled pixels.

AGMM uses feature extraction from the annotated pixels to compute the mean features of each labelled class. The unlabelled pixels are then assigned soft predictions of condition classes depending on the distance to the labelled classes' mean features in feature space. These predictions are used to provide supervision from unlabelled pixels by added loss functions. A more thorough description of AGMM can be found in section 2.6.1.

Tree Energy Loss is a method that is used to create pseudo labels for the unlabeled pixels in three steps: tree affinity generation, cascaded filtering, and soft label assignment. The intricate functions are more thoroughly explained in section 2.6.2. These pseudo labels can then be utilised to provide supervision even on unlabeled pixels while training. Both methods have proven to provide increased semantic segmentation performance on datasets such as Cityscapes [5] and PASCAL VOC [40]. Sparse annotations are available for PASCAL, but for Cityscapes, which is a densely annotated dataset, the sparse labels are randomly chosen from the ground truth [14]. Additionally, the project will adhere to the values set by the original authors for any hyperparameters that regard the creation and computation of loss in these methods unless stated otherwise.

The sparse labels in the data were soft, meaning each label came with a probability distribution over road surface conditions in that pixel. However, it was uncertain whether the extra information provided by soft labels would be beneficial or if hard labels would be better. For hard labels, each labelled pixel is considered to be 100% certain upon a single road surface condition and are the labels commonly used in

other datasets, such as Cityscapes [5], for semantic segmentation, making them more lucrative as they conform to other research within the subject.

This project made use of both approaches, and studies were performed where the soft labels were converted into hard ones, allowing the project to determine which labelling standard was the best. The conversion to hard labels was done in two different ways. The first one assumes that the condition with the largest probability in the distribution is the actual surface condition of the road and is used as the hard label. This is a reasonable assumption as long as the probability of the most likely condition is high, say $0.7 <$. However, the assumption is less reasonable when the probability between conditions is closer, say snow with 0.51 and grey ice with 0.49. In such a case, the model may associate features prominent in grey ice conditions with snow, which would harm the model's ability to learn. Therefore, the second approach was to ensure that the assumption holds reasonably by setting a threshold that the condition with the largest probability has to exceed. This threshold was set to 0.7 in the project. If the threshold was not exceeded, the label was seen as too controversial and was removed. Both of these approaches were tested to prove which of them is more effective.

There are also methods that essentially require hard labels. One such is the AGMM [14]. The loss functions introduced by this method assume hard labels, and modifying these functions to allow soft-labelled pixels would have been difficult and, therefore, not pursued. Note that while the added loss functions use soft predictions from unlabelled pixels, the unlabelled pixels had to be based on hard-labelled pixels in the AGMM due to the function that calculates the mean feature vectors of each class condition, see section 2.6.1.

However, the base cross-entropy loss function did allow soft labels, so it would have been possible to utilise both soft and hard labels in the AGMM. In such a case, the soft labels would be used in the cross-entropy loss function, which was still expected to contribute to a small majority of the total loss, while the added loss functions from AGMM would have used hard labels. The same principle applies to the TEL method [7], where the cross-entropy loss allows soft labels, but the added tree loss had to be computed based on hard-labelled pixels. However, the project decided not to follow through with soft labels for any of these methods due to limited computing time. Instead, they were implemented with hard point labels both for calculating the loss for the base cross-entropy loss and the added loss function introduced in each method.

In total, there were seven different labelling methods. Gaussian kernels with soft labels, larger square labels with both soft and hard labels, hard and soft point labels, AGMM with hard labels and TEL with hard labels. However, the project lacked enough time or computing power to apply each technique to each model. Furthermore, in the case where one labelling technique had a significant lack of performance on one of the machine learning models, it was assumed to have a lack of performance on the other machine learning models as well. In such a case, that labelling technique was not studied further, at least not in the same context. Additionally, partly following the label enhancement approaches made by the provided pipeline, the Gaussian labels were only applied to bev data, while the larger square

labels were limited to non-bev data.

As such, ENet was trained using the labelling techniques implemented in the provided pipeline: Gaussian labels for bev data and larger soft square labels for non-bev data. Additionally, AGMM was applied to bev and non-bev data on ENet to gauge the method's performance. Consequently, since AGMM uses hard point labels for the base cross-entropy loss, the ENet model was also trained with hard point labels for bev data to evaluate the efficiency of AGMM.

SFNet and PIDNet used the same labelling techniques. For bev data, this included soft and hard point labels as well as Gaussian labels. For non-bev data, the labelling schemes used was hard point labels, large square hard and soft labels. Additionally, the Tree Energy Loss method was applied exclusively to the SFNet model.

The deeplabv3plus model was solely used for the purpose of trying out the performance of AGMM as this was the model adopted by the authors of AGMM [14]. As such, the performance of the model was tested on hard point labels with AGMM applied and not applied. Additionally, for curiosity, the model was also trained on soft large labels for non-bev data, which enabled further comparison to the other models as well as comparison between AGMM and soft large labels.

Overall, the methodology provided sufficient results to draw conclusions about the efficiency of the label enhancement technique and the efficiency of the applied machine learning models.

3.2.6.4 Loss functions

Several different loss functions were used depending on the type of labels and the methods used for label densification. A common factor independent of the model and labels is that the total loss was primarily based on a cross-entropy function similar to the one introduced in section 2.5. Minor adjustments were made to this loss function to accommodate the different labelling techniques, such as the Gaussian labels introducing a weighted contribution to loss depending on the distance to a labelled pixel. Furthermore, the sparsely annotated semantic segmentation methods of AGMM and TEL introduced additional loss functions to provide supervision from unlabeled pixels. These additional loss functions are presented more thoroughly in section 2.6. However, these additional losses typically contributed to a minority of the total loss compared to the base cross-entropy loss function.

3.2.7 Evaluation

The project kept a large part of the evaluation methods implemented in the provided pipeline. In particular, the tensorboard implementation was found to be a convenient tool. However, other evaluation methods were also implemented. In particular, the project implemented a mean intersection over union metric (mIoU), a popular approach for evaluation in semantic segmentation tasks [11, 9, 14, 10, 13]. The mIoU value is calculated with the help of the created confusion matrices according to equation (2.16) in the theory section 2.5.2.

The project also opted to perform evaluation steps between each epoch and used

the whole validation dataset for evaluation. In comparison, the provided pipeline performed evaluation steps every X batch and only used a subset of the validation data in each step. The project also chose to discard evaluations done on non-augmented training data, which was a part of the provided pipeline.

However, since only the labelled pixels can be used to compute the accuracy and mIoU, these metrics were only able to evaluate the model's ability to predict the road conditions at the sparse labels. They are further estimated by converting the soft labels to hard ones, making them even more unreliable. The flaws in these metrics are discussed more thoroughly in section 5.10. Further evaluation is required to assess the model's performance for semantic segmentation on the entire image. Consequently, much like the provided pipeline, images of the input and the model's predictions on some validation batches were saved in each evaluation step. However, these images are sometimes also unsatisfactory as some of them will visualise uninteresting scenarios.

Therefore, to further analyse the models' overall ability for semantic segmentation, the trained models were also applied on complete recording sessions. These sessions consisted of a significant number of consecutive frames, simulating the result should the model have been deployed in a vehicle. This provides a clearer view of the model's performance and reduces biases made while studying the still images of the validation dataset. In conclusion, accuracy and mIoU were used as indicators of the model's performance, and the few best models in terms of mIoU were saved to apply them to recording sessions with consecutive frames, which provided unbiased results of the model's predictions and allowed for manual evaluation.

4

Results

This chapter will present the result of this thesis. The results are based on the evaluation metrics discussed in the methodology section 3.

There will also be a significant amount of figures in this chapter, some of which will include information about the input image, the model prediction and the sparse laser annotations. However, those images can be difficult to interpret without some guidance, which will be provided here. See Figure 4.1 for a reference figure. Each figure will include six sets of images taken from the validation dataset, and each image set will consist of three different images: input data, model output and model output with sparse labels added. The figures will have a matrix-like appearance where the top row corresponds to the input data for each image set, the middle row is the model output, and the bottom row is the model output with the sparse labels added. The class conditions are colour-coded, where the dry, wet, slush, snow and grey ice conditions are represented as green, blue, orange, yellow and red, respectively. The sparse labels take the form of small rectangles, which may sometimes be difficult to spot depending on image, condition classes and labels. Note that even though the labels are visualised as rectangles, they may represent a point label where only a single pixel is labelled. Additionally, there is a loss in the top left corner of the figures that will appear in this section. This loss is insignificant to the result, and its value largely depends on the labelling technique used, where one labelling technique may come with a significantly lower loss even though the actual performance, measured in mIoU, is lower compared to another labelling technique.

Additionally, the image sets presented in this section are chosen because they effectively present the results in multiple different and interesting scenarios. However, they don't necessarily reflect a model's overall performance. Some models may perform well on these example image sets but not on those omitted in the report. Consequently, these images are intended to present possible results that can be used as indicators, but more context is needed to draw any well-established conclusions.

Furthermore, the methodology also mentioned that trained models are visualised on sections with consecutive frames. The resulting videos are presented in the appendix section A, where the trained models are applied on an example session.

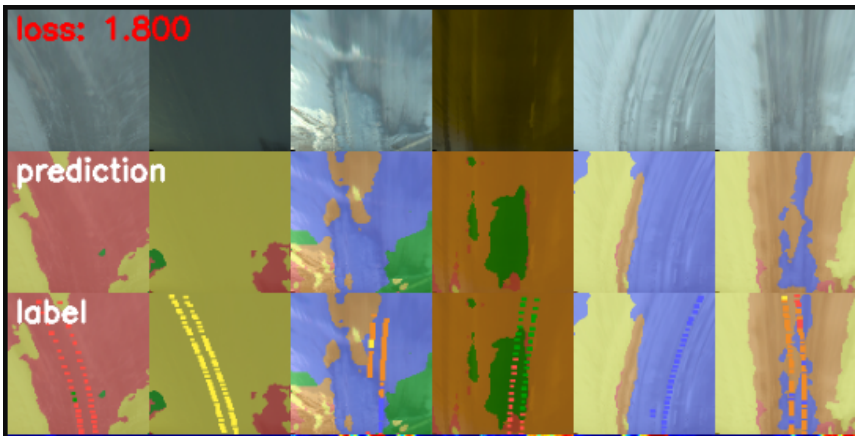


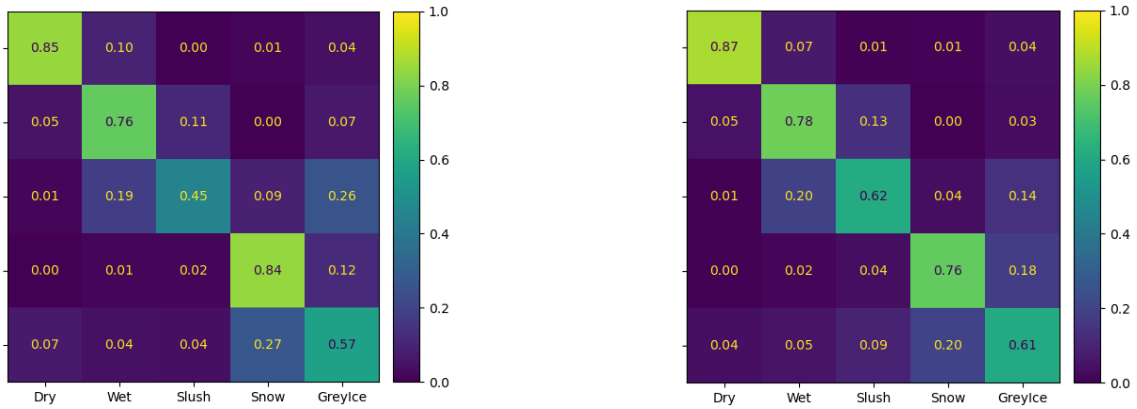
Figure 4.1: The figure shows an example of results from SFNet trained with bev input data from the first dataset. From left to right, there are six different image sets, each consisting of three images. From top to bottom, these images correspond to input data, model prediction and model prediction with sparse labels added. The colours correspond to different class conditions where dry, wet, slush, snow and grey ice are coloured green, blue, orange, yellow and red, respectively. The sparse labels can be seen in the images in the bottom row, where each label is represented as a rectangle coloured according to its corresponding class. Note that these labels are originally soft but have been converted into hard ones by taking the condition with the highest probability. The rectangle form of the labels is only adapted to increase visibility. The actual labels are, in this case, confined to single pixels.

4.1 Class weights

Class weights were introduced to tackle the problem of class imbalance in the datasets. The first dataset was the most imbalanced one, where the total amount of labelled slush conditions was half that of the other conditions. As such, the slush condition had a class weight value of 2 and the rest a value of 1. The class weights were applied in the loss functions, punishing the network twice as much for incorrectly classifying slush conditions, creating a bias towards the model learning and predicting slush conditions. The results are shown in Figure 4.2, where the accuracy for slush is significantly increased with implemented class weights, while the accuracies of the other conditions are not changed dramatically.

4.2 Data augmentation

Given the differences in data pre-processing methods, it is expected that data augmentations will influence the performance of bev and non-bev data differently. While the implemented data augmentations were effective for bev data, they did not perform well for non-bev data. For instance, it was noted that while using all augmentations for the first dataset, the model developed some undesired predictive behaviour, seen in Figure 4.3, where the model consistently predicts a single condition over the whole image.



(a) **(b)**

Figure 4.2: The resulting confusion matrices of the ENet model trained on non-bev data from the first dataset with a) no class weights and b) implemented class weights. The difference in slush accuracy is quite significant. Without class weights, the accuracy is 45%, and with class weights, it is 62%. The only condition that is harmed with the application of class weights is the snow condition, while the other conditions are improved.

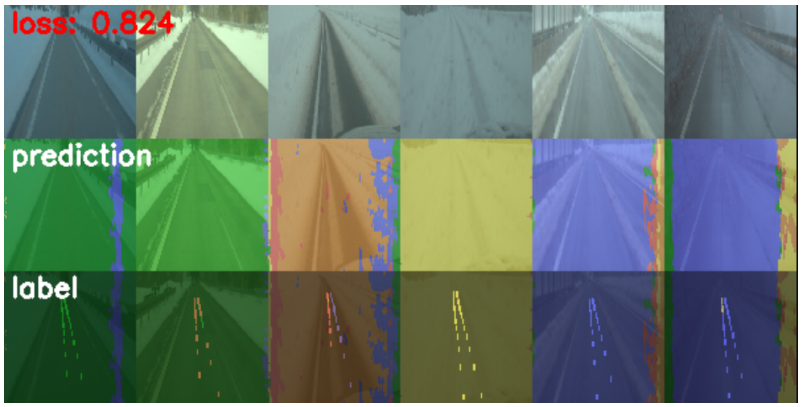


Figure 4.3: The figure shows input images (top row), model prediction (middle row) and model prediction with labels (bottom row). These images come from a run where the ENet model is trained on non-bev images with all augmentations applied. The figure shows that the model predicts a single condition over the entire image. As the goal is to create a semantic segmentation of the input image, this is undesired.

To identify the cause of this undesired behaviour, the model was trained with all data augmentations turned off except for one. This process was repeated consecutively for each augmentation method, revealing that the translation was the lead cause of the undesired behaviour. Additionally, it was observed that the other augmentations were not beneficial for the performance either. This led to the suspicion that the training frames for the first dataset were likely too similar to the validation frames. Consequently, augmenting the training frames introduced discrepancies between the training and validation sets, resulting in poorer performance. The second dataset

was mainly introduced to test this theory. This dataset is more extensive and was created through methods that further separate frames added to the training dataset and frames added to the validation dataset that is discussed more thoroughly in section 3.1.2.

However, the same behaviour from the translation augmentation appeared even in the second dataset, as seen in Figure 4.4. Furthermore, it was decided that the flip augmentations would also be removed as they do not provide a reasonable situation for non-bev images as they do for bev-images. That leaves Gaussian blur and colour augmentations. It was noted that these did not significantly impact the performance on the second dataset, see Figure 4.5. However, they were kept since they are expected to increase the model's ability to generalise by artificially creating new reasonable training frames.

In conclusion, as a result of these discoveries, it was decided that there would not be any augmentations applied for non-bev data on the first dataset, and only the Gaussian blur and colour augmentations would remain for non-bev data on the second dataset.

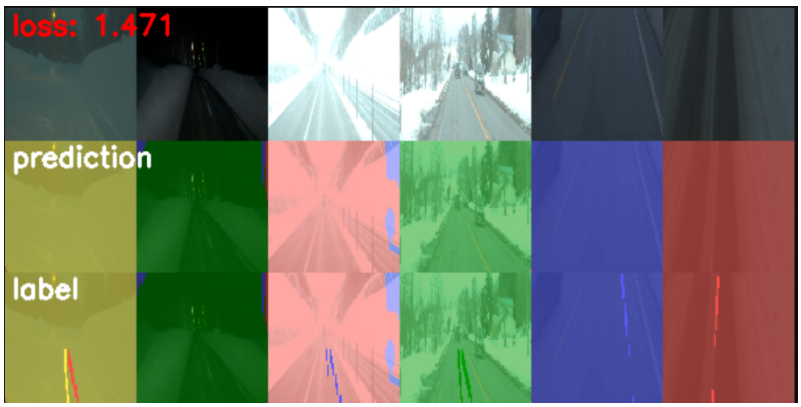


Figure 4.4: The figure shows the results from a training run where the SFNet model was trained on non-bev images from the second dataset using all augmentations. It is clear that the undesired behaviour where the model predicts a single condition over the entire image is still present, even though the second dataset introduced larger discrepancies between the training and validation data.

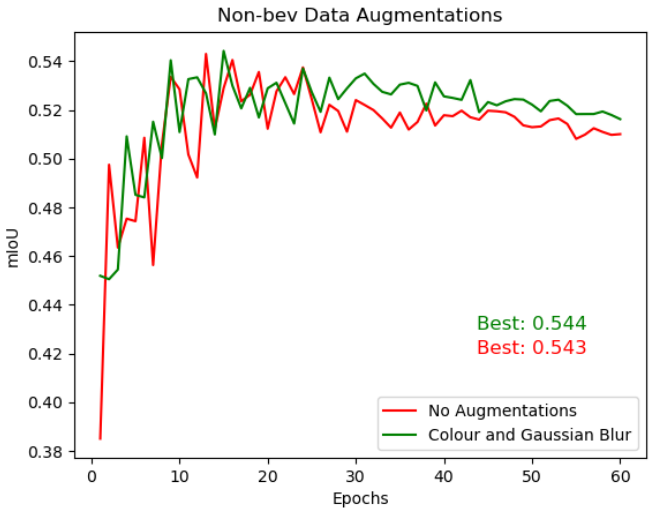


Figure 4.5: The performance graphs between training runs with the SFNet model while applying different augmentations. The performance between the two is similar, and one can conclude that the two augmentations do not affect the performance by much. However, it is probable that the augmentations improve the model's ability to generalise since they help create a more diverse training set.

4.3 Optimisers

As discussed in the methodology section, the optimisers used were the SGD [29] and Adam [30] optimisation algorithms. The project opted to use the same optimisers that the original author of each model opted to use. As such, all models, excluding ENet [9], were trained using the SGD optimiser. However, as the Adam optimisation algorithm is less sensitive to hyperparameters, which is a valuable factor as computing power is a limited resource, it was of interest to test this algorithm even on models whose developers used the SGD optimiser. The Adam optimiser was tested on the SFNet model with the result, as seen in Figure 4.6, that even if the learning rate for SGD was not optimally chosen, it still performed better than the Adam optimiser.

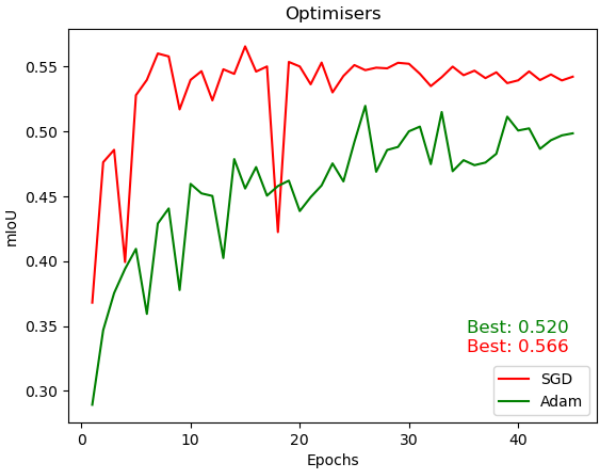
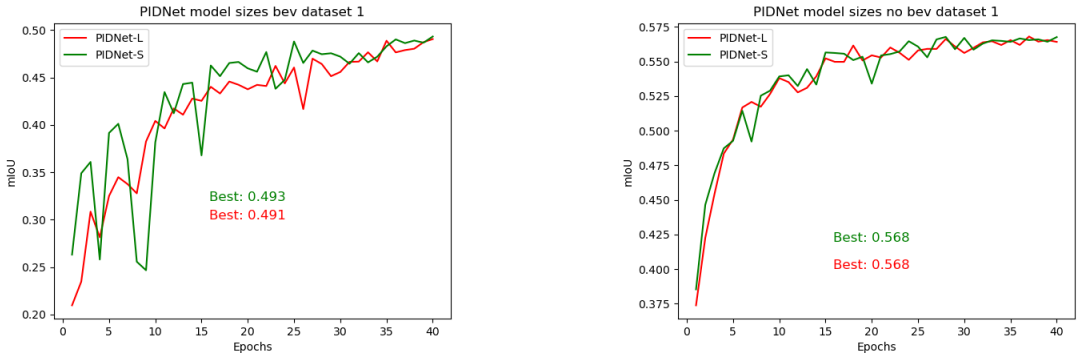


Figure 4.6: The figure shows the mIoU graph of SFNet using the Adam optimiser with a learning rate of 0.001 and the SGD optimiser with a learning rate of 0.01 on non-bev data from the first dataset. It is clear that the SGD outperforms the Adam optimiser in terms of mIoU, even though it is unlikely that the optimal learning for SGD has been found. The Adam optimiser was also tested with other learning rates, such as 0.01, but the results were even worse compared to the learning rate of 0.001 shown in the figure.



(a) **(b)**

Figure 4.7: The figures present the resulting mIoU graphs for different-sized PIDNet models on data from the first dataset1. a) corresponds to bev data with the Gauss labelling scheme and b) to non-bev data with the large soft labelling scheme. As illustrated, there is barely any difference between the performance of the different-sized PIDNet models, which indicates that a more complex model is not necessarily beneficial for the provided data.

4.4 Model complexity PIDNet

The methodology section argued that the small variant of PIDNet would be utilised because of the desire for decreased computing time. Additionally, it was suspected

that the performance of a more complex model would still be limited due to the characteristics and flaws of the data.

To verify this theory, some training runs were performed on the large variant of PIDNet as well. The results, presented in Figure 4.7, show that both the small and large variants of PIDNet achieve essentially the same performance. The lack of performance gains indicates that increased model complexity does not necessarily provide an advantage for the provided data. Consequently, the choice to use the smaller-sized PIDNet model and the shallow ResNet-18 backbone network for SFNet and deeplabv3plus was validated as appropriate for reducing computation time without sacrificing performance.

4.5 Hard label conversion

The project tested two different approaches to convert the soft labels in the provided data to hard labels. The first approach simply assigned the condition with the largest probability as the hard label. The other approach first ensured that the condition with the largest probability exceeded a threshold of 0.7. Otherwise, the label was seen as too controversial and was removed. This was intended to prevent controversial soft labels, where there are multiple conditions with a large probability, from being reduced to a single condition, which could harm the training process.

The results, presented in Figure 4.8, indicate that the direct conversion without thresholds was the superior one and was, therefore, the approach used for all training runs with hard labels.

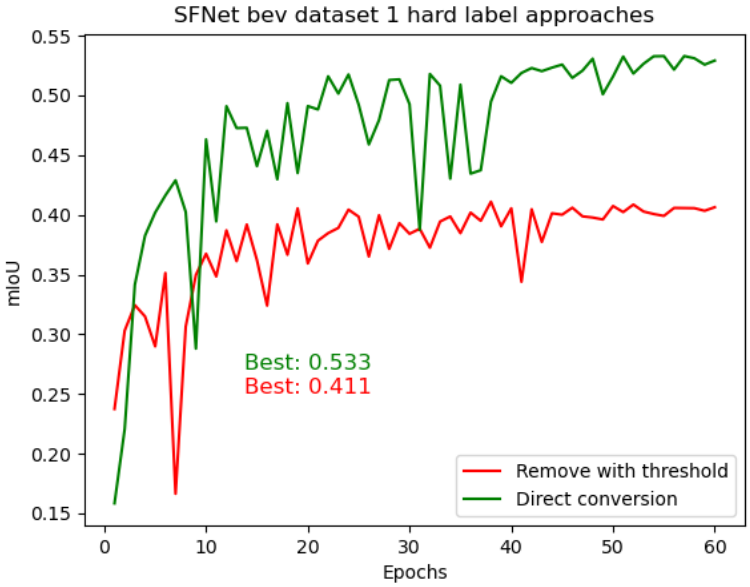


Figure 4.8: The figure shows the results from testing runs with different approaches to convert the soft labels to hard ones. It is clear that the direct conversion, with no regard for ambiguity in the soft labels, is the superior one. The results were generated through training with SFNet and using bev data from the first dataset.

4.6 Tree Energy Loss

The Tree Energy Loss sparse annotated semantic segmentation method failed completely. Unfortunately, due to the method's inherent functions, it created an undesired behaviour where the entire image would be predicted as a single condition. This is most likely due to the way the pseudo labels are set, as these labels are very dependent on the sparse labels. Consequently, it had difficulties predicting conditions on unlabeled pixels that should belong to classes other than those represented by the sparse labels.

As illustrated in Figure 4.9, the method incentivised a behaviour, much like the translation augmentation for non-bev data, where the model classifies the whole image as a single condition. While this behaviour still yielded respectable results in terms of mIoU, since a lot of data only contain labels of a single condition, the goal is semantic segmentation and not image classification. Therefore, the testing of TEL was discontinued.

Figure 4.9: The figure presents input, predictions and labels for the SFNet model trained with TEL on non-bev data in the first dataset. As the figure shows, the TEL method incentivises a behaviour where the model classifies the whole image as a single condition. Additionally, for these particular example images, all labels, excluding those in image 5, are correctly classified, which would result in a high value in mIoU. The 5th image also provides quite an interesting scenario, where the laser annotations indicate an icy road, but the road seems to be entirely made out of snow.

4.7 First dataset

This section will present the results acquired from the first dataset. The results will be divided into two parts, one for bev data and the other for non-bev data, each beginning with a brief summary of the results for the data in question. The summary is followed by subsections for each model, including figures presenting the mIoU graphs, the confusion matrices and images showing input, model predictions and labels.

4.7.1 Bev input data

All data augmentations were used for the bev input data. The labeling techniques used for bev data are soft point labels, hard point labels and Gauss labels. However, as seen in Figure 4.10, the soft point labels are vastly underperforming when trained with the SFNet model. Consequently, the soft point labels were not used for training with the other models.

The hard point labels slightly outperformed the Gauss labels for all models in terms of mIoU. Additionally, the ENet that was utilised in the provided pipeline also underperforms with a mIoU value of 0.48, as illustrated in Figure 4.13, while SFNet, PIDNet, and Deeplabv3plus achieved roughly the same mIoU value of 0.530, as seen in the Figures 4.10, 4.16 and 4.19.

The confusion matrices for each model are found in Figure 4.11, 4.14, 4.17 and 4.20. These indicate that wet and slush, as well as snow and grey ice, were often misclassified as each other. Furthermore, the largest difference between the Gauss labels and hard point labels appears to be in the predictions of slush and wet conditions, where Gauss labels have a preference toward slush conditions, while the hard point labels prefer wet conditions.

Another observation can be made in the example validation images, shown in Figures 4.12, 4.15, 4.18 and 4.21, where all models appear to be struggling with the fourth image set in a darker setting. This could indicate that either the colour augmentation is not aggressive enough or, more reasonably, that the dataset has a substantial majority of frames in a lighter setting, making it difficult for the models to learn to predict images in darker lighting. Otherwise, it is pretty clear for all different machine learning models that for these specific validation images, the hard point labels result in a more refined semantic segmentation, which is further supported by the increased performance in mIoU.

4.7.1.1 SFNet

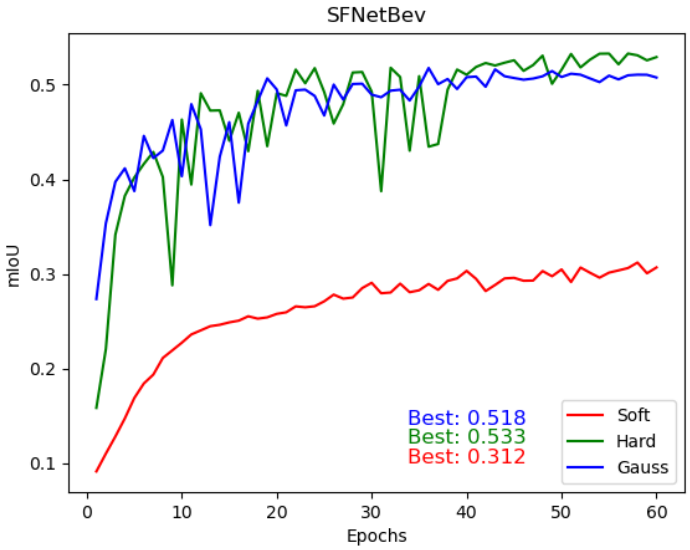
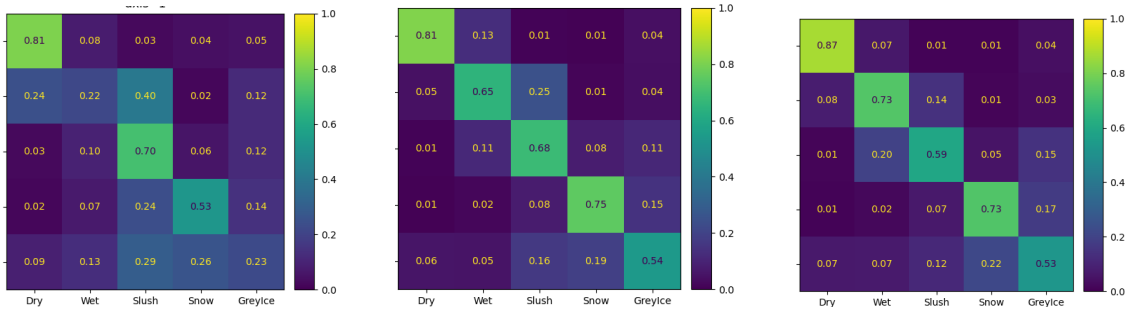
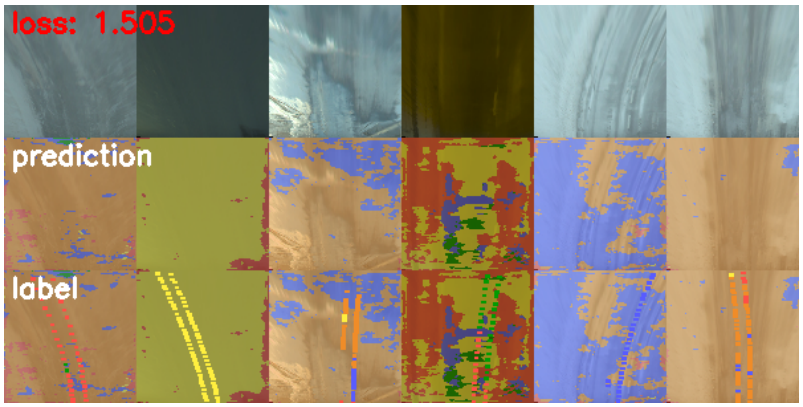


Figure 4.10: The final result for the three different labelling schemes used for bev data trained with the SFNet machine learning model. It is noticeable that soft point labels are not effective, while hard point labels and Gauss labels are quite similar in performance.

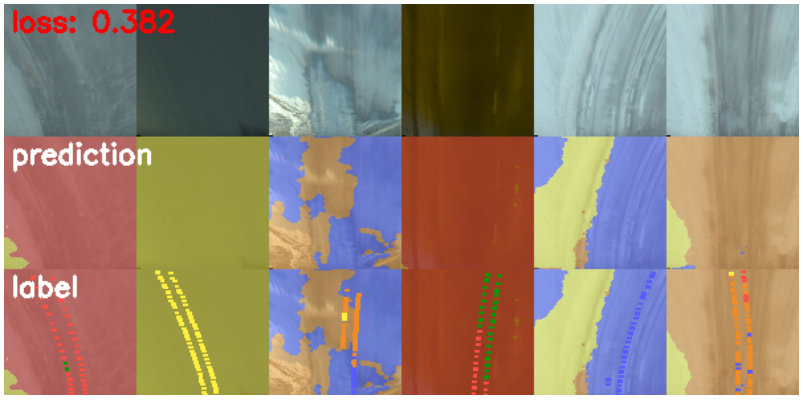


(a) **(b)** **(c)**

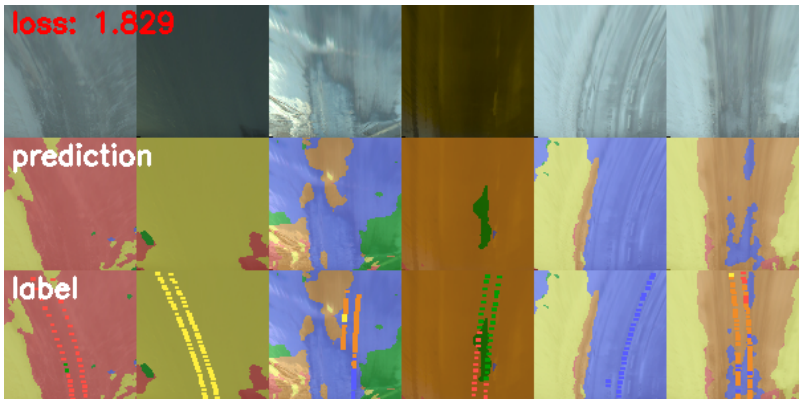
Figure 4.11: Confusion matrices for SFNet bev images in the first dataset. From left to right, the matrices belong to soft point, Gauss, and hard point labels.



(a)



(b)



(c)

Figure 4.12: The figure contains three figures, each representing the results from six validation images while training with SFNet with different labelling schemes. From top to bottom, the labelling scheme while training is: a) soft point labels, b) Gauss labels and c) hard point labels.

4.7.1.2 ENet

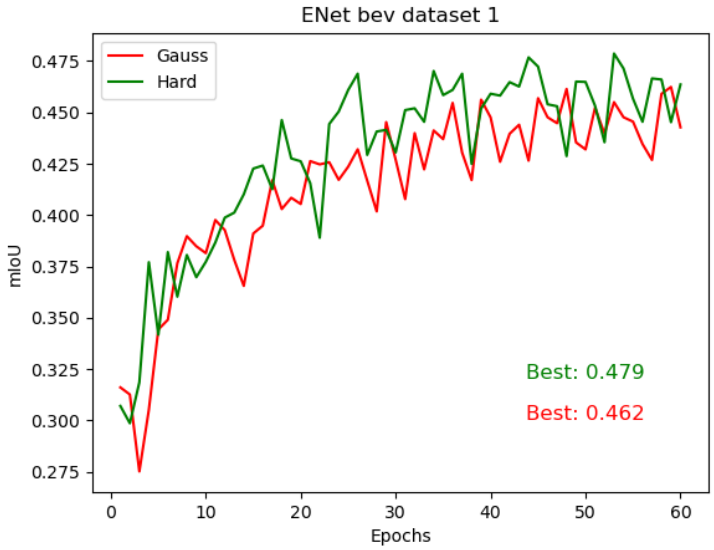
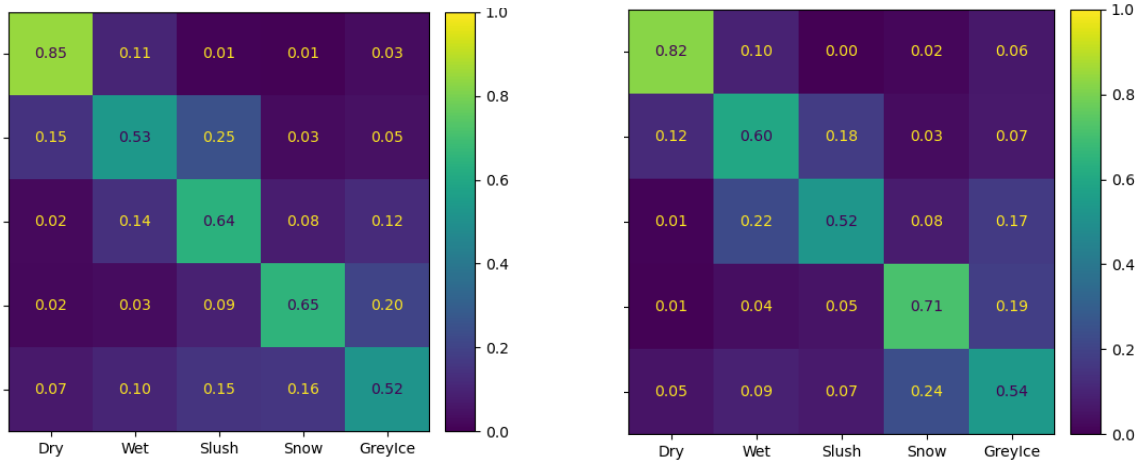


Figure 4.13: mIoU graph for bev data on the first dataset trained with ENet model. Similarly to SFNet, the hard point labels slightly outperform the Gauss labels. However, the performance is significantly lower than the performance of SFNet in the terms of mIoU.



(a) **(b)**

Figure 4.14: The resulting confusion matrices for bev data trained on the first dataset with Gauss labels a) and hard point labels b). The most notable difference between the two labelling techniques is that the Gauss labels are slightly better at classifying slush conditions, while the hard point labels are better at wet and snow conditions. The overall behaviour is otherwise quite similar.

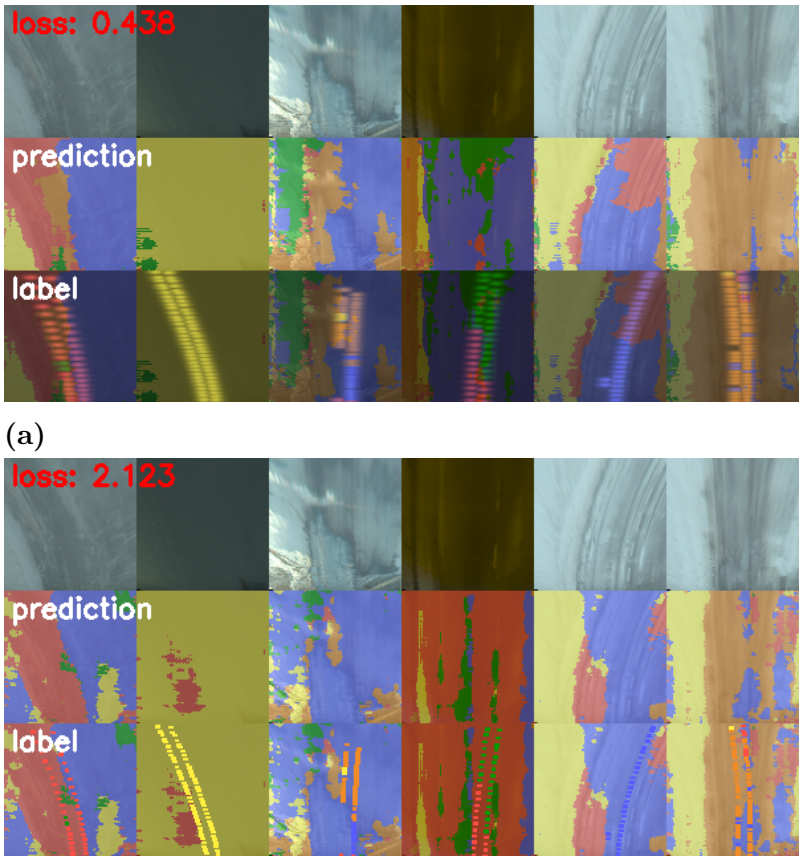
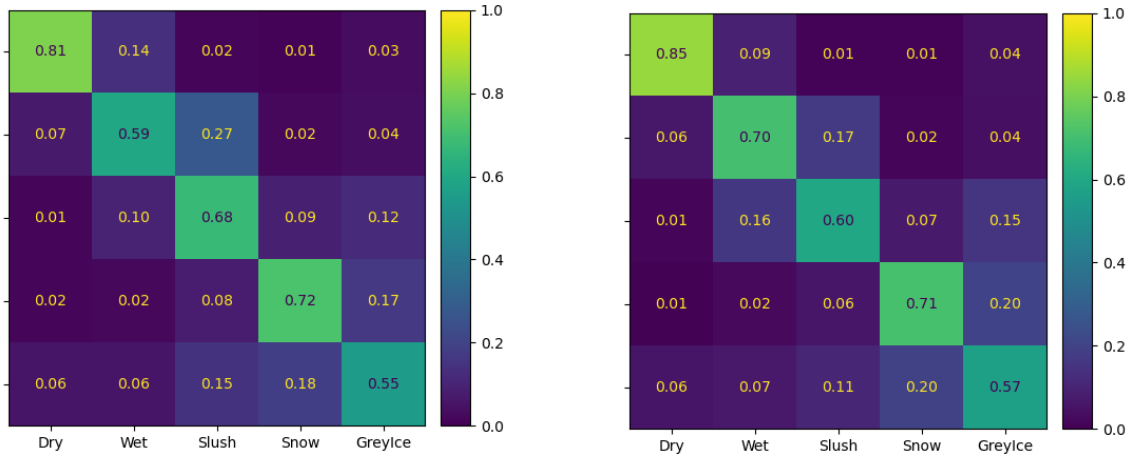


Figure 4.15: Two different figures, each representing the results from six validation images while training with ENet on bev data from the first dataset with different labelling techniques. a) represents the predictions when training with Gauss labels, and b) represents the result from training with hard point labels.

4.7.1.3 PIDNet

Figure 4.16: The results for PIDNet when trained on bev data with all augmentations applied. Similarly to the results for SFNet, the hard point labels slightly outperform the Gauss labels according to the mIoU index. Note that the soft labelling scheme is omitted here since the results were severely underwhelming and, therefore, of no interest.



(a) **(b)**

Figure 4.17: Confusion matrices for PIDNet trained on bev data from the first dataset. From left to right the confusion matrices originate from a) Gauss labels and b) hard point labels.

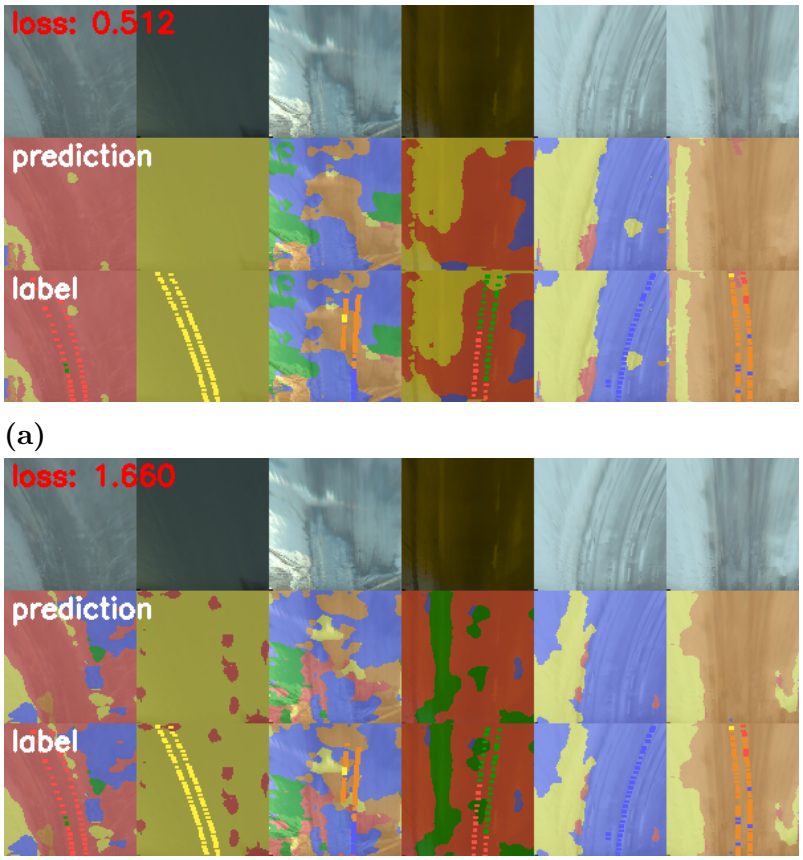


Figure 4.18: a) represents the results from PIDNet trained on bev data on the first dataset with Gauss labels, while b) presents results from PIDNet the same data but with hard point labels.

4.7.1.4 DeepLab

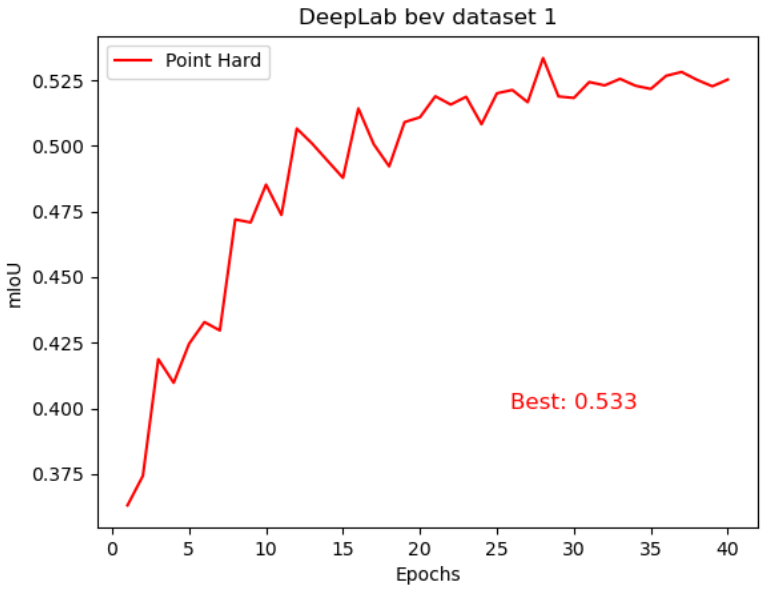


Figure 4.19: The mIoU graph of Deeplabv3plus trained on bev data from the first dataset with hard point labels.

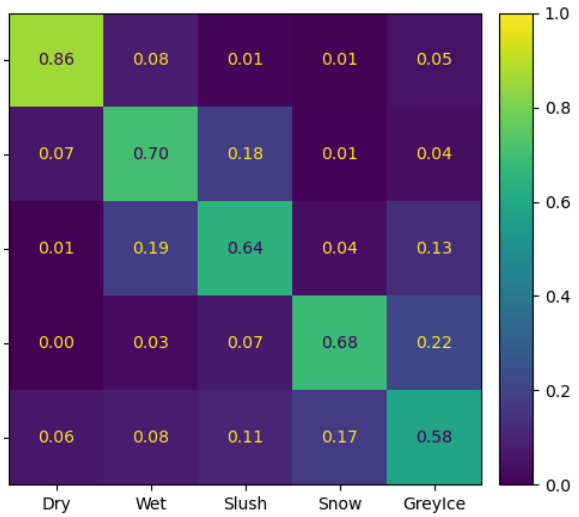


Figure 4.20: Confusion matrix of Deeplabv3Plus trained on bev data from the first dataset with hard point labels.

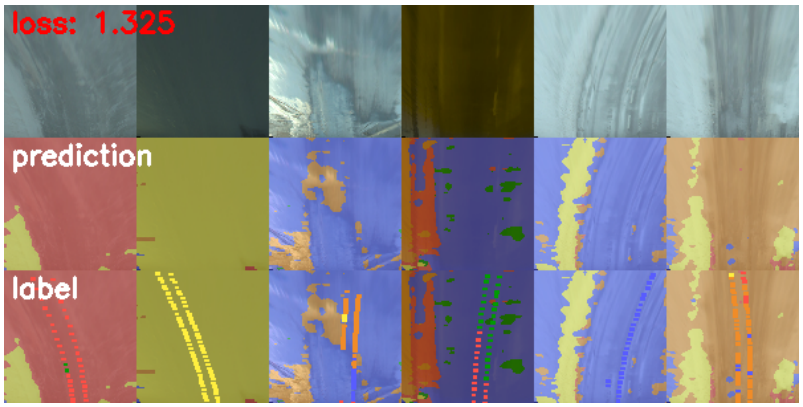


Figure 4.21: The figure showcases how the Deeplabv3plus model trained on bev data from the first dataset predicted six image set examples from the validation data.

4.7.2 Non-bev input data

This section will include the results from training on non-bev data from the first dataset. In this case, no augmentations were applied to the training dataset.

Interestingly, from Figure 4.22, it appears that the labelling technique did not affect the mIoU performance of the SFNet model. In comparison, Figure 4.28 shows that the PIDNet achieves slightly better performance for the large labels than the hard point ones, where the large soft labels were the best-performing ones. Deeplabv3plus, on the other hand, achieves its best performance with hard point labels. However, the baseline ENet network trained on large soft labels slightly outperforms every other model.

Similarly to the bev data in the same dataset, the confusion matrices of each model indicate the same misclassification between slush and wet as well as snow and grey ice. This further supports the concept of biases in the data from the first dataset. According to the matrices in Figure 4.23, 4.30 and 4.33, the main difference between the different labelling techniques is that the large soft labels massively outperform the hard labels when it comes to the slush conditions. However, the increased performance for the slush condition did come with the price of decreased performance of other conditions, especially for the SFNet model.

Additionally, the images in Figures 4.24, 4.27, 4.29 and 4.33 suggest that, much like the bev data for the first dataset, frames in darker settings still presented a problem for the model. Each model shares multiple predictions that are almost exactly alike, but there are generally one or two images that induce larger differences between the models' predictions. However, all models struggled with the outskirts of the road, which probably comes from the failure to spread out the sparse labels in the training data and is discussed further in the discussion section 5. There are, however, no clear indications in these images of which of the labelling techniques performed the best. In particular, when comparing the results from SFNet and PIDNet found in Figure 4.24 and 4.29, the hard point labels appear to have performed better than the soft large labels in image 5 and worse in image 6 for PIDNet, but vice versa for

SFNet.

4.7.2.1 SFNet

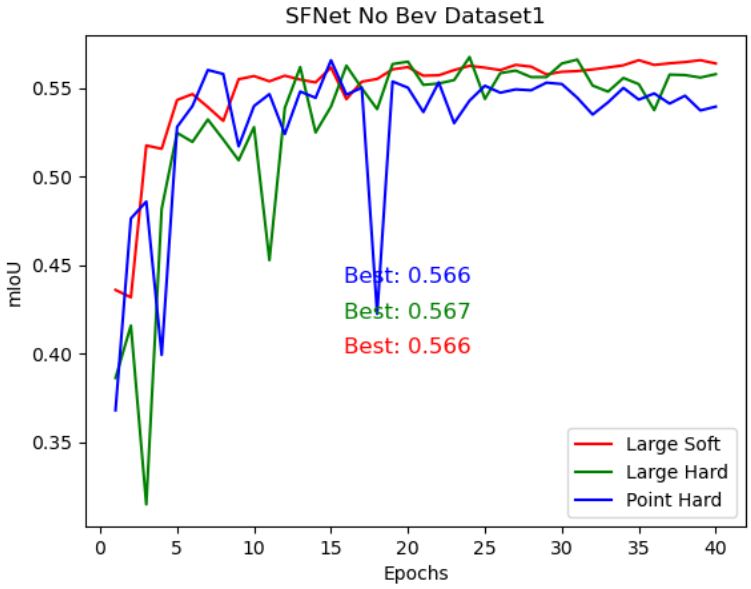
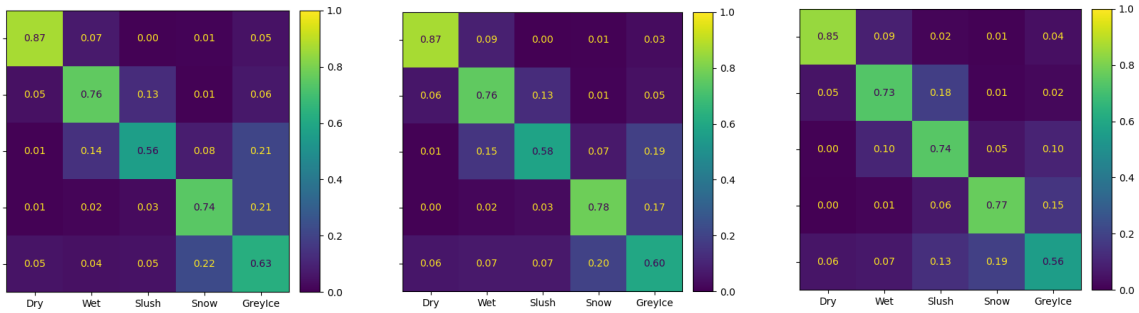
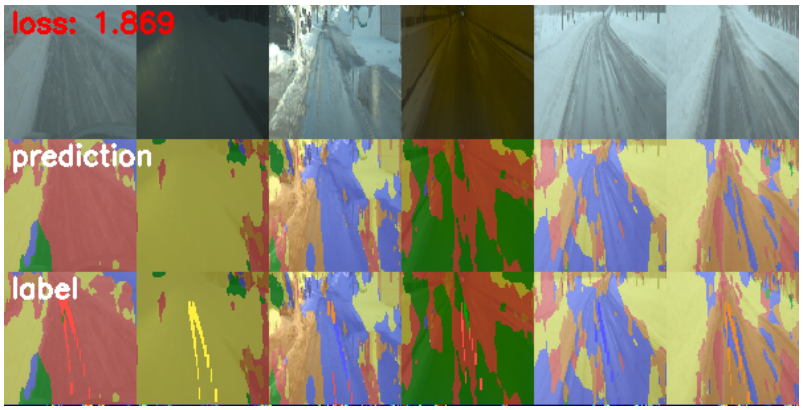


Figure 4.22: The mIoU curve for SFNet trained on non-bev data from the first dataset. Almost all labelling techniques performed quite similarly in terms of mIoU.

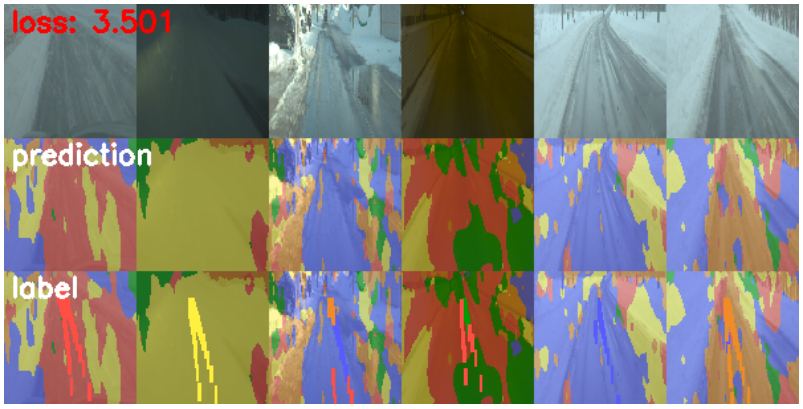


(a) **(b)** **(c)**

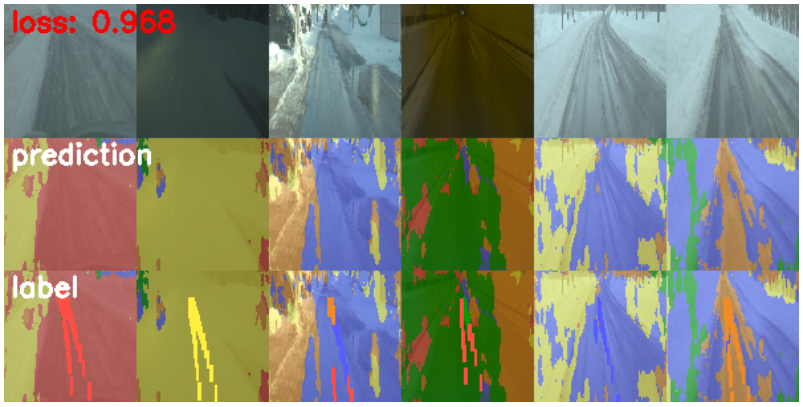
Figure 4.23: Confusion matrices from SFNet trained on non-bev data from the first dataset for the labelling schemes: a) hard point labels, b) large hard labels and c) large soft labels.



(a)



(b)



(c)

Figure 4.24: The resulting semantic segmentations performed by SFNet trained on non-bev data from the first dataset. a) corresponds to hard point labels, b) large hard labels and c) large soft labels.

4.7.2.2 ENet

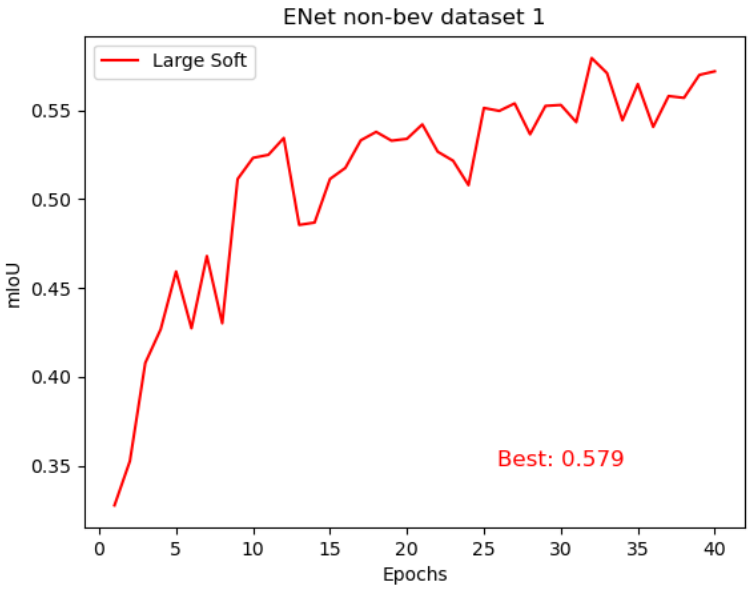


Figure 4.25: The mIoU curve for ENet trained on non-bev data from the first dataset without any augmentations. The labelling technique ENet uses in no-bev cases is the large soft labels. The mIoU peaks at a respectable value of 0.579.

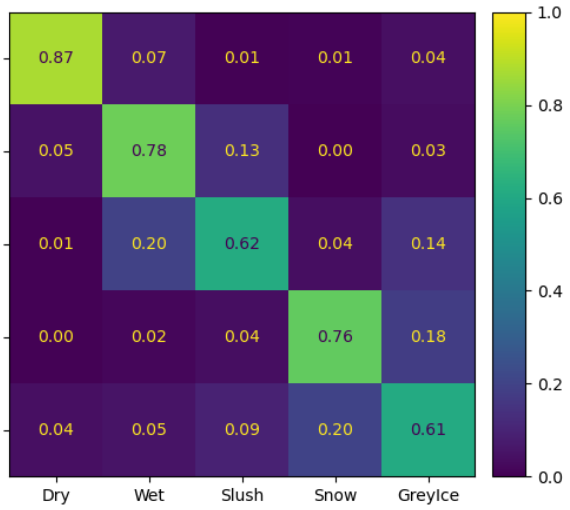


Figure 4.26: The confusion matrix for ENet trained on non-bev data from the first dataset without any data augmentations applied. The model struggled the most with slush and ice, where slush is often misrepresented as a wet or ice condition, and ice is misrepresented as a snow or slush condition. This could be due to the similar characteristics of the conditions.

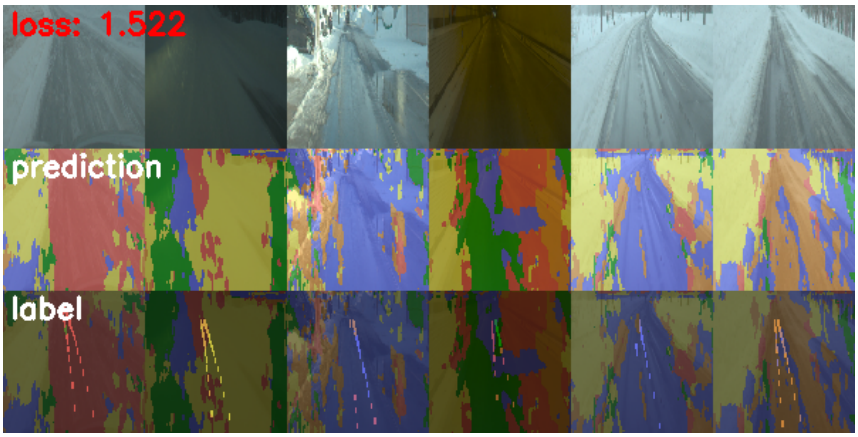


Figure 4.27: Predictions from ENet trained on the non-bev data from the first dataset without augmentation. The model is generally able to predict the road conditions. However, it still struggles in darker settings and towards the edges. Now, remember that the edges are not actually part of the road and are not as interesting.

4.7.2.3 PIDNet

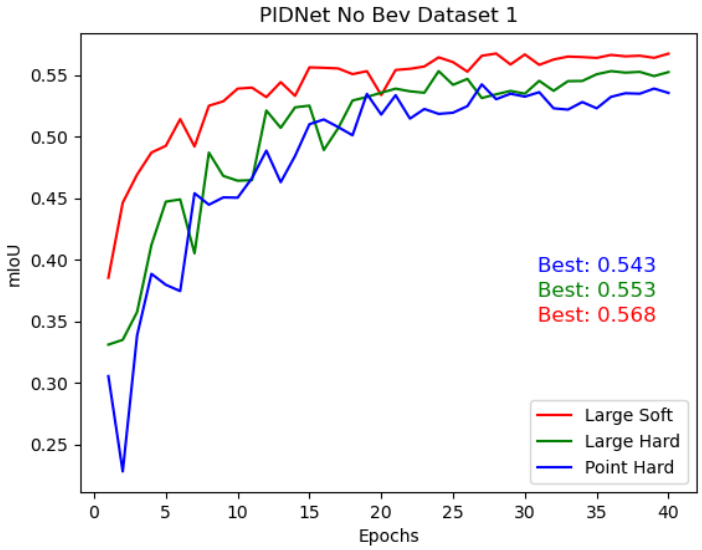
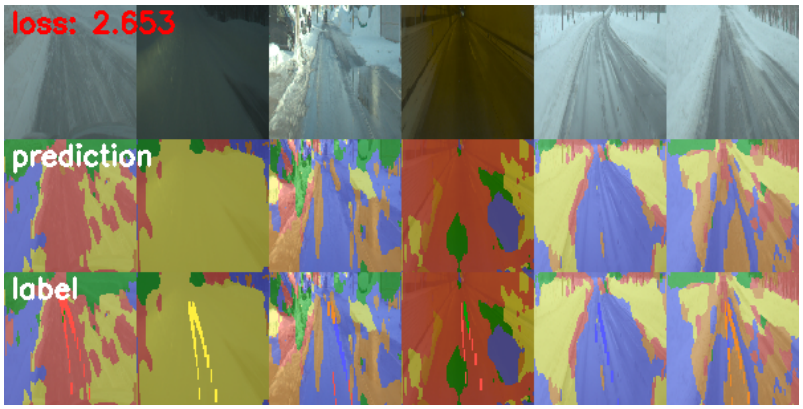
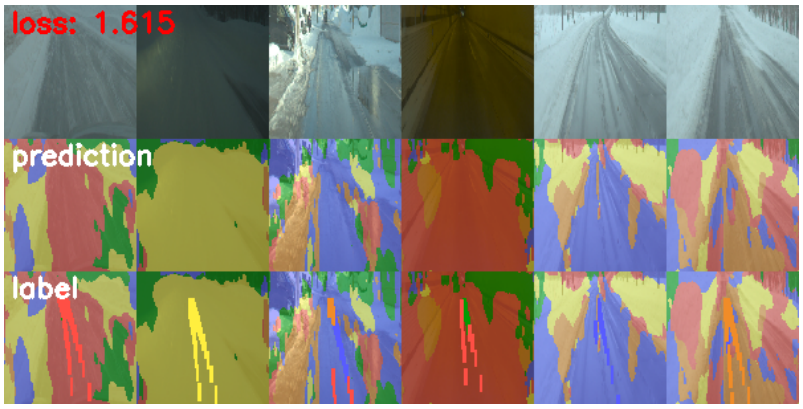


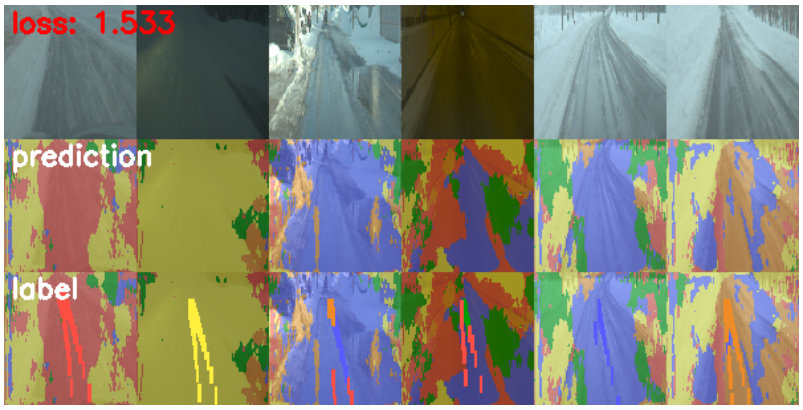
Figure 4.28: The resulting mIoU graphs from PIDNet trained on the first dataset using different labelling techniques. The large soft labels slightly outperform the other labels but cannot directly challenge the performance of the ENet model applied to the same dataset.



(a)

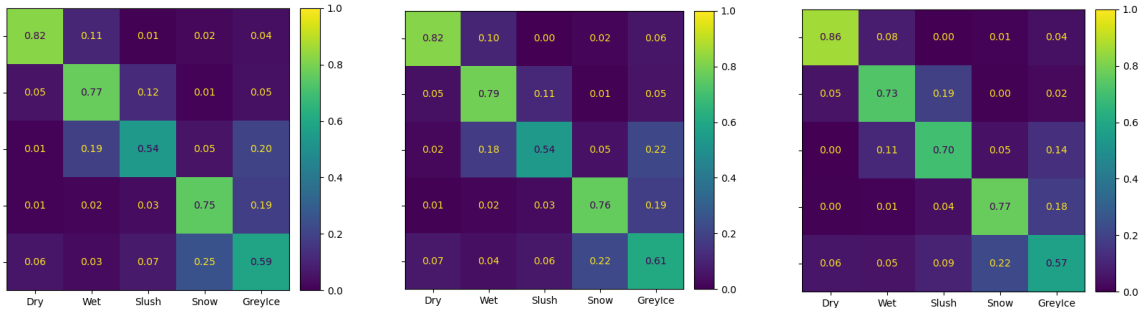


(b)



(c)

Figure 4.29: The figure presents example image sets from PIDNet training on hard point labels a), hard large labels b) and soft large labels c). Generally, the results seem to improve the further down one goes, with the best results from the soft labelling scheme, which corresponds well with the mIoU graph in Figure 4.28. However, one can argue that the hard point labels perform better on the 3rd, 4th and 5th images than the soft labelling scheme. The 1st and 6th images are still classified better by the soft labelling scheme.



(a) **(b)** **(c)**

Figure 4.30: The confusion matrices from training the PIDNet model with different labelling schemes. a), b) and c) represent hard point labels, hard large labels and soft large labels, respectively. The hard labelling schemes generally struggle more with slush conditions but perform better on wet conditions than soft labelling.

4.7.2.4 DeepLab

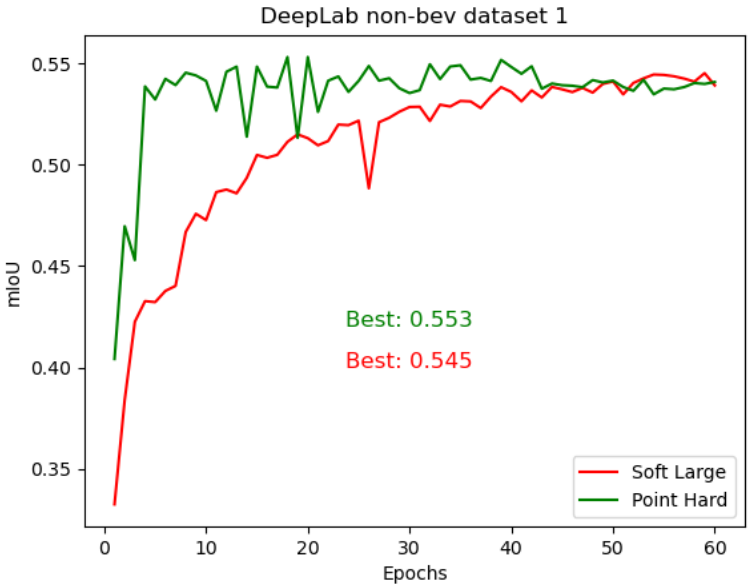
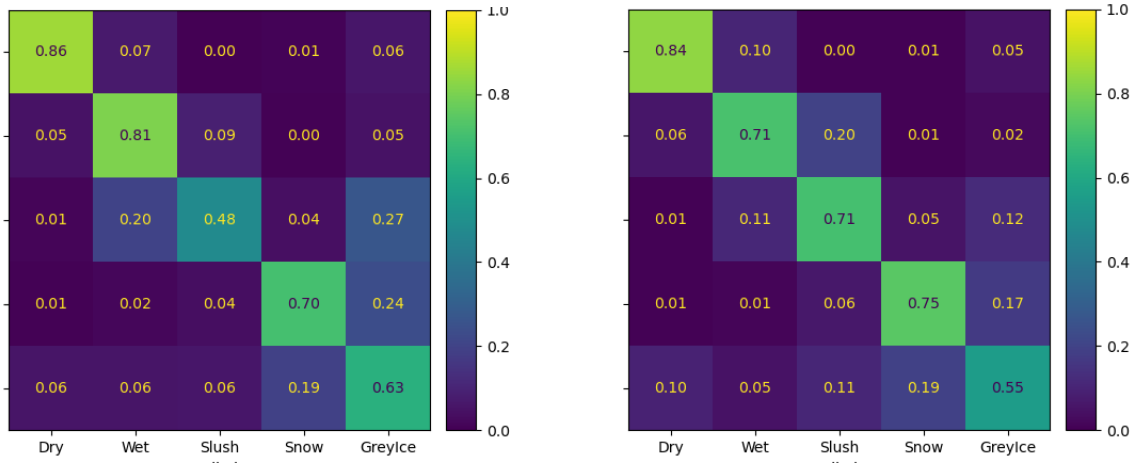
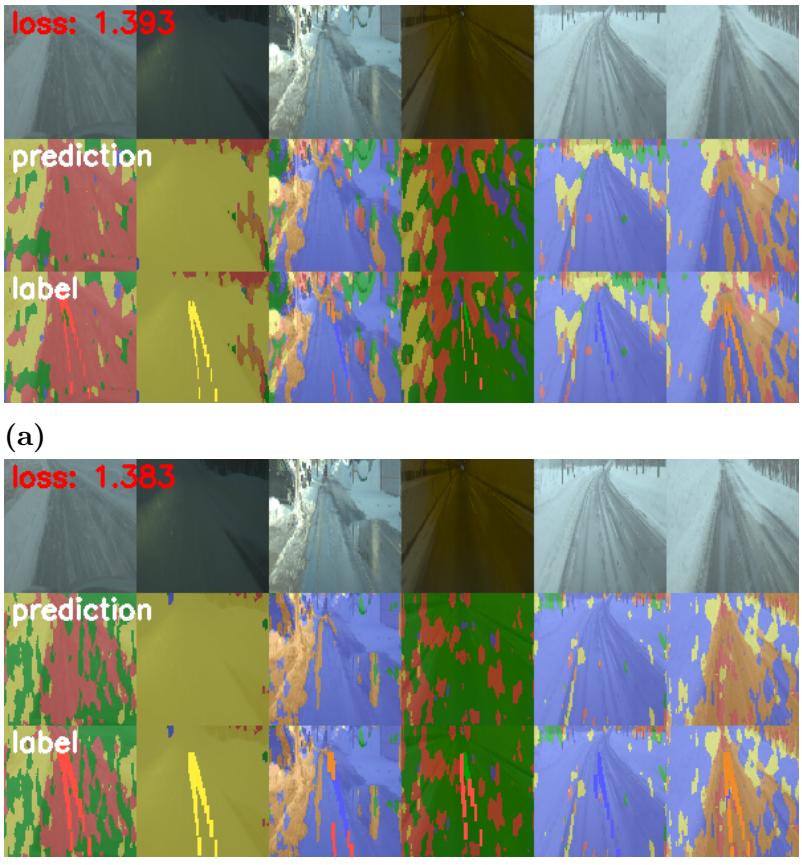


Figure 4.31: mIoU graph of Deeplabv3plus trained on non-bev data from the first dataset.



(a) **(b)**

Figure 4.32: Confusion matrices for deeplabv3plus trained on non-bev data from the first dataset with a) hard point labels and b) soft large labels



(a)

(b)

Figure 4.33: Resulting predictions of 6 image sets from the validation data for the deeplabv3plus model trained on non-bev data from the first dataset with a) hard point labels and b) soft large labels.

4.8 Second Dataset

This section will briefly present the results achieved while training the models with data from the second dataset.

4.8.1 Bev input data

For bev data, all data augmentations are applied. As seen in the results for the first dataset, the soft point labelling technique underperformed substantially, and it is unlikely that it would prove to be much more efficient in the second dataset. As such, the soft point labelling technique was not used when training models on data from the second dataset.

Unlike the results for the first dataset on bev data, the Gauss labelling technique appears to have performed better than the hard point labels, as seen in Figure 4.34 and 4.40. This is likely due to the characteristics of this dataset, which contained more homogeneous data. Consequently, the assumptions made for Gauss labels, that pixels close to the labelled pixel share the same properties, held better for this dataset. Additionally, the ENet model was underperforming significantly compared to the other models in terms of mIoU, especially with Gauss labels.

The confusion matrices for all models also show that there was often misclassification between dry and wet, with wet being misclassified as dry in a worrying number of cases, as illustrated in Figure Figure 4.35, 4.38, 4.41 and 4.44. Additionally, as with the case for Bev data on the first dataset, snow and ice were often misclassified as each other. In particular, for the ENet model, wet conditions were more often incorrectly classified as dry than correctly as wet. The matrices also indicate that the Gauss labels were significantly better at predicting the slush condition but also induced a better performance for wet and ice conditions, while snow conditions exhibited a slightly reduced performance. Additionally, the confusion matrices for SFNet, PIDNet and Deeplab are essentially equivalent, indicating that biases in the data limited the performance and not the choice of model.

The images for this dataset are perhaps a little less interesting than those presented for the first dataset, which is another factor to keep in mind for the second dataset. Even though it did contain more data overall, a lot of the data was homogeneous, where all sparse labels were of the same condition, leading to less interesting scenarios in individual frames. Additionally, it was discussed in the methodology section 3.1.2 that the second dataset would suffer from lesser quality data compared to the first, which is quite evident when studying the example images in Figure 4.36, 4.39, 4.42 and 4.45.

However, considering the image sets shown in the previously mentioned figures, it is a bit ambiguous whether the models performed well even in darker settings. Image sets 1,2, and 3 are set in a dark setting and have quite decent results for all models, but image set 5, which is in an even darker setting, presents quite different results for each model. This could be attributed to the ambiguous nature of that image set, as it is difficult even for a human to discern the actual road conditions there. Additionally, image set 4, containing a wet road in a bright setting, causes significant

problems for the ENet model, while the other models were able to correctly predict the road condition with only slight difficulties. Furthermore, the figures indicate some difficulties in predicting the conditions on the outskirts of the road, especially for the PIDNet and DeepLab models. In accordance with the mIoU results, the Gaussian labels also seem to result in slightly more reasonable images compared to the hard point labels, especially for the SFNet model.

4.8.1.1 SFNet

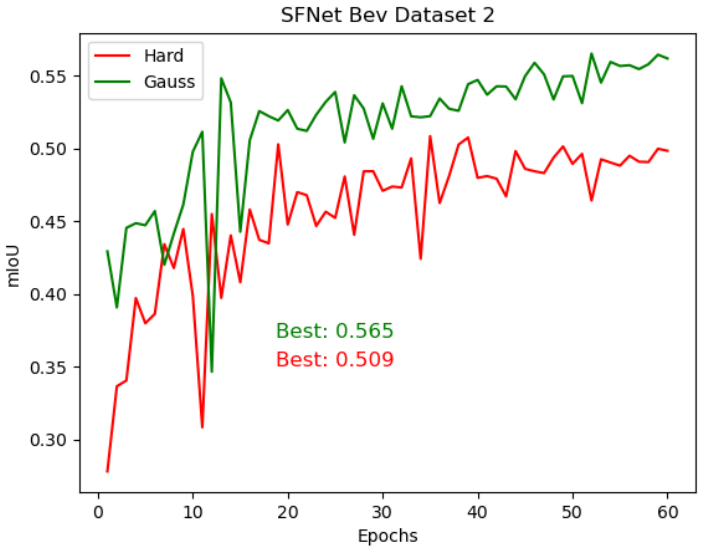
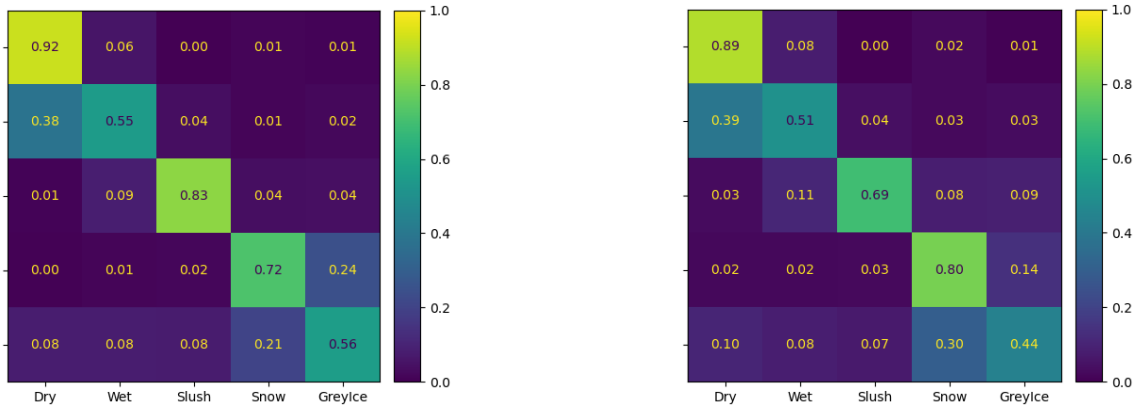


Figure 4.34: As seen in the figure, the Gauss labelling scheme performs significantly on the second dataset. In contrast, the hard labelling scheme performed slightly better than the Gauss labels in the first dataset.



(a) **(b)**

Figure 4.35: The two confusion matrices for SFNet trained on bev data from the second dataset. a) presents the results with Gaussian labels and b) with hard point labels. The most noticeable difference between the two is that the Gauss labels are significantly better at predicting slush and grey ice conditions.

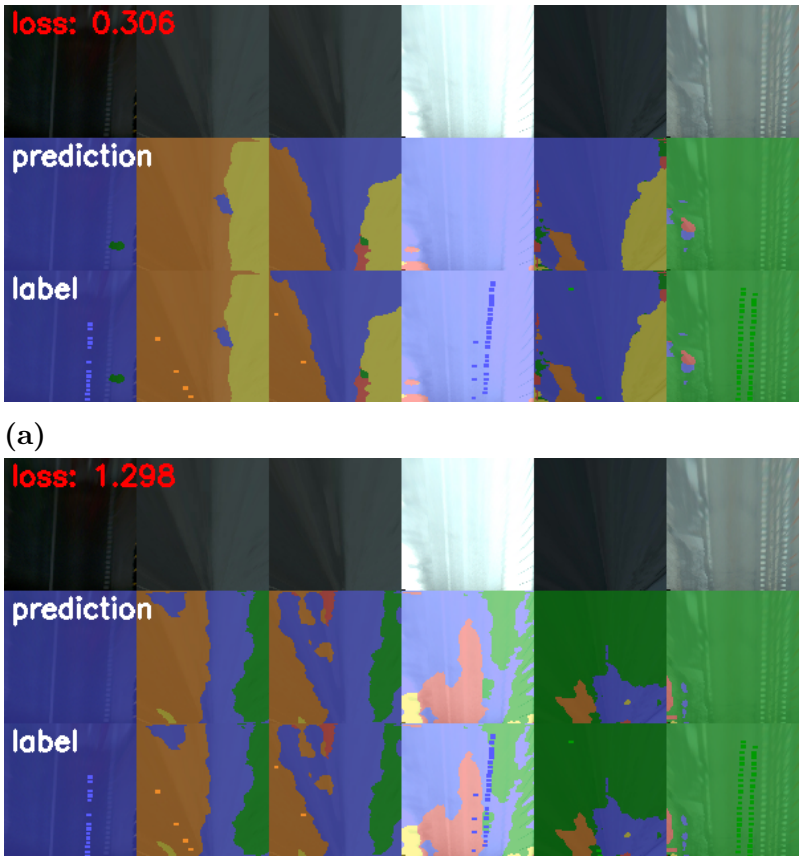


Figure 4.36: Example images from the second dataset with Gaussian labels a) and hard point labels b). Compared to the first dataset, these images are generally in a much darker environment and are more homogeneous than the example images for the first dataset. That is, the images generally have all their labels of a single condition. Even so, there is a noticeable difference between the two labelling schemes, where the Gauss labels provide much more reasonable classifications, especially focusing on the second and third image sets.

4.8.1.2 ENet

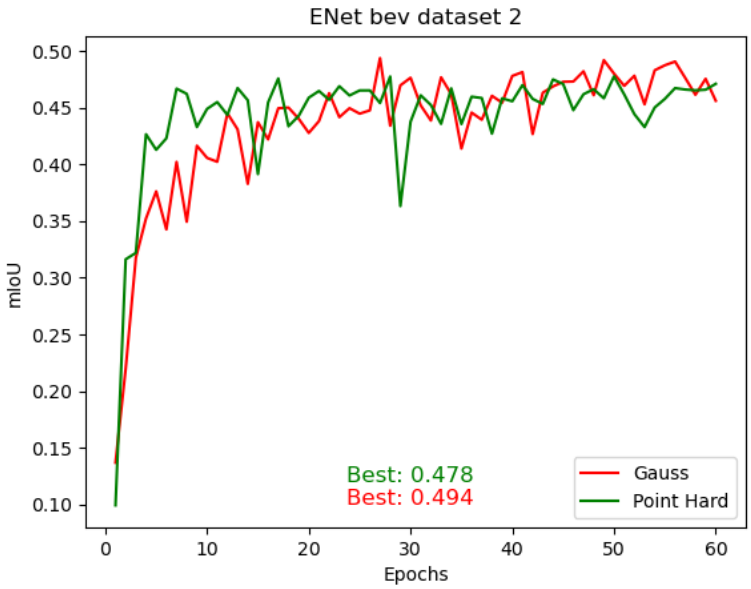
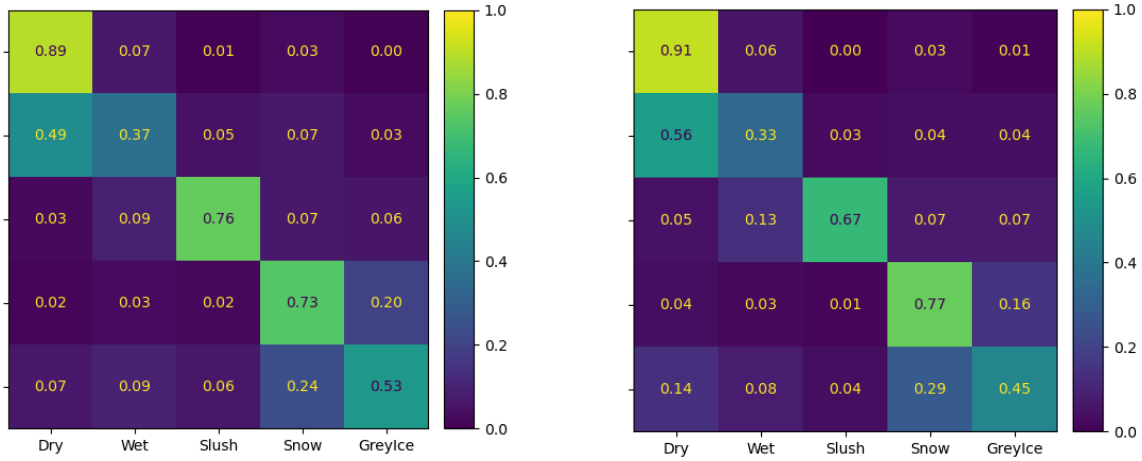


Figure 4.37: mIoU graph for bev data on the second dataset trained with ENet model.



(a) **(b)**

Figure 4.38: The resulting confusion matrix from the ENet model when trained on bev data from the second dataset. Subfigure a) presents the results of Gauss labels, while b) shows the results of hard point labels. The matrices show that the model struggles with the wet condition, commonly misrepresented as dry. Additionally, the model has difficulty distinguishing between the snow and grey ice conditions.

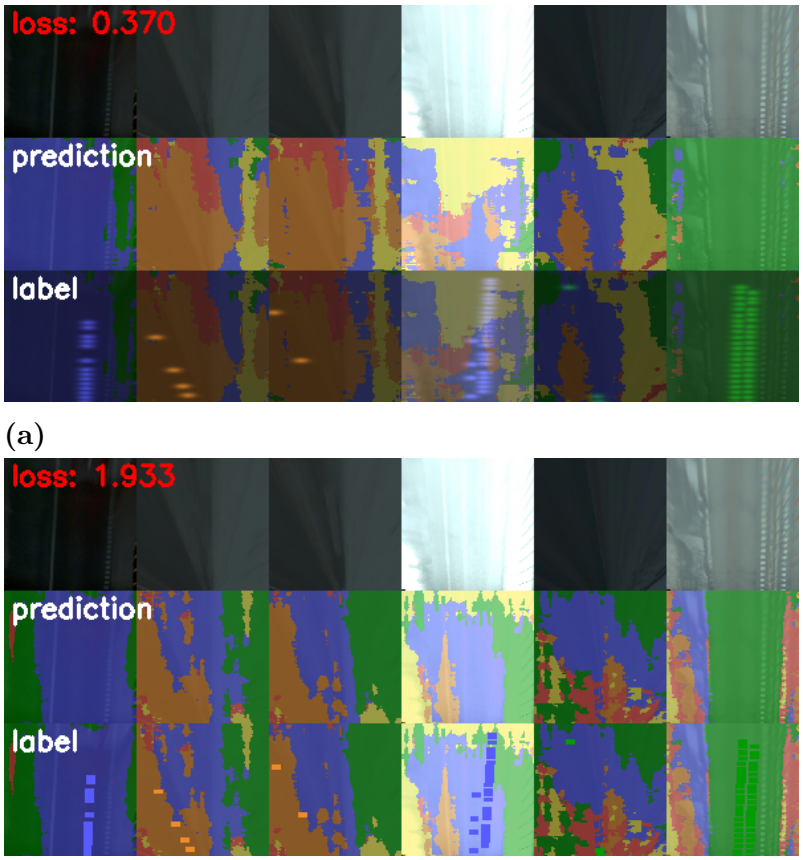
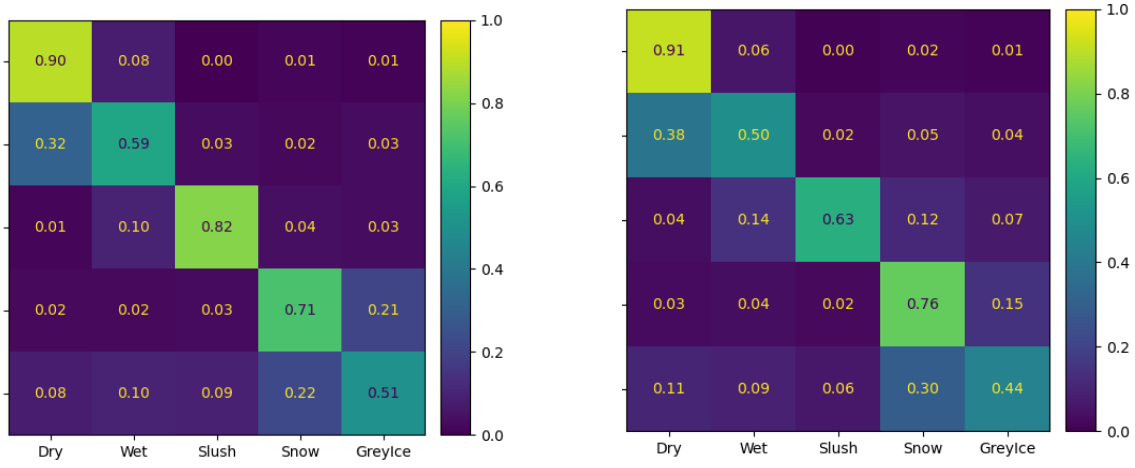


Figure 4.39: ENet predictions on example images on bev data from the second dataset with Gauss labels a) and hard point labels b). There is a slight difference in image set 4, where the hard point label technique allows the model to correctly classify the wet road even in such a bright setting. However, the hard point labels are slightly worse at classifying the outskirts of the road in image set 2 and 3.

4.8.1.3 PIDNet

Figure 4.40: The mIoU graph for PIDNet trained on bev data from the second dataset. The performances are quite similar to those of SFNet in Figure 4.34, which corresponds well to the same comparison for bev data from the first dataset, where the models also had similar mIoU performance.



(a) **(b)**

Figure 4.41: Confusion matrices for PIDNet trained on bev data from the second dataset. a) corresponds to Gaussian labels and b) to hard point labels. With slight differences of a few percentages, the matrices are almost equivalent to those of SFNet, which indicates some biases in the data.

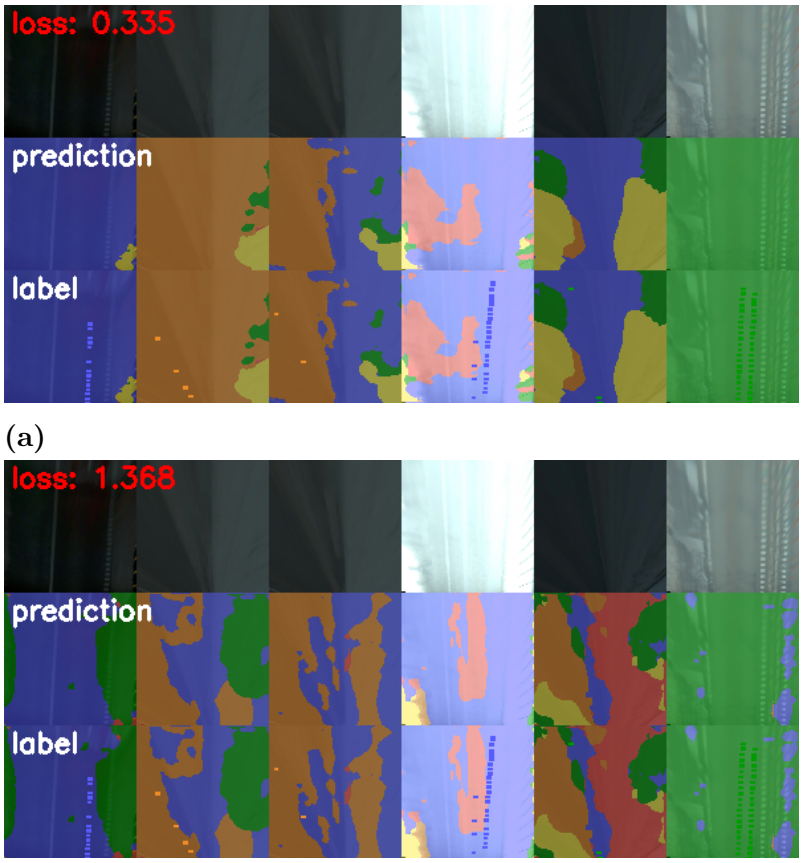


Figure 4.42: Example validation images for PIDNet trained on bev data from the second dataset with a) Gaussian labels and b) hard point labels.

4.8.1.4 DeepLab

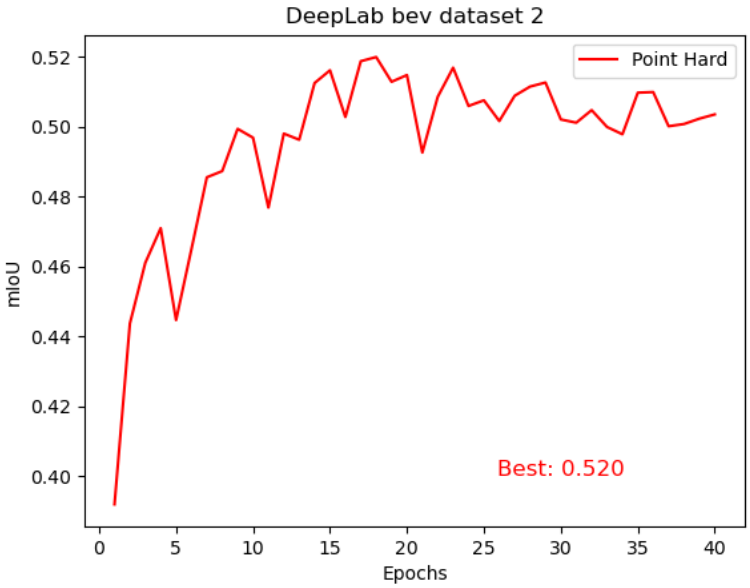


Figure 4.43: The mIoU graph of Deeplabv3plus trained on bev data from the second dataset with hard point labels.

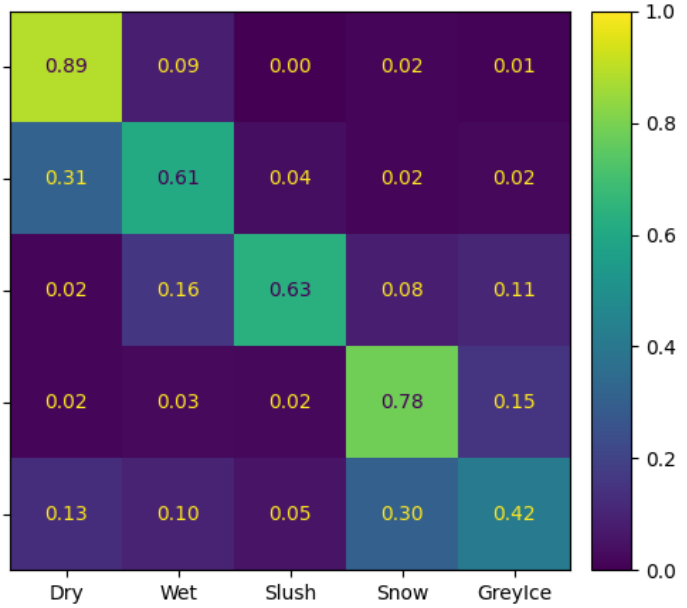


Figure 4.44: Confusion matrix of Deeplabv3Plus trained on bev data from the second dataset with hard point labels.

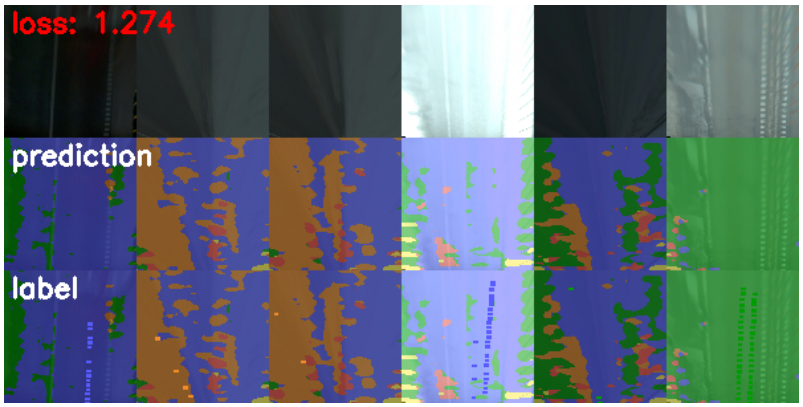


Figure 4.45: The figure showcases how the Deeplabv3plus model trained on bev data from the second dataset predicted six image set examples from the validation data.

4.8.2 Non bev input data

In the second dataset, Gaussian blur augmentation, brightness, saturation, hue, and contrast augmentations wee applied. These were deemed to have a slight beneficial contribution to performance. While they may not significantly affect the mIoU value, they were expected to improve the model's ability to generalise to new data.

In comparison to non-bev data from the first dataset, the SFNet and PIDNet outperformed the ENet model, if only by a margin, as illustrated in Figure 4.46, 4.49 and 4.52. However, the performance, much like non-bev data in the first dataset, for Deeplab was lacking compared to the other models, as seen in Figure 4.55. Additionally, the large labels heavily outperformed the point labels for both SFNet and PIDNet, where the soft large labels performed the best. This was also the case for PIDNet on non-bev data in the first dataset, but for SFNet, the labelling scheme did not have much effect on the mIoU performance on that dataset. For the Deeplab model, the soft large labelling technique performed worse than the hard point labelling technique.

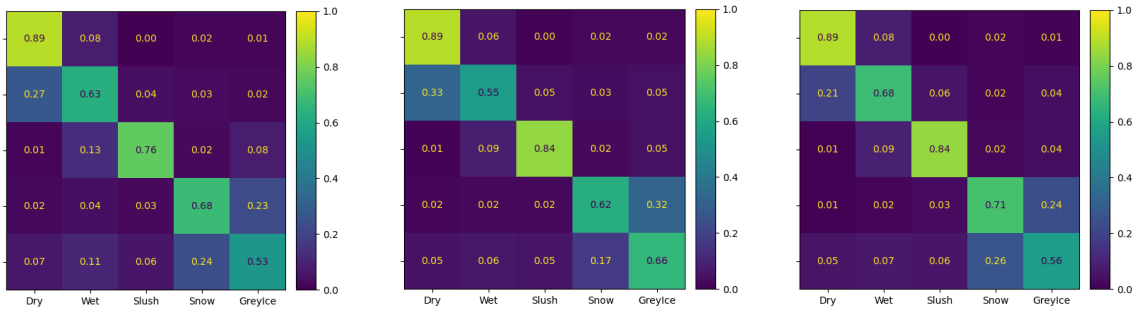
As seen in the confusion matrices in Figure 4.47, 4.50, 4.54, and 4.56, the non-bev data suffered from the same problems as the bev data. That is, the misclassification of wet as dry and between snow and ice. However, the misclassification was not quite as extreme as for the bev data. Nonetheless, this further indicates biases in the second dataset.

Much like previously revealed results, the images in Figure 4.48, 4.51, 4.53 and 4.57 showcases that, while the predictions on the actual road was reasonable, the models had problems classifying the conditions on the outskirts of the road. Additionally, image set 4 with a wet road in a bright environment was misclassified for all models and labelling techniques, where the road was classified as icy for all models except ENet, which classified it as snow. Image set 5 also indicates that the models struggled to classify conditions during the night, while image set 1,2,3 shows slight success in less dark settings. Otherwise, it isn't easy to discern any major improvements in performance between the different labelling techniques used in the images, even

though there is quite a large difference in mIoU performance between them.

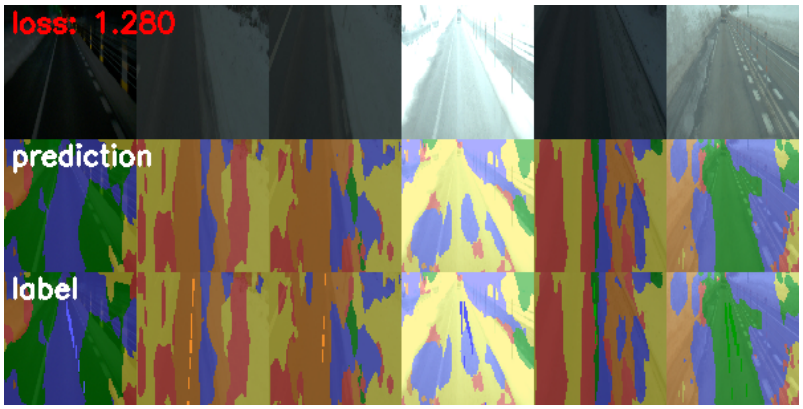
4.8.2.1 SFNet

Figure 4.46: mIoU graph for different labelling schemes trained on no bev data from the second dataset. Similarly to the first dataset the large soft labels perform the best, while large hard labels perform slightly better than hard point labels.

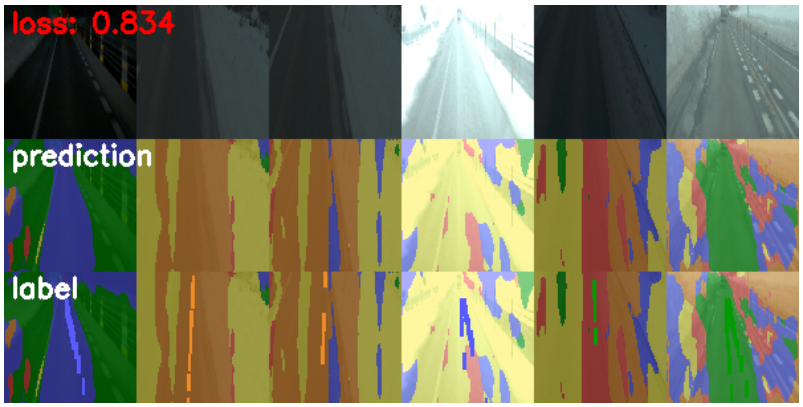


(a) **(b)** **(c)**

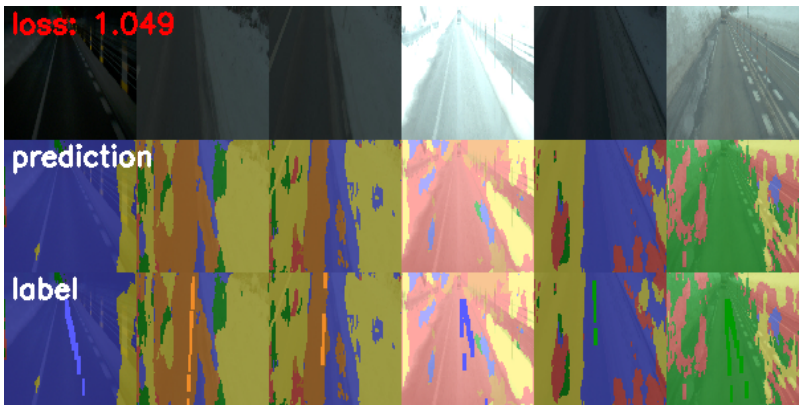
Figure 4.47: Confusion matrices from SFNet trained on non-bev data from the second dataset for the labelling schemes: a) hard point labels, b) large hard labels and c) large soft labels. The overarching theme for all labelling schemes is that the model often misclassifies wet conditions as dry. The ice condition is also commonly misclassified as snow and vice versa.



(a)



(b)



(c)

Figure 4.48: The example image sets from the validation data in the second dataset. What is most noticeable here is that the brightness of the images significantly affects the model's predictions. This is most notable in the 4th image set, where a wet road on a sunny day is misclassified as snow or ice for all labelling schemes. Another interesting observation can be made in the fifth image set, where the predictions vary quite a bit for each labelling scheme. For a human, it looks like a classic winter road at night, but the sparse labels imply a dry road, and the best-performing large soft labelling scheme indicates that it is a wet road with snow to the left.

4.8.2.2 ENet

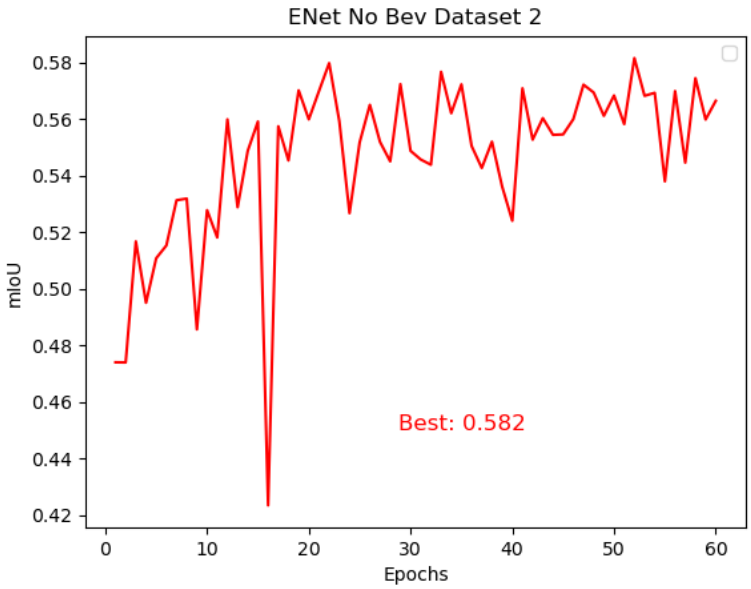


Figure 4.49: The mIoU graph acquired from the ENet model trained on non-bev data from the second dataset.

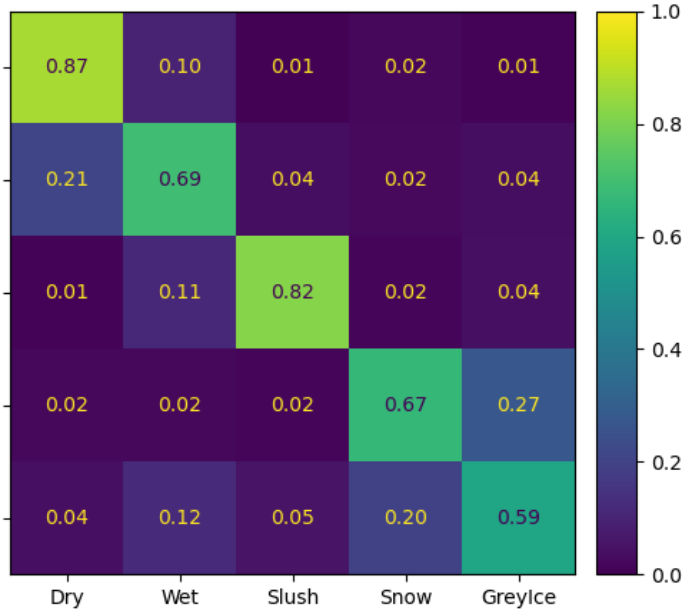


Figure 4.50: The confusion matrix for ENet model trained on non-bev data from the second dataset. The model struggles most with misclassifying wet conditions as dry and seems to have difficulty separating snow from ice.

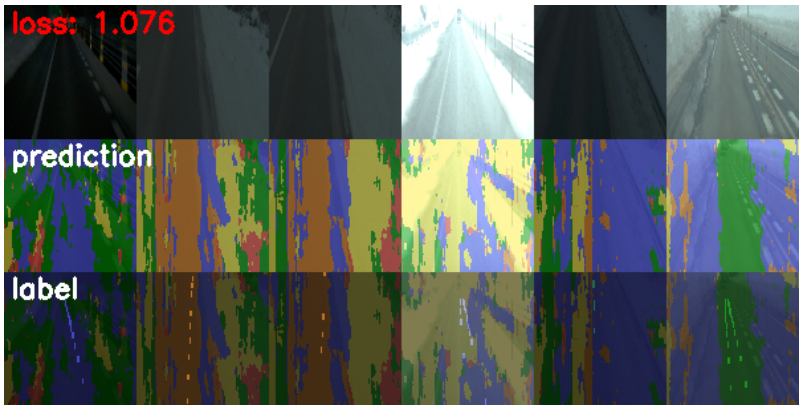


Figure 4.51: The figure presents six different image sets from the validation data in the second dataset. The model performs quite well on the evening snowy roads in images 2 and 3 but fails quite severely on the bright, wet road in image 4, which is misclassified as snow. The predictions of image 5 are quite ambiguous, but at the same time, the image is also quite challenging to classify, mostly due to the lighting of the image.

4.8.2.3 PIDNet

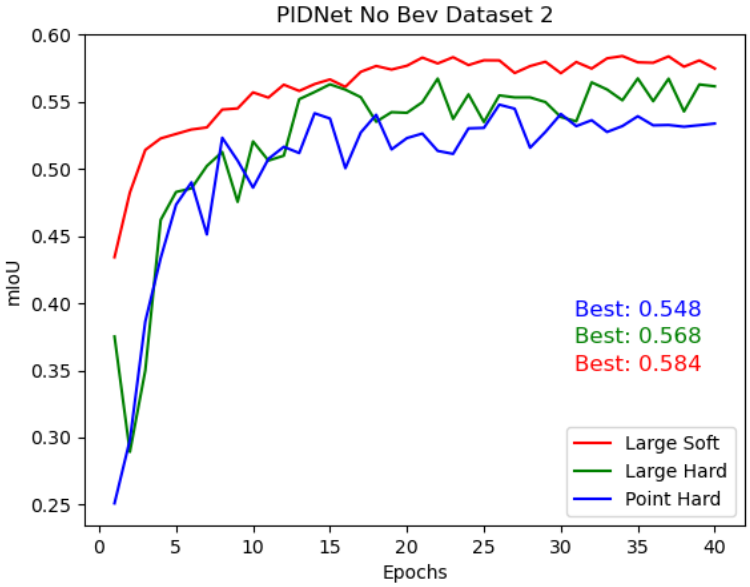
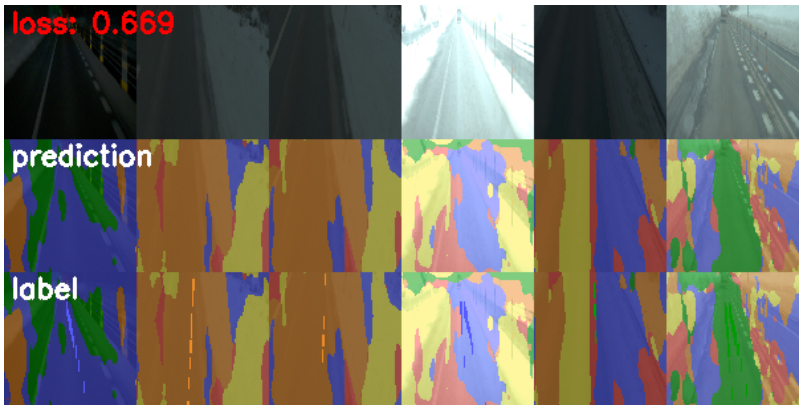
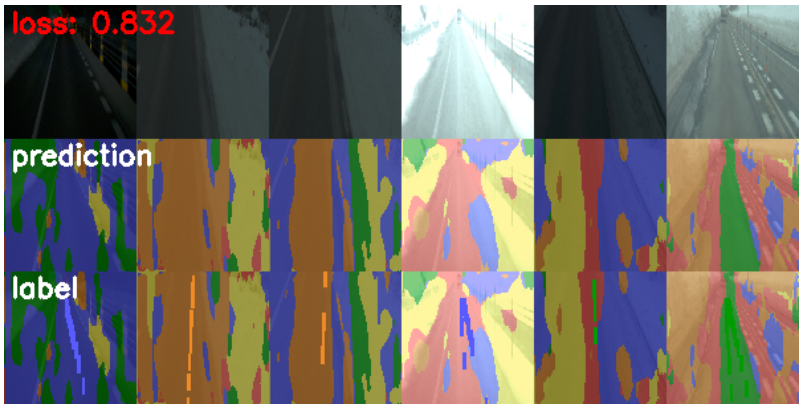


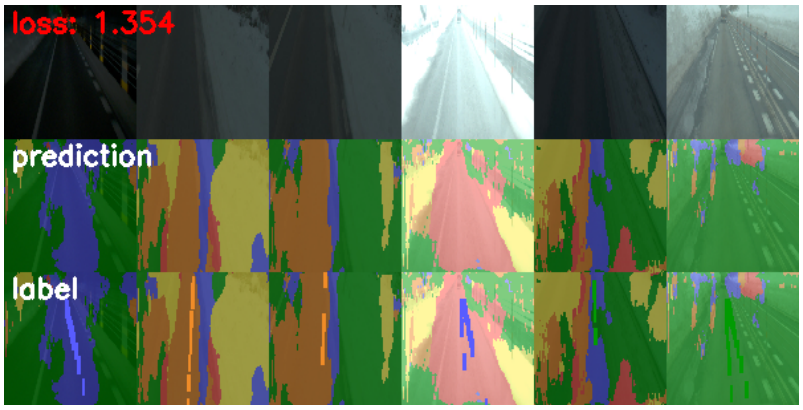
Figure 4.52: The resulting mIoU graph of PIDNet trained on non-bev data from the second dataset. Similarly to the first dataset and SFNet on the same dataset, the most successful labelling scheme is that of the soft large labels.



(a)

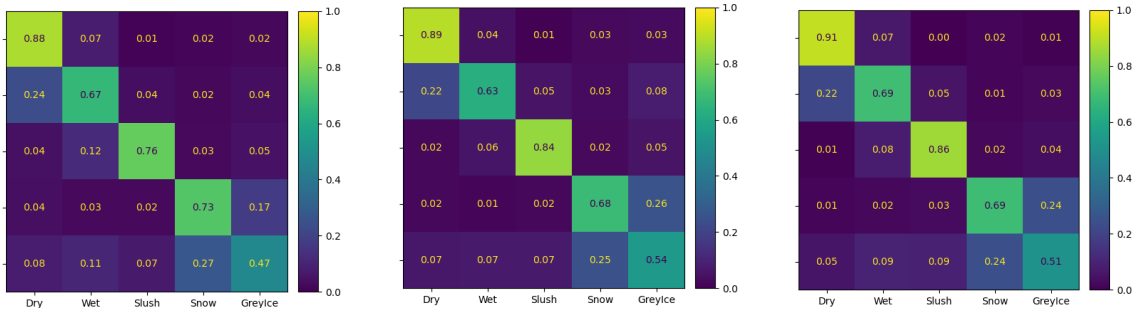


(b)



(c)

Figure 4.53: Resulting input, prediction and labelled images from the PIDNet model validation data in the second dataset. From top to bottom, the labelling techniques are a) hard point labels, b) large hard labels and c) soft large labels. Similarly to the other models trained on the same data, it is not able to correctly classify the bright wet road in image 4. Another interesting observation can be found in the 3rd image for large soft labels, where what seems to be snow on the outskirts of the road is classified as dry.



(a) **(b)** **(c)**

Figure 4.54: Confusion matrices from the PIDNet model with different labelling schemes on non-bev data from the second dataset. From left to right, the matrices belong to hard point labels, large hard labels and soft labels. Much like the other models, the most significant observation is that the model often misclassifies wet conditions as dry and has difficulty distinguishing snow and ice conditions.

4.8.2.4 DeepLab

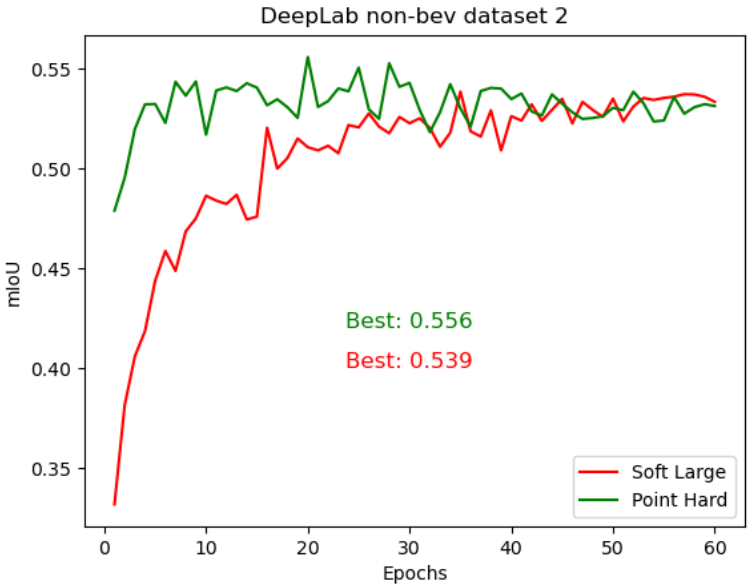
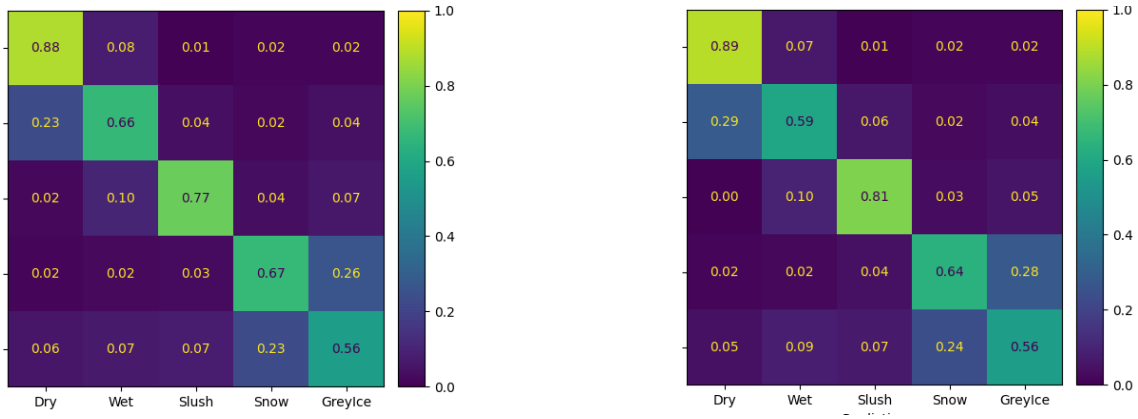


Figure 4.55: mIoU graph of Deeplabv3plus trained on non-bev data from the second dataset.



(a) **(b)**

Figure 4.56: Confusion matrices for deeplabv3plus trained on non-bev data from the first dataset with a) hard point labels and b) soft large labels. The model has the same problems as the other models, where misclassification between wet and dry, and snow and grey ice is common.

4.9 AGMM

This section will present the results achieved through the AGMM labelling scheme. For this case, the labels will always be hard point labels, but the loss functions introduced in section 2.6.1 will be added with the supervision. The method was supposed to act as a bridge between a sparsely annotated dataset and a densely annotated one by having unlabeled pixels also contribute to the supervision. Consequently, the aim of the model was to provide increased performance, especially on the outskirts of the road. The method will be applied on ENet and deeplabv3plus, which is one of the models that the developers of the method [14] applied it on.

4.9.1 AGMM on ENet

In general, the AGMM method harmed performance when applied to the ENet model. The performance in terms of mIoU is simply lower than when ENet was trained on the same data without the AGMM method applied. The cause of this will be discussed further in the discussion section, but one likely reason is that the sparse labels of the provided data were just too different compared to what the method was developed for. In particular, the images seen in Figure 4.60 indicate that the method specifically struggled with classification in locations where the sparse labels rarely occur, which is precisely what AGMM was supposed to improve.

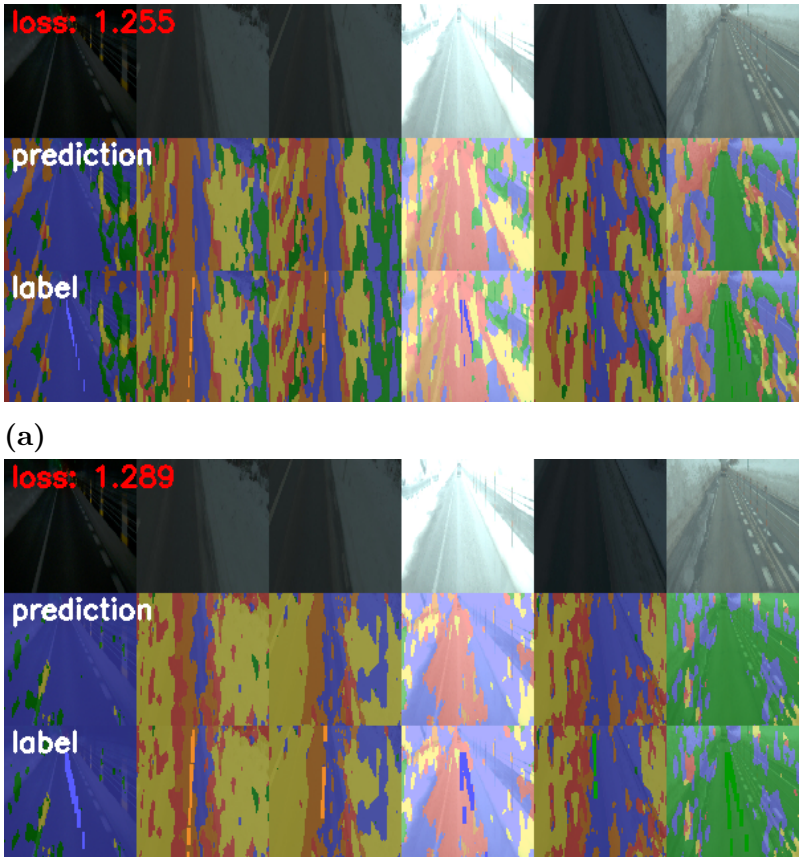
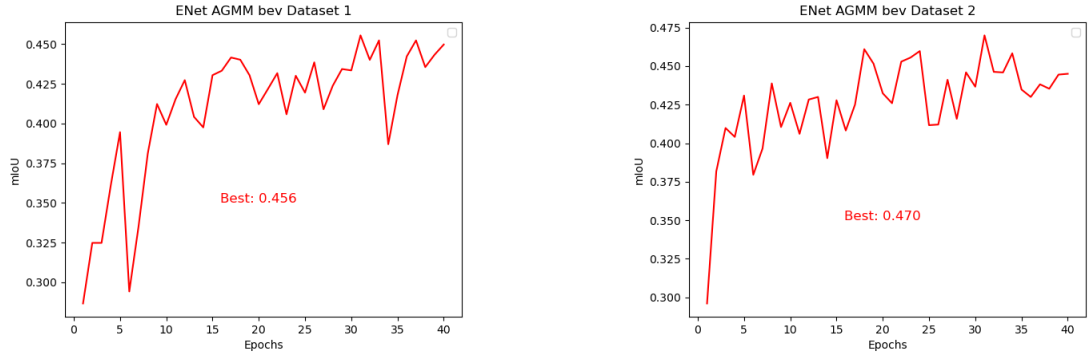


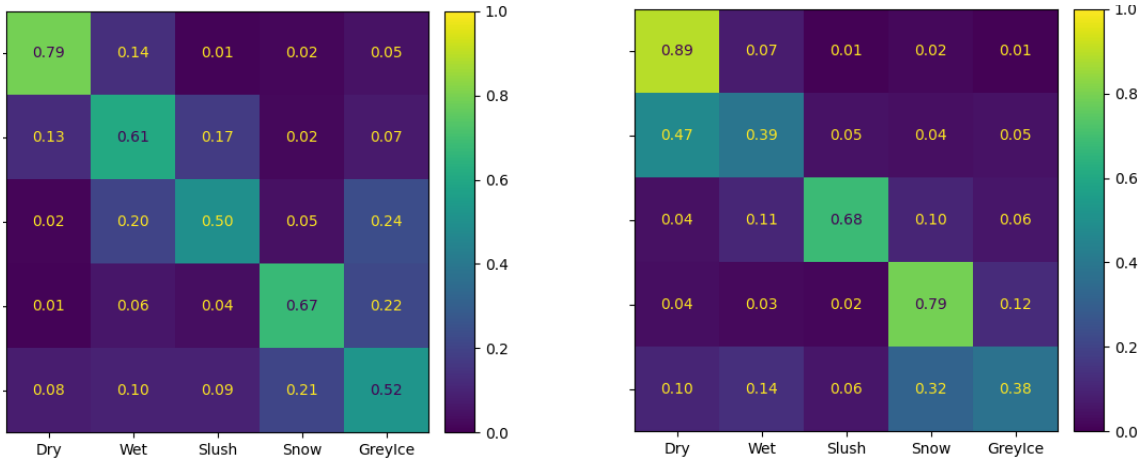
Figure 4.57: Resulting predictions of 6 image sets from the validation data for the deeplabv3plus model trained on non-bev data from the first dataset with a) hard point labels and b) soft large labels. Interestingly, the images from the deeplabv3plus model come with less consistent predictions compared to those of the other models. In particular, for hard point labels, there are a lot of seemingly random blotches of colour predictions, which could indicate overfitting. The deeplabv3plus model is also more complex than the other models, further supporting the overfitting notion.

4.9.1.1 Bev input data



(a) **(b)**

Figure 4.58: The mIoU graphs for ENet trained on the first a) and second b) dataset on bev input data with the AGMM loss functions applied. In comparison with ENet using the Gauss labelling scheme on bev data from dataset 1 and 2, see Figure 4.13 and 4.37 for reference, the performance is worse in terms of mIoU. Due to limited computing power, these were merely trained for 40 epochs, which may have reduced the performance slightly.



(a) **(b)**

Figure 4.59: Confusion matrices for ENet trained on bev data from dataset 1 a) and dataset 2 b) with AGMM applied. The model seems to struggle with the same problems as ENet without AGMM. Especially the second dataset, which was earlier known to cause the models to misclassify wet conditions as dry and grey ice as snow. However, compared to ENet without AGMM applied, found in Figure 4.14 and 4.38, the accuracy of almost all conditions is lowered by a few percentages.

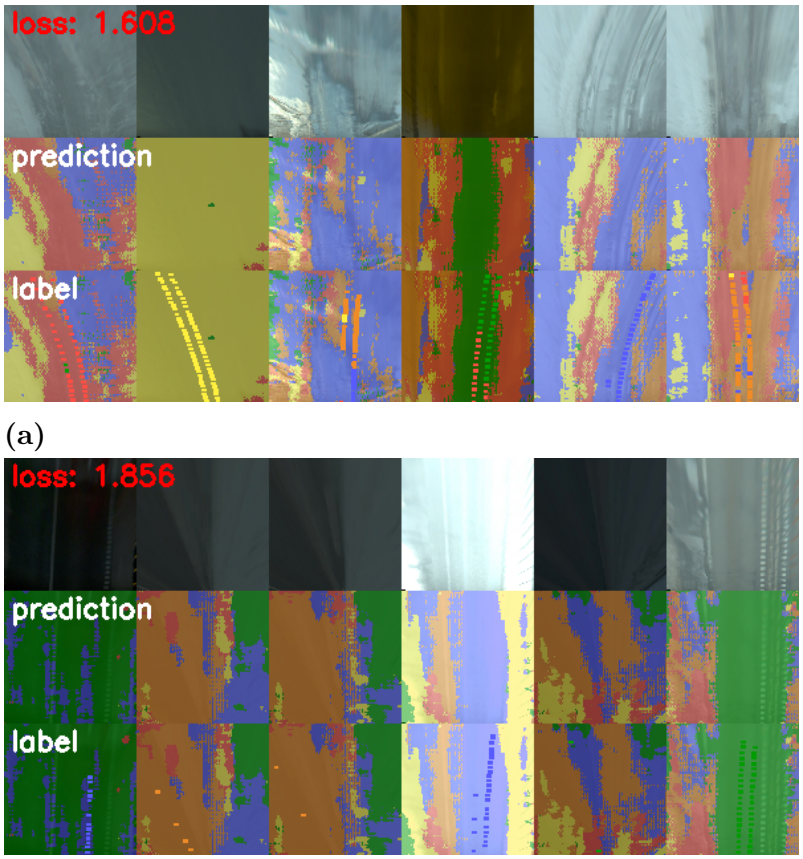
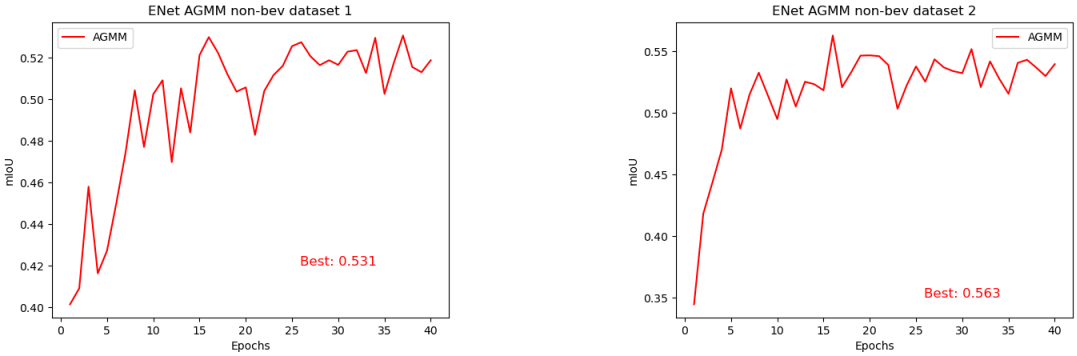


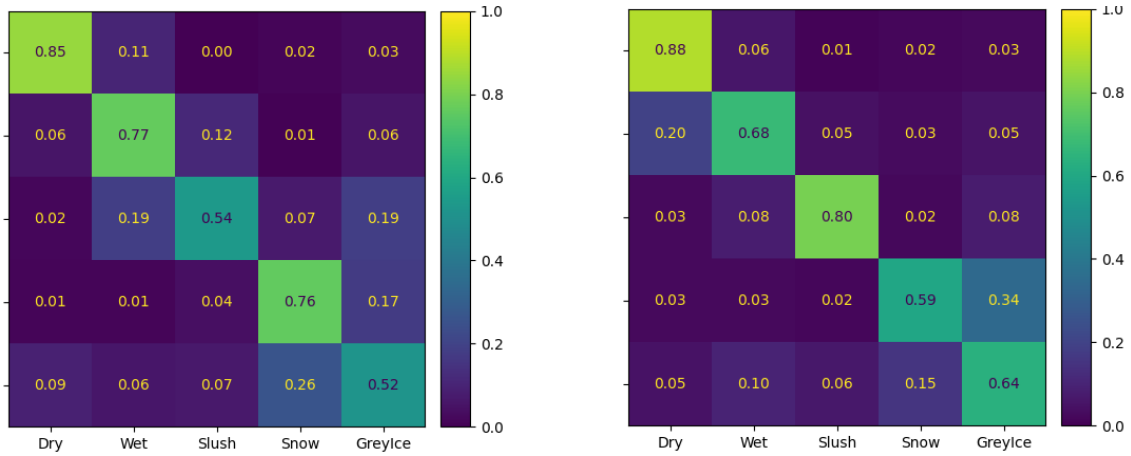
Figure 4.60: Example image sets with the input, predictions and labels from the validation dataset. Both figures use results retrieved from ENet trained with AGMM on bev data, where a) represents the first dataset and b) the second dataset. In both cases, the predictions match reasonably well with the sparse labels, but outside of that, the results are quite disappointing. AGMM is supposed to help the network classify unlabeled pixels. However, it is apparent in the results, especially from image sets 5 and 6 in a) and 2 and 3 in b), that it is not able to do so.

4.9.1.2 Non-bev input data



(a) **(b)**

Figure 4.61: The mIoU graph of ENet trained with AGMM applied on non-bev data from the first dataset a) and second dataset b). Similarly to the results for bev data, the mIoU values are simply lower than those achieved with the large soft labels instead of AGMM. The first dataset exhibits a significantly lower mIoU of 5% while the second dataset is only lowered by 2%.



(a) **(b)**

Figure 4.62: Confusion matrices for ENet trained with AGMM on non-bev data from the first dataset a) and the second dataset b). In comparison to the results from ENet with soft large labels, found in Figure 4.26 and 4.50, the first dataset performs worse for all conditions, while the second performs slightly better for some conditions, but in general the performance is worse.

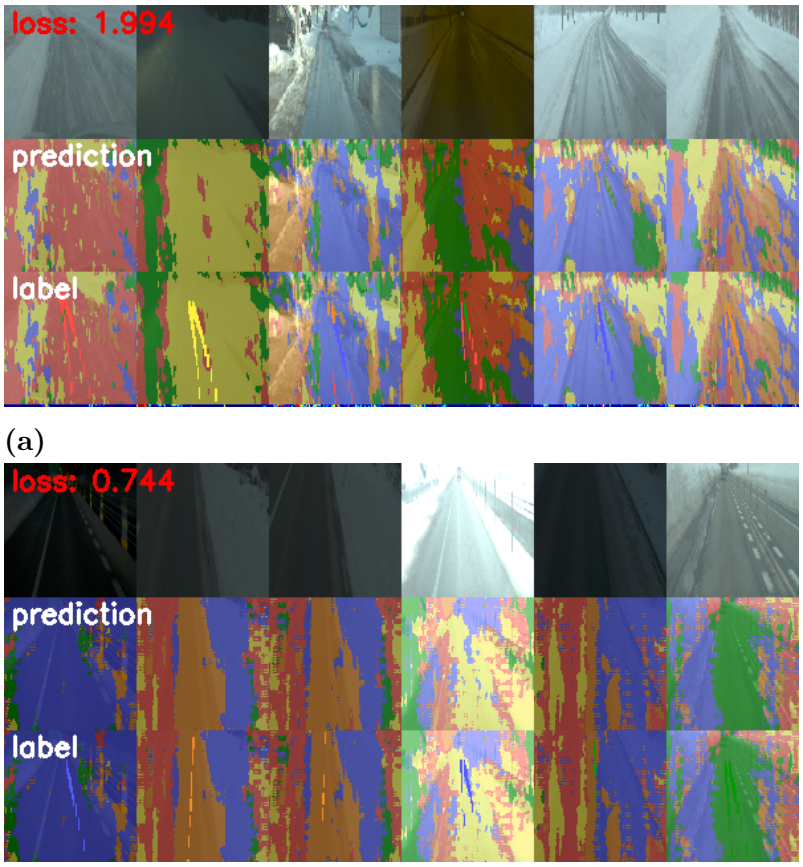
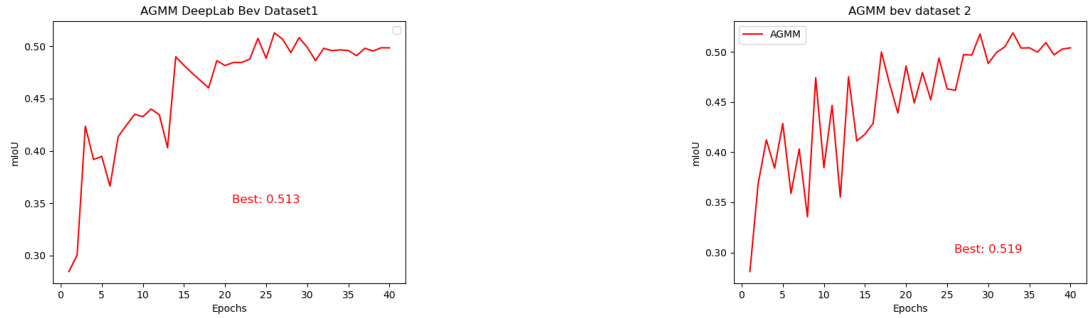


Figure 4.63: Much like the case for ENet trained with AGMM on bev data, the model is able to predict the conditions of the sparse labels reasonably but fails quite severely on the unlabeled pixels, which is precisely what the method was supposed to help with. Compared to ENet trained without AGMM applied, see figure 4.27 and 4.51, the predictions are worse.

4.9.2 AGMM on Deeplabv3plus

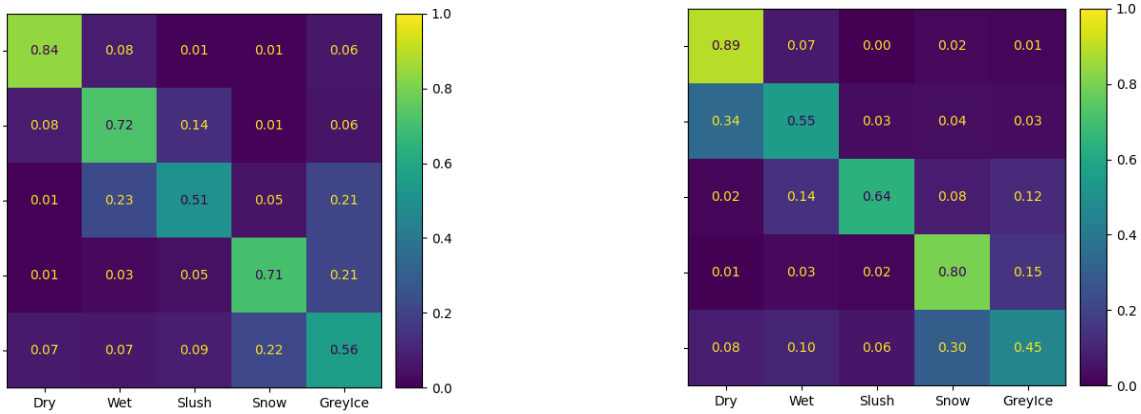
Deeplabv3plus is one of the machine learning models that the authors of AGMM [14] used. Consequently, AGMM should be more suitable for this model, and better results can be expected. This is generally true, but compared to the Deeplab model trained simply on hard point labels, the AGMM performed slightly worse, indicating that the method still harmed the model's performance by adding confusing losses.

4.9.2.1 Bev input data



(a) **(b)**

Figure 4.64: The mIoU graphs for Deeplabv3plus model trained on bev data with added loss from AGMM. a) represents results on the first dataset, while b) represents results on the second dataset. Compared to Deeplab trained with simple hard point labels, found in Figure 4.19 and 4.43, the AGMM labelling technique performs slightly worse for the first dataset and has an equal performance for the second one.



(a) **(b)**

Figure 4.65: Confusion matrices for Deeplabv3plus trained with AGMM on bev data. a) presents the matrix from the first dataset and b) from the second dataset. Compared to Deeplab trained with simple hard point labels, see Figure 4.20 and 4.44, the most notable difference is the accuracy of the slush condition in dataset 1. The other conditions are roughly the same.

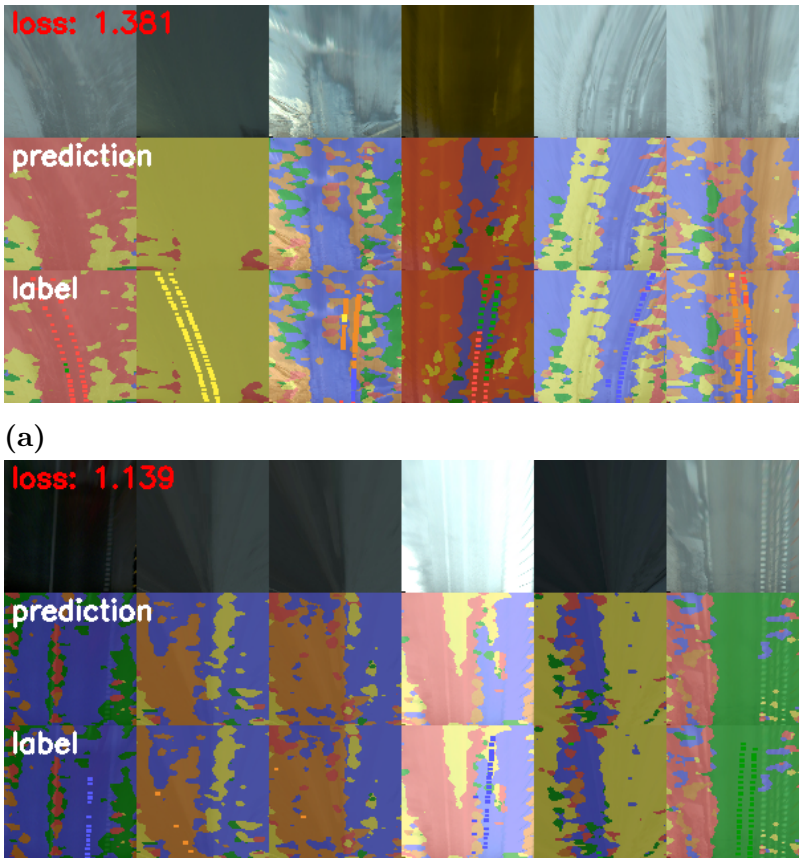
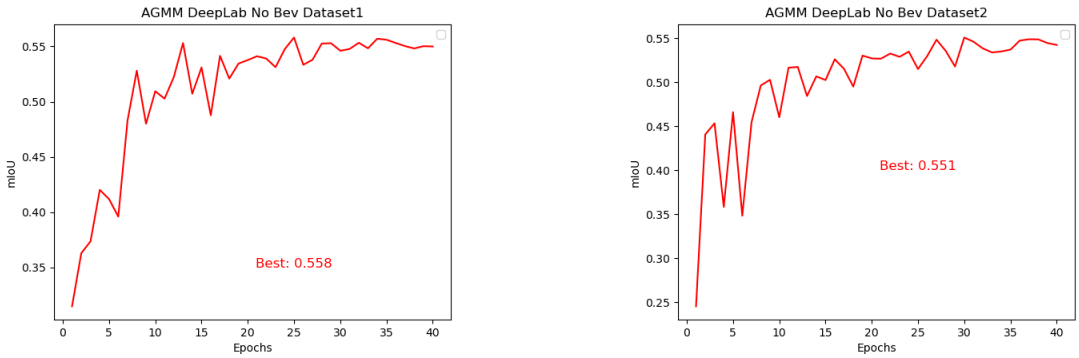


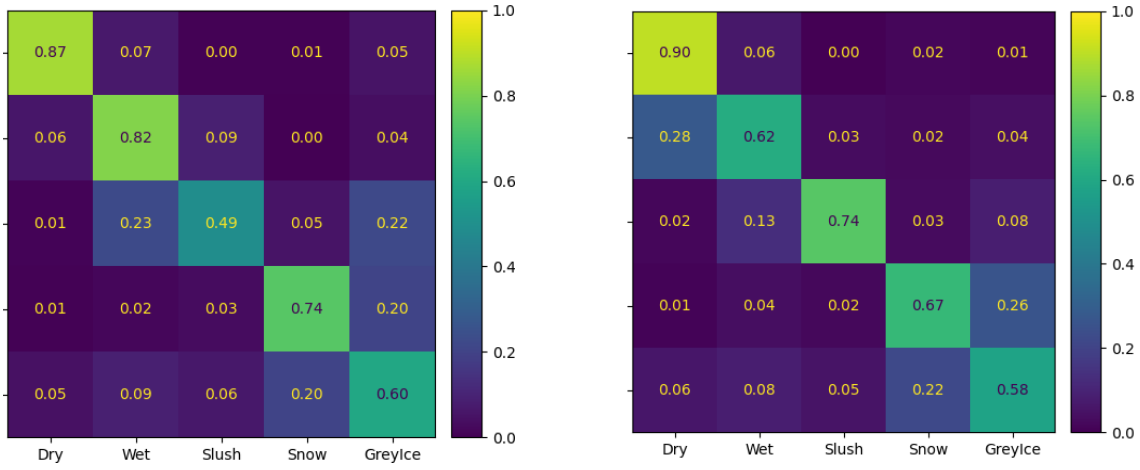
Figure 4.66: Prediction examples from the Deeplabv3plus model trained with AGMM for bev data on the first dataset a) and second dataset b). The most notable observation is that, even though AGMM was implemented to help with this, the model is not able to classify the conditions in places where the sparse labels typically do not occur.

4.9.2.2 Non-bev input data



(a) **(b)**

Figure 4.67: The mIoU graphs for the Deeplabv3plus model trained on non-bev data with added loss from AGMM. a) represents results on the first dataset, while b) represents results on the second dataset. The resulting mIoU values are roughly the same as those acquired from the DeepLabv3plus model trained with hard point labels.



(a) **(b)**

Figure 4.68: Confusion matrices from the Deeplabv3plus model trained on non-bev data with added loss from AGMM. a) represents results on the first dataset, while b) represents results on the second dataset. Similarly to the mIoU value, the matrices are roughly equivalent to those acquired from Deeplabv3plus trained on hard point labels.

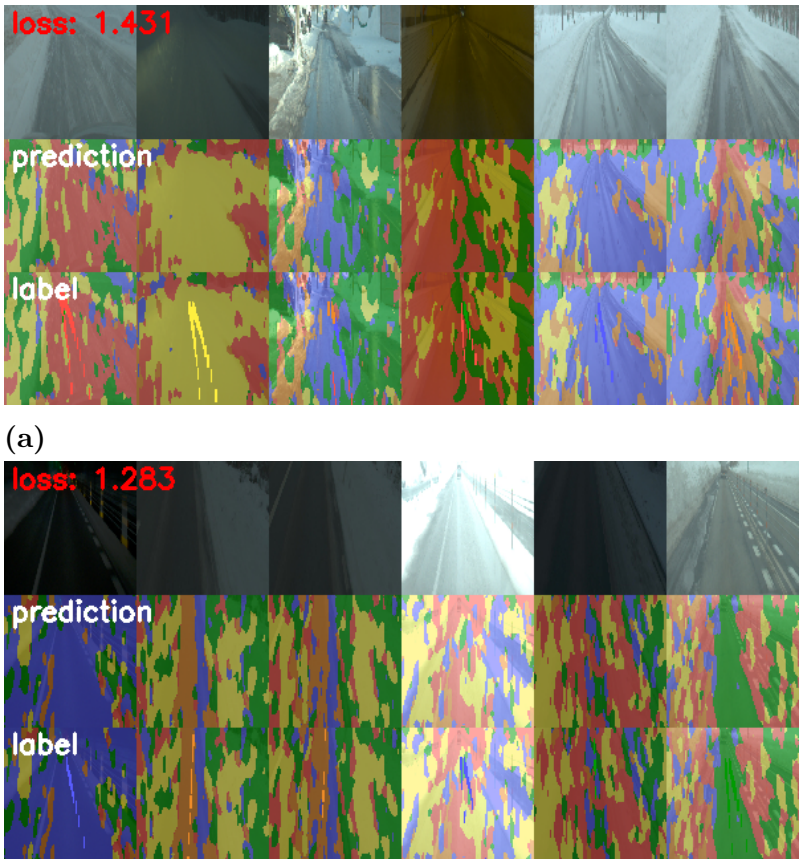


Figure 4.69: Example image sets from AGMM trained on non-bev data with added loss from AGMM. The most noticeable observation is the seemingly random blotches of classifications that appear on the outskirts of the road. However, the model struggled with these models when trained with hard point labels as well and could, therefore, be a limitation in the model. Nonetheless, the AGMM fails to improve the performance by bridging a gap between densely and sparsely annotated data.

4.10 Result conclusion

This section presents the results more concisely, with tables presenting key parameters such as accuracy and mIoU. Note that the results from soft point labels used in Bev data have been omitted due to their disappointing results.

Table 4.1: The table contains all the results performed on bev data from the first dataset. The best-performing metric of each category is marked by bold text. All metrics, except for the mIoU, are based on the mean batch accuracy of the validation data. Based on mIoU, the Deeplab model trained with hard point labels performed best. However, SFNet is barely 0.1 % away and exhibits an arguably better slip/non-slip accuracy.

Performance results for bev data from the first dataset									
Model	Labeling	mIoU	Dry	Wet	Slush	Snow	Grey ice	Non-Slip	Slip
ENet	Gauss	0.462	0.827	0.479	0.581	0.617	0.470	0.803	0.871
ENet	Hard point	0.479	0.812	0.579	0.493	0.694	0.534	0.817	0.873
ENet	AGMM	0.456	0.782	0.585	0.480	0.643	0.515	0.83	0.855
SFNet	Hard point	0.533	**0.861**	**0.705**	0.548	0.698	0.511	**0.873**	0.883
SFNet	Gauss	0.518	0.795	0.620	0.645	**0.718**	0.497	0.817	**0.919**
PIDNet	Hard point	0.528	0.835	0.676	0.577	0.683	0.546	0.853	0.897
PIDNet	Gauss	0.498	0.798	0.565	**0.665**	0.684	0.518	0.801	0.840
Deeplab	AGMM	0.5128	0.787	0.688	0.473	0.646	0.537	0.851	0.872
Deeplab	Hard point	**0.534**	0.845	0.669	0.602	0.652	**0.561**	0.850	0.888

Table 4.2: The table contains all the results performed on bev data from the second dataset. The best-performing metric of each category is marked by bold text. All metrics, except for the mIoU, are based on the mean batch accuracy of the validation data. Based on mIoU, the SFNet trained with Gauss labels was the best-performing model and also achieved the best value in two other metrics, whereas any other model only managed to perform best in a maximum of one metric.

Performance results for bev data from the second dataset									
Model	Labeling	mIoU	Dry	Wet	Slush	Snow	Grey ice	Non-slip	Slip
ENet	Gauss	0.494	0.878	0.384	0.777	0.563	**0.581**	0.903	0.895
ENet	Hard point	0.478	**0.892**	0.368	0.730	0.640	0.527	0.927	0.869
ENet	AGMM	0.470	0.878	0.398	0.759	0.690	0.444	0.920	0.868
SFNet	Hard point	0.509	0.867	0.526	0.740	0.700	0.482	0.933	0.899
SFNet	Gauss	**0.565**	0.891	0.581	**0.812**	0.642	0.562	0.948	**0.912**
PIDNet	Hard point	0.500	0.870	0.526	0.721	**0.704**	0.467	0.922	0.895
PIDNet	Gauss	0.554	0.871	0.602	0.804	0.640	0.505	0.940	0.890
Deeplab	AGMM	0.519	0.847	0.581	0.694	0.611	0.545	0.927	0.896
Deeplab	Hard point	0.520	0.859	**0.640**	0.692	0.679	0.459	**0.951**	0.872

Table 4.3: The table contains all the results performed on non-bev data from the second dataset. The best-performing metric of each category is marked by bold text. All metrics, except for the mIoU, are based on the mean batch accuracy of the validation data. Based on mIoU, the ENet trained with soft large labels was the best-performing model and achieved the best value for wet conditions. The model did not quite achieve any best performance for non-slip or slip accuracies, but achieved among the highest for both of them.

Performance results for non-bev data from the first dataset									
Model	Labeling	mIoU	Dry	Wet	Slush	Snow	Grey ice	Non-slip	Slip
ENet	Large soft	**0.579**	0.867	**0.756**	0.583	0.730	0.604	0.885	0.911
ENet	AGMM	0.531	0.838	0.725	0.517	0.740	0.516	0.886	0.884
SFNet	Hard point	0.566	0.862	0.731	0.511	0.723	**0.615**	0.871	0.916
SFNet	Large hard	0.5675	0.865	0.731	0.540	**0.758**	0.569	**0.887**	0.902
SFNet	Large soft	0.566	0.830	0.721	**0.710**	0.740	0.529	0.867	0.922
PIDNet	Hard point	0.544	0.848	0.701	0.557	0.731	0.607	0.854	0.910
PIDNet	Large hard	0.553	0.811	**0.756**	0.527	0.730	0.591	0.874	0.906
PIDNet	Large soft	0.568	0.860	0.697	0.666	0.735	0.551	0.864	**0.925**
Deeplab	AGMM	0.557	0.855	0.715	0.501	0.746	0.505	0.882	0.891
Deeplab	Hard point	0.553	**0.897**	0.681	0.561	0.734	0.566	0.871	0.917
Deeplab	Large soft	0.545	0.835	0.685	0.677	0.722	0.531	0.857	0.909

Table 4.4: The table contains all the results performed on non-bev data from the second dataset. The best-performing metric of each category is marked by bold text. All metrics, except for the mIoU, are based on the mean batch accuracy of the validation data. Based on mIoU, the SFNet trained with soft large labels was the best-performing model, even though it did not achieve the highest value of any accuracy metric. However, it does achieve a reasonable accuracy through all conditions, which is something the models that did achieve the highest accuracy in a metric did not achieve. It also has the most well-balanced trade-off between slip and non-slip conditions.

Performance results for non-bev data from the second dataset									
Model	Labeling	mIoU	Dry	Wet	Slush	Snow	Grey ice	Non-slip	Slip
ENet	Large soft	0.582	0.858	0.689	0.810	0.608	**0.608**	0.923	0.900
ENet	AGMM	0.531	0.838	0.725	0.517	**0.740**	0.516	0.886	0.884
SFNet	Hard point	0.544	0.877	0.5988	0.757	0.614	0.558	0.934	0.890
SFNet	Large hard	0.559	0.873	0.545	0.830	0.558	0.677	0.912	**0.925**
SFNet	Large soft	**0.588**	0.867	0.620	0.842	0.668	0.568	0.929	0.924
PIDNet	Hard point	0.548	0.866	0.627	0.777	0.673	0.499	0.922	0.882
PIDNet	Large hard	0.568	0.866	0.589	0.837	0.634	0.550	0.882	0.922
PIDNet	Large soft	0.584	**0.890**	0.682	**0.848**	0.638	0.524	**0.939**	0.915
Deeplab	AGMM	0.564	0.797	**0.743**	0.618	0.632	0.580	0.866	0.915
Deeplab	Hard point	0.556	0.868	0.637	0.782	0.594	0.591	0.923	0.910
Deeplab	Large soft	0.539	0.877	0.568	0.800	0.593	0.569	0.915	0.912

5

Discussion

This chapter delves into the interpretation and implications of the findings presented in the previous section. The primary aim is to try to understand the results and the underlying factors that caused them. Additionally, the chapter will discuss difficulties encountered during the project and other sources of errors that may have affected the outcome. It will also explore significant limiting factors and suggest directions for potential future work.

5.1 Difficulties

Most difficulties that occurred in the project can be traced to the unique characteristics of the provided data and the desired output of semantic segmentation. As previously mentioned, most research within semantic segmentation is based upon densely annotated datasets and sparsely annotated data, which, while promising, has not yet gained a lot of foothold within the subject. The sparsely annotated data and the soft labels define the uniqueness of the provided data. While there is existing research within machine learning based on soft labels and some methods for supervision with sparsely annotated data, the combination of the two is rare. Therefore, an overarching problem has been finding solutions and compromises, such as AGMM and converting soft labels into hard labels.

The machine learning models utilised in the project were originally designed for dense, hard-label annotations, providing a significant amount of information the model can train with. In contrast, the sparse annotations significantly decrease the amount of information, and the soft labels contribute to a reduced certainty while training the model. Furthermore, the provided dataset differs from other sparsely annotated datasets used for research in sparsely annotated semantic segmentation. For instance, the sparsely annotated datasets used in AGMM and TEL [14, 7] were generated from densely annotated datasets. Consequently, every class present in the input image has at least one corresponding label in the ground truth. In such cases, one can extract information from each class in the input and utilise it for supervision. For instance, the AGMM calculates mean features for each labelled class in a batch and uses it to create Gaussian mixtures that are later used for supervision from unlabelled pixels. The method also has a calculated loss that actively punishes the model for predicting classes that are not represented by the labelled pixels. However, the dataset that is provided in this report does not guarantee that each class present in the input image are represented by corresponding labelled pixels.

Therefore, the efficiency of previously researched methods for sparsely annotated semantic segmentation is limited. This will be discussed further in section 5.6.

5.2 Class weights

The provided data also included less worrying difficulties, such as a less-than-optimal condition diversity. To address this problem, the project opted to introduce class weights. Class weights punish the model more when failing to predict the rarer conditions, which in this case was the slush condition. This causes the model to develop a certain bias while training, making it more inclined to learn more from pixels labelled with the rare class conditions and make the model successfully achieve better results for those conditions. However, theoretically, this should come at a cost in performance for the other more common conditions. Consequently, class weights may harm the overall value of mIoU, especially since the model does not weigh in class weights while evaluating, even though the validation data share the same distribution of conditions as the training data. However, as seen by the results in Figure 4.2 this does not appear to be the case. The model's ability to predict slush conditions has improved significantly with applied class weights. Except for a slightly deteriorating performance in snow conditions, the other conditions are actually somewhat increased, which is a positive surprise.

Additionally, it is possible that class weights could cause some obscure biases in the resulting semantic segmentation that are not apparent from the confusion matrices as they are only based on the model's performance on labelled pixels. However, as observed in the example images from models trained on different data and models, there may be some biases towards slush but not large enough to believe it would be better without applying class weights. Furthermore, even if the class weights did harm the overall performance of other conditions, would it not be preferable to have models with more balanced predictions? That is, it would be preferable to have a model that predicts all conditions with a 50% accuracy than 2-3 conditions with an accuracy of 100% and the other conditions with an accuracy of 0%.

5.3 Difference between the two datasets

Initially, the primary focus was on the first dataset, and almost all tests were performed on that dataset. However, suspicions started arising that the training and validation datasets were too similar and would not provide a reliable evaluation of the pipeline's ability to generalise to new data. These suspicions were primarily rooted in the fact that applying data augmentation while parsing the data in a non-bev manner would only worsen the pipeline's performance. Data augmentations slightly alter the training data in reasonable and realistic ways to create new scenarios that the model can learn, making it more robust when encountering new data. It can be argued that some of the data augmentations used in this project are unsuitable for non-bev data parsing. Still, this project tested every single augmentation individually, and each of them worsened the performance. As such, it is possible that even slight augmentations in the training dataset altered the data

enough to create scenarios that do not appear in the evaluation dataset, indicating that the validation dataset was too similar to the training data and did not subject the model to new data. This would severely harm the model's ability to generalise and result in inflated evaluation numbers.

Another cause for the suspicions was that the model predicted one condition over either the entire image of some evaluation frames or simply split the frame into two different conditions. For instance, there were evaluation frames with two tracks where one track was labelled as snow and the other as grey ice. In this case, the model would predict the left side of the frame as snow containing the snowy track and the right side of the frame with the icy track as grey ice, meaning it did not make any effort of trying to predict the conditions over areas where the sparse labels rarely occur. A theory of what caused this behaviour is that the model had encountered a very similar frame while training, and since the sparse labels are limited to the tracks, the pipeline was not punished severely enough to make an effort in trying to correctly predict the pixels outside of the track.

The second dataset resulted from the suspicions that the first dataset's training and validation data were too similar. This dataset consists of recordings from multiple different collaboration companies in different countries. In contrast, the data in the first dataset were all recorded by the same company in a single country. As a result, the data was also different in other ways. For instance, smaller factors such as the recording angle and the recording vehicle.

Additionally, the second dataset split the training and validation data more harshly. The split function for the first dataset was based on session, ensuring all frames in a single session would go to either the training or evaluation frames. However, the split function for the second dataset was instead based on days. All sessions recorded on the same day and by the same company are either attributed to the training or validation dataset. The main reasoning behind this originates from how the sessions are recorded. Excluding some exceptions, each session is 10 minutes long, and, commonly, sessions are recorded consecutively, say six sessions spread out on a 60-minute drive. As such, there are often sessions recorded on the same road, in the same weather and therefore in the same conditions, except they are some 10 minutes apart.

It is also not uncommon that the vehicle is driving solely for recording. In such cases, the vehicle often drives for, say, 20 minutes, then turns and drives back the same way. This recording process contributes to more sessions containing essentially the same information. Consequently, the splitting based on days instead of sessions makes even more sense since the goal was to cause differences between the training and validation data. The second dataset was also much larger than the first, introducing even more data variety, which should increase the model's ability to generalise to new data.

However, the data quality also varies depending on which collaboration company recorded the data. The collaboration company that recorded the data used in the first dataset was deemed to have recorded the highest quality data, which is apparent when observing the resulting image sets from validation data in section 4. The

other data added to the second dataset were of less quality and contained more of the flaws presented in section 3.1.3 including, but not limited to, imprecise motion estimation. Such flaws can cause significant harm while training, resulting in incorrect gradients that can cause contradictions. Additionally, a lot of data from the other collaboration companies have little variety in the individual frames.

It is pretty evident from observing the validation example images that the data is, for lack of another word, less interesting in the second dataset. The second dataset contained much more homogeneous data, where all sparse labels in a single image were assigned to the same condition, which could be a factor as to why the methods enlarging the labels were so efficient on the second dataset. Additionally, when comparing validation images from both datasets, there is a noticeable difference in the amount of sparse labels in the data. The data from the first dataset typically has 20-50 labels, while some data in the second dataset have less than ten labels. The importance of information has been reiterated frequently in this project, and data with such a small number of labels could lessen the model's ability to learn. Consequently, while the second dataset contained almost twice the number of frames as the first dataset, the total number of labels may not have differed quite as much. After all, it is the number of labels that determine how much supervision can be provided to the model and, subsequently, how much the model can learn.

Furthermore, the main indicator for the first dataset not separating the training and validation data enough was that data augmentations were not improving the performance for non-bev data. However, with the exception of colour and quality augmentations, the same applied to the second dataset. This indicates that either both datasets suffer from poor variety between training and validation data or the data augmentations implemented were not well-suited for the non-bev case. It is also possible that the data augmentations were too aggressive and should have been tuned down even further.

In conclusion, the first dataset contains higher-quality data, which makes it easier for the pipeline to learn correctly and more efficiently. However, it is plausible that the dataset suffered from too little variety between the training and validation data. If that is the case, it would harm the pipeline's ability to generalise to new data but would, at the same time, inflate the results obtained in the project.

In contrast, the second dataset contained more data variety as it was recorded from different sources in different countries. The second dataset also had an adjusted splitting function that caused larger differences between the training and validation data, which should result in a model that has a higher performance when applied to new data. However, the main indication of training and validation data being too similar in the first dataset also occurred in the second dataset. Additionally, the performances between the two datasets are quite similar, which further indicates that either both datasets suffered from data that was too similar in the training and validation sets or neither of them did. In such a case, the similar performances of both datasets suggest that, even though the second dataset contains less quality data, the sheer amount of data makes up for it.

5.4 Models

According to the results from section 4, it seems that SFNet, PIDNet and Deeplabv3plus were able to outperform ENet on bev input data, where SFNet and Deeplabv3plus are the overall best-performing models. However, on non-bev data, ENet still seems to have come out on top by a slight margin on dataset 1 while being slightly overperformed by SFNet and PIDNet on the second dataset. The most reasonable argument for this is that the SFNet and PIDNet models are perhaps a bit better at generalising, while ENet is better at memorising data with less variety. As discussed before, the first dataset contains less variety than the second, and the second dataset should also have a larger discrepancy between the training and validation data. The fact that PIDNet and SFNet performed better than ENet on the second, but not the first, dataset could, therefore, indicate a better ability to generalise. Another factor supporting this is that these networks also performed better on bev data, which is augmented quite aggressively, introducing more variety.

However, the difference between some of the results, especially for non-bev data, is relatively small, and the models were trained stochastically, meaning that the results may fluctuate slightly between training runs. The results were typically based on a single or a few runs and, as such, did not properly address the fluctuation part. It is possible that if the same training runs were run for, say, ten runs and the average was taken, the outcomes would be slightly different. Therefore, the fact that all models seem to converge towards a similar mIoU, especially for non-bev data, could indicate that the data is the bottleneck here.

As discussed in section 3.1.3, the data came with a few flaws, which, combined with the limited information provided by the sparse labels, could be a cause of why the choice of model did not affect the results in any significant way. In contrast, for the Cityscapes dataset [5], which has fewer flaws and provides substantially more information, both SFNet and PIDNet outperform the ENet model by a substantial margin, 80.4% mIoU for SFNet-R18, 78.6% for PIDNet-S and merely 58.3% for ENet [16]. It has previously been discussed that PIDNet and SFNet are specifically built for the Cityscapes and similar datasets, which one can argue would explain why those models perform so much better on the Cityscapes dataset, but so was ENet originally. Another argument would be that ENet was developed way earlier (2016) than both PIDnet (2022) and SFNet (2020). During this time, there have been several new discoveries that could be used to engineer functions that specifically improve the models on the Cityscapes dataset. While this is plausible, it is also unlikely that none of those functions would be beneficial for the provided dataset.

Another potential issue could be that the SFNet, PIDNet and Deeplabv3plus models are too complex for the problem at hand, and it is shown in figure 4.7, that a more complex version of PIDNet does not induce an increase in performance. This can be attributed to other factors in the data. In particular, the most efficient way to differentiate road conditions is the texture, edges, and colour. In contrast, the dataset Cityscapes [5] has classes such as humans and signs. Object-oriented classes such as these are more easily differentiated through shapes or object parts such as hands. Shapes and object parts are examples of high-level features, while texture,

edges, and colour are low-level features.

Consequently, the model does not benefit much from extracting high-level features since the classifications of conditions in the provided dataset are more easily achieved through low-level features. Model complexity is directly correlated to the depth of a model, and, as discussed in the theory section, the layers in a model are built up hierarchically. Early layers of a CNN capture the lower-level features, while deeper layers capture high-level features. As such, an excessively deep network will indeed capture higher-level features, but these are not beneficial to the task at hand. Therefore, a shallow network that is more focused on capturing lower-level features will still be able to achieve good performance on the provided data.

Considering that the choice of model has little effect on performance, it is clear that the focus should instead be brought to adjust the data. The simplest way to do so would be data augmentations, which was shallowly explored in the project and is discussed further in 5.9. Another approach would be to minimise the flaws found in the datasets, such as faulty motion estimations, see section 3.1.3 for reference, which could be done by either manually filtering or developing some function that better filters the data in the dataset. Considering that there are 16000 frames in the training data for the first dataset and 32000 in the second dataset, this is quite an ungrateful task to do manually. On the other hand, since an adequately filtered dataset would not include contradicting scenarios caused by faulty motion estimation, which in a counteractive manner affects the learning process, such a dataset would likely require fewer frames to achieve similar performance, as the model would learn more from each individual frame. Consequently, while the filtering may come at a cost, this cost may be mitigated by saved computing time on a smaller but higher-quality dataset.

The last approach that comes to mind is to acknowledge the differences between the provided data and the data used in prominent literature within the subject of semantic segmentation, such as the Cityscapes dataset [5], which is the amount of information provided in the labelled data. The model would get more supervision by pushing towards a more dense dataset with a larger fraction of labelled pixels, making it learn better and faster. Furthermore, the data would be more akin to data used in other research, making it easier to adopt the models and methods they developed. Different approaches to achieving this are discussed more thoroughly in section 5.5.

5.5 Labeling techniques

The approach taken in this project to address the problem of sparse labels was to adjust the labels according to assumptions, i.e. large labels or Gauss labels, and implement methods that aim to provide supervision even on unlabeled pixels, in this case, AGMM [14] and TEL [7]. As the results indicate, the large labels were a success, and the Gauss labels worked quite well for the second dataset but not as well in the first dataset. This indicates that the assumptions made, that is, the pixels around the sparse labels have similar features to the labelled pixel, hold for

the provided data. The larger labels provide more information and supervision for the model, making it easier for them to learn features of specific conditions as long as the assumptions hold. However, the large labels essentially provide uncertain supervision that is masked as if it were certain, reducing the average quality of supervision.

There is a threshold where a decrease in average data quality outweighs the benefits of having more data for supervision. This threshold is directly dependent on the characteristics of the provided data. While increasing the size of the large labels may be beneficial for homogeneous data, that is, data where all labels belong to the same condition, it can have a detrimental effect for inhomogeneous data. In inhomogeneous data, where the assumptions about label consistency fall apart, large labels can instead create contradictions. For instance, pixels with snowy features might be incorrectly labelled as icy conditions, causing confusion and reduced model performance.

Another observation is the significantly worse results obtained when using soft point labels instead of hard point labels for bev data, where the model barely learns anything with the soft labels. Interestingly, when using large labels, the soft large labels generally outperform the hard ones, albeit by a small margin. A possible cause is, once again, the amount of information provided by the sparse labels used for supervision. Point labels correspond to the least amount of information possible, while large labels provide significantly more, though less reliable, information. The models might struggle to adapt to the uncertainty introduced by soft labels when combined with the limited amount of information from singular pixel labels. However, with larger labels, the increased amount of information helps the model to adapt better to the uncertainties provided by the soft labels.

As seen in the results, the sparsely annotated segmentation methods did not provide any significant improvement, which is likely due to assumptions taken by the authors of those methods and is discussed further in section 5.6. However, sparsely annotated semantic segmentation is a relatively new concept and has previously mainly been seen in a medical context [41, 42]. Should the subject get a bit more attention, perhaps more literature will be published that develops more universally applicable methods.

An obvious choice to improve semantic segmentation performance on road surface conditions would be to alter the annotated data. The most beneficial approach, and also the bane of sparsely annotated semantic segmentation, is to provide more information by increasing the amount of annotated pixels in the data. This could be done by adding more laser detectors to the recording vehicle, which would also likely cause the sparse labels to be more spread out and, as such, address the problem of sparse labels mostly occurring in the centre of the road. Another approach would be to imitate the datasets used in semantic segmentation literature, such as the Cityscapes dataset [5], which would entail converting the sparsely annotated data to densely annotated. This is probably best done by manual annotations, which require a lot of tedious work.

Additionally, manual annotation is limited by a human's ability to determine the

right road conditions, which evidently is not easy. However, the best approach would probably be to combine more laser detectors and manual annotation, where the laser annotations are used as guidelines while manually annotating the data. Consequently, human error would be reduced while still providing a densely annotated dataset with the maximum amount of information possible.

5.6 Sparsely annotated semantic segmentation methods

Overall, the SASS methods AGMM and TEL were kind of a disappointment as they did not improve the performance at all, quite the contrary. This lack of performance is likely due to the difference between the datasets these methods were developed for and the dataset used in this project. Firstly, their datasets assume that all classes in the image, or at least in a batch of images, have corresponding labels. That is, if there are five different classes in the ground truth, there should be at least one label corresponding to each class. This is not the case with the dataset used in this project. A road where the middle parts, and therefore the labelled parts, are either dry or wet while the outskirts are snow is not at all unheard of. In such a case, neither of these methods would provide any additional supervision for the conditions on the outskirts, as that additional supervision is limited to labelled classes.

Secondly, the datasets these methods are developed for also do not have any fundamental limitations for where the sparse labels can appear. In particular, the datasets are generated by decreasing the information in a densely annotated dataset until it consists of only a few labelled pixels. There are no conditions for where these labels will appear, unlike with the provided dataset used for this project, where a large majority of the sparse labels are simply in two straight rows in front of the recording vehicle. As such, they do not suffer as much from a discrepancy between features of the same condition in unlabeled pixels and labelled pixels due to factors such as viewing angle and reflection. This discrepancy and how it affects the performances in the project is further discussed in section 5.7.

The GMM predictions that are a focal point of AGMM are computed batch-wise. Consequently, a small batch size may not include corresponding labels for all conditions that appear in the input images, or there may be cases where there are too few pixels of a certain condition to accurately extract a mean feature value for creating the Gaussian mixtures for that condition. Either way, a larger batch size would allow more information to be used while computing the Gaussian mixtures, leading to more accurate Gaussian mixtures. As such, the unsatisfactory performance of AGMM applied to the deeplabv3plus model could be due to the reduced batch size of 8 necessary to fit in the GPU memory. However, this does not entirely explain why AGMM applied on ENet lacked performance, even though it did not have a reduced batch size unless a batch size of 16 is also too small.

It is probable that if the batch size was increased, the performance of AGMM would also be increased, as the mean features of each condition, which subsequently determines the conditions Gaussian mixture, are based on more labelled pixels,

resulting in a more accurate mean. Alternatively, the mean features could instead be computed in an epoch manner, such that the mean features are calculated on the average of all labelled pixels in the dataset. However, several pitfalls are introduced through such a procedure, mainly concerning the update of parameters, which is done in a batch-wise manner. The parameters are updated to encode features for each condition better. In an epoch, the feature representation of a condition can vary drastically between the beginning and the end of the epoch, especially when the learning rate is high.

It is also not impossible that the AGMM was incorrectly implemented on the ENet model. In the AGMM paper [14], it is claimed that the features used for the GMM predictions are extracted right before the classification layer. Consequently, the AGMM features were extracted right before the full convolutional layer in the ENet model. However, the features that are fed into the classification layer of deeplabv3plus are different from those that are fed into the full convolutional layer of ENet, which could have a substantial impact depending on how specialised AGMM is towards the features extracted in deeplabv3plus model.

5.7 Classification of conditions

This section can be seen as a continuation of section 5.3 that discussed differences in the two datasets. However, this section's discussion will focus more on the results between the two datasets.

5.7.1 Dataset 1

As seen in the confusion matrices for the first dataset bev data in Figures 4.11 4.14, 4.17 and 4.20, all models struggle with the same conditions but in various degrees. It is also clear that most misclassifications are between adjacent conditions, i.e. dry and wet, wet and slush, etc. The most reasonable argument for this is that adjacent conditions share more characteristics than non-adjacent conditions and are the primary reason for ambiguity in the soft labels. Ambiguous in the sense that the soft labels will indicate a large probability for both adjacent conditions. As such, these adjacent conditions share a boundary in feature space, and in cases where the model extracts features that are close to the boundary, it will be prone to misclassification. This is also likely a reason why the dry condition achieved the highest accuracy since this is a condition where the default characteristics are the most distinguished.

Adjacent conditions are also more likely to appear in inhomogeneous data due to more ambiguous laser annotations. This could explain why the largest misclassifications happen between slush and wet conditions as well as ice and snowy conditions. The first dataset contains more inhomogeneous data of these conditions compared to the other conditions, and image set 3 in Figure 4.12 even showcases a perfect example of inhomogeneous data with mixed wet and slush conditions. Another example can be seen in image 5 in Figure 4.9, which admittedly is non-bev data but still from the first dataset, which presents a case where the features in the roads are

at the boundary between ice and snow.

The common misclassification of adjacent conditions is also the main reason why slip accuracies were introduced. All conditions were divided into two classes: slip and non-slip. Slip conditions were slush, snow, and ice, and non-slip conditions were dry and wet. The resulting slip accuracies are presented in the tables in section 4.10. The slip classes are a bit more lenient towards models misclassifying adjacent conditions and arguably represent more relevant information since, in real-life applications, it is more interesting to know whether there is a risk of slipping than the exact road surface condition. At the same time, one could argue about how the road surface conditions were divided into the slip classes. For instance, does a wet road not also pose a risk of slipping? However, this project adhered to implementations in the provided pipeline and did not choose to make any adjustments regarding slip and non-slip conditions.

The same misclassification biases that occurred in bev data for the first dataset also occurred in non-bev data, which makes sense. However, as seen in the confusion matrices in the Figures 4.26, 4.23, 4.30 and 4.20, the models were generally better at distinguishing adjacent conditions compared to bev data. There are several reasons why that could be. Firstly, non-bev data does not depend on outside parameters introduced in the bev transformation. Secondly, non-bev data contains more contextual information, which the model may or may not have utilised. Thirdly, on average, there are more labels in the non-bev data and, therefore, more information that can be used for supervision. These three factors are discussed more thoroughly in section 5.8.

Lastly, the larger labels in non-bev data also add more information to the non-bev data. However, one can argue that larger labels should perform less when encountering inhomogeneous data, which is claimed to be the main culprit for misclassification since the assumptions made when using the larger labels should not hold for inhomogeneous data. This could be the case, and it may be the reason why there were still quite a few major misclassifications, even in non-bev data. However, it seems that, in general, the increased amount of information made up for the fact that the features in some pixels may have been mislabeled.

One can also argue that the Gauss labelling scheme for bev data also increases the amount of information in a similar way as large labels. Still, one has to acknowledge that the Gauss labels add less and more uncertain information, in the sense that they are weighted in a Gaussian manner, while the larger labels used in non-bev data add what the model interprets as 100% certain information to unlabeled pixels. However, it is a bit difficult to validate this conclusion since the large labels were, unfortunately, not implemented for bev data.

5.7.2 Dataset 2

The results of the bev data in the second dataset are more contradictory between different models than those of the first. However, the same argument of misclassifications between adjacent conditions still applies as in the first dataset. Additionally, some confusion matrices indicate that there is a severe misclassification between wet

and dry conditions for both bev and non-bev data in the second dataset. This is likely because the sessions in the second dataset were manually picked to include a larger fraction of data in darker settings.

Now, there are several factors used when models extract features from the data and interpret what they indicate. The theory section discusses that convolutional layers further extract high-level features such as objects and faces. However, the provided data does not really have any high-level features such as mouth and eyes that would indicate a human face. As such, the low-level features are more relevant, and the most prominent of those is undoubtedly the colour, which is also the main factor in human classification of road conditions. That said, wet roads are generally darker than dry roads, which could be attributed to the difference in reflective ability between the two and the fact that dry roads are typically encountered during sunny weather and wet roads, which follow rain, during cloudy and murky weather. Consequently, dry roads during nighttime will look like wet roads, causing misclassification between the two. Semblances of this behaviour can be seen in image 1 in Figure 4.53, where the labels are correctly classified, but the surrounding road is classified as the dry condition instead of the expected wet condition. Another reason could be that much like inhomogeneous data in the first dataset often includes wet and slush conditions, the second dataset's inhomogeneous data often includes wet and dry roads. However, that does not explain why it is mostly the wet conditions that are misclassified as dry and not vice versa.

5.8 Difference between bev data and non-bev data

The company had implemented a function that performed an image transformation on the original image in the provided pipeline. Much like the transformation seen in [43], it is intended to create a bird's-eye view of the road. The transformation is performed by utilising the known camera parameters to generate an image of a 50x8 meter road segment, one meter ahead of the centre of the wheel section of the recording vehicle. There are several potential benefits of such a transformation, and one of the more significant is that the bird's-eye view sample is almost guaranteed to contain only the road and no irrelevant background, thus creating a more intuitive view of the road.

The transformation also ensures that the pipeline is only fed labels that are relatively close to the recording vehicle, which are considered more reliable. Laser annotations that are further off suffer more from flaws seen in the dataset, such as tapering quality and sub-optimal motion estimation. However, image transformation also has some drawbacks. One of them is the added uncertainty introduced by the transformation itself. Several more sources of error are introduced due to the dependency on the camera's parameters. Such parameters could contain further estimations and noise, even though they are regarded as known. The transformation also limits the amount of information that the pipeline is given. A lot of contextual information that the model may use is lost by reducing the original image to a road segment of 50x8 meters.

In contrast, a much less altered version of the original image was also used as input to the pipeline, referred to as non-bev data. The original image is still cropped to reduce the amount of, more or less, irrelevant background. However, it isn't easy to crop all of the background since each image's road differs. Some roads have turns, and others go straight ahead. Therefore, it is impossible to create a static function that is able to crop everything but the road. It could perhaps be done dynamically by using laser data or by utilising another machine learning pipeline to detect roads in an image. Even so, such advanced cropping functions would also contribute to further error sources to solve a relatively minor problem, such as some background still being left in the image. It would also cause the input images to be of different resolutions, which could lead to further problems. Furthermore, it is uncertain whether removing all background is even desirable since it could provide some contextual information that the model may be able to utilise.

However, since the background is expected to be quite far away from the laser annotations used as supervision in this project, it is unlikely that the pipeline will put much weight on it while training. Some machine learning models and methods, such as AGMM, still use unlabeled regions while training. However, these regions have less impact than the labelled annotations, and the number of pixels connected to roads will still majorly outnumber the pixels connected to the background.

Compared to using the bev samples as input, the cropped versions of the original image will contain a lot more information from which the pipeline can learn. For example, these images will, on average, contain more labelled annotations. It also does not have any dependency on outside parameters, such as the camera's parameters, which leads to fewer estimations and, as such, fewer sources of errors. However, flaws in the dataset, such as sub-optimal motion estimation, will have more impact since these images contain more annotations that are further away. In the same spirit, the labels further away will also suffer from a decrease in resolution.

The non-bev samples will contain more information, but the information will be of less quality unless the dependency on outside parameters outweighs the flaws of labels further away. Additionally, as discussed in section 5.5, there is a trade-off between having more reliable information and having more information in general, and it is argued that for the provided dataset, more information results in better performance, even though it is on average less reliable. This is evident in the results as non-bev data generally achieved higher performance than bev data.

Since the provided data was sparsely annotated, it was also difficult to evaluate the actual performance of the resulting semantic segmentation on unlabeled pixels. As such, the project required some manual evaluation, where the model was applied to example images from validation data or entire sessions containing consecutive frames. Manual evaluation is easier for non-bev data since it is largely unaltered, with the exception of some cropping functions. This makes it easier for a human to determine the actual road conditions. In comparison, bev transformation sometimes results in more ambiguous road segments, making it difficult even for humans to determine the correct road conditions. Therefore, the manual evaluation is more suited for non-bev data as humans.

5.9 Data Augmentation

Data augmentation is a valuable tool to prevent overfitting and increase the robustness of a machine learning pipeline [17]. By introducing data augmentation, the dataset is artificially extended, creating more information from thin air. Depending on the model's and the dataset's complexity, this can lead to a significant improvement during evaluation. However, it is important to apply augmentations that are appropriate to the problem at hand. Otherwise, augmentation can instead severely harm the performance. It is not desirable to augment the data to an extent where it is no longer similar to the rest of the dataset.

Some of the more common methods of data augmentation are implemented in the pipeline. Augmentations such as vertical and horizontal flip and horizontal translation were primarily intended to spread the locations where laser annotations can appear. Due to how the provided dataset was created, the laser annotations were at almost the same place in every image. During training, the model could pick this up and set less priority to predicting the regions where laser annotations are less likely to appear. This could be beneficial to increase the hard accuracy, which was only dependent on the sparse labels. Still, the project aims for a semantic segmentation of the original image which can predict the road surface condition on the entire road and not just right in front of the moving vehicle.

The project has also used augmentations to reduce or increase quality and adjust the image's colouring. The augmented quality was meant to address the problem that laser annotations further away will have less resolution. The pipeline should also be able to predict the road surface condition at any time of day. By adjusting the colour of the image, the model should be able to adapt to roads in different lighting. For instance, a road at night will look substantially different compared to the same road with light from a bright morning sun.

However, while training the network on non-bev data, it was clear that the pipeline's performance was much better without augmentations. With all of the augmentations active, it essentially predicted a single condition over the entire image, as illustrated in Figure 4.3. This behaviour could still lead to relatively high accuracies since many images contained roads of a single condition, and even more images had laser annotations that were all of a single condition.

Even so, this project aimed to create a more robust pipeline for semantic segmentation, and as such, the behaviour was certainly undesirable. To address the problem, the pipeline was trained with only a single augmentation turned on, which was done consecutively for all data augmentations. Thus, when the behaviour appeared, one could determine the culprit. As it turned out, the augmentation that caused the undesired behaviour mentioned above was the translation in the x-direction. It is a bit unclear why the behaviour appeared because of this augmentation, but perhaps the network realised that since the only feedback was from the sparse labels, it does not matter what condition it predicts for the other pixels. Consequently, it is possible that the network only learned to predict the conditions where sparse labels occur. Without the translation to shift these positions, it only puts semi-random

predictions elsewhere. Therefore, when the translation was in place, it learned that instead of trying to make pixel-wise predictions, it would simply predict a single condition over the whole image since it was common for the input images only to have labels of one condition. On top of this, the behaviour was not punished sufficiently due to the loss function being limited to sparse labels. One could imagine that a faulty implementation of the augmentation would cause this, but the augmented frames were visualised, and no peculiarities could be found. Additionally, the features of pixels in unlabeled areas may differ from those in labelled regions due to factors such as view angle. Consequently, when the translation was applied, and the model supposedly had to diversify its attention, the differing features created contradictions from which the undesired behaviour arose.

However, it was not only the translation augmentation that caused problems. The other data augmentation also reduced the performance, that is, accuracy and mIoU, for non-bev data. While the accuracy and mIoU are not final evaluation metrics, since they only evaluate the model's performance on the sparse labels, they are indicators of performance. The actual semantic segmentation is of most interest. Still, there were no significant indications while visualising the model's performance on a complete data session, suggesting the augmentations improved the overall semantic segmentation while harming the accuracy and mIoU. Most likely, this could be due to the characteristics of the datasets. That is, the training and validation data were too similar, even though the data originates from sessions exclusive to either training or validation.

Data augmentation's primary function is to adjust the training data to create situations that are not present in the training data but can appear in the validation dataset or, more importantly, in real life. For example, a road could turn both left and right, therefore it could be beneficial to apply a flip to make sure roads that turn both left and right exist in the training dataset. However, the decreased performance with applied augmentations indicates that the training and validation datasets are already so alike that augmenting the training data causes unrealistic scenarios that do not appear in the validation data, which leads to reduced performance. A solution could be to try less aggressive augmentations by lowering the magnitude or probability of these augmentations being applied so the augmented images are not as different from the original ones. Additionally, it is possible that the augmentations used, while effective for bev data, are not suited for non-bev data, as they create scenarios that are unlikely to occur in real life. As such, it would have been beneficial to implement different methods to augment the non-bev data. However, this project already had a significant amount of other hyperparameters, and training the network is costly in terms of time, which is limited to the project. Therefore, it was decided that the time should be allocated to try out the impact of other hyperparameters, such as different models, instead of committing to a more extensive study of which parameters or methods are best for data augmentations. That said, there was still some minor testing with different values on the data augmentations, but none of these resulted in significant discoveries. Ultimately, this resulted in the decision that training for non-bev data was to be performed without or with fewer augmentations for the first and second datasets, respectively.

5.10 Evaluation

Evaluation is an integral part of the project to assess the model's performance. Most literature on the topic of semantic segmentation uses the mean class intersection over union to determine the model's effectiveness, which has also been implemented here. However, as previously stated, most of the research on the topic is based on densely annotated datasets. In that context, the intersection over union is a perfectly reasonable evaluation metric, while the concept, introduced in section 2.5.2, does not quite suit sparse labels equally well. Furthermore, the sparse nature of the provided dataset limited the number of labels for each image, ranging from five to 50 labelled pixels, which also put constraints on evaluation metrics such as mIoU and accuracy that are solely based on the labelled pixels.

While these evaluation methods were still used as an indicator of the model's performance, it is not certain that the best model for semantic segmentation would get the best values in such metrics due to the limitations provided by the dataset. In particular, one must also take into consideration how well the model's predictions work for the unlabeled pixels, as the conventional evaluation metrics, such as mIoU and accuracy, do not represent those. For instance, as previously discussed, the model can predict a single condition over the whole image, which would still lead to a reasonable value of mIoU. However, this disregards the unlabeled pixels and does not reflect a model's ability to predict conditions over the entire image, leading to poor semantic segmentation performance.

Consequently, on top of using conventional evaluation methods, such as confusion matrices and mIoU, manual evaluation of the finished models was required to determine the semantic segmentation performance on the entire image. This was done by visualising the finished model's predictions upon example images from the validation dataset or by applying the trained model on a video consisting of several consecutive images. By doing this, one could quickly sift out the models that had undesired behaviours, such as only being able to make correct predictions in the middle of the road or models that predicted a single condition over the entire image, see Figure 4.3 for reference.

Developing an adequate evaluation method that assesses the semantic segmentation performance of a model was one of the project's main objectives. The project did implement the metric mIoU and a visualisation tool that allows manual evaluation of the performance on complete sessions. However, these evaluation methods are still unsatisfactory. The mIoU is a metric that is often used in semantic segmentation. Still, the concept of mIoU, introduced in section 2.5.2, is more suited for semantic segmentation on densely annotated datasets and falls flat on sparse annotations. As such, in the context of this project, the mIoU could easily have been replaced with a simple mean accuracy of class conditions, which were already calculated in the provided pipeline.

Additionally, while the visualisation tool allows for assessing the model performance should it be deployed in a real-life scenario, it still requires manual evaluation, which is undoubtedly undesirable. Furthermore, the visualisation tools are the only

means to evaluate the model's performance on unlabelled pixels and are, therefore, paramount in the final evaluation of semantic segmentation. Having such an important tool be confined to manual evaluation is unsound and does not fulfil the goal of an accurate evaluation method. However, due to the project's limitation of sparse annotations, no alternative methods were found to assess the model's semantic segmentation performance comprehensively. As far as the project is aware, the absence of information on unlabeled pixels prevents the implementation of complete evaluation methods, leaving manual evaluation as the only viable option.

5.11 Future work

While this project's quantitative study is extensive, the results are pretty indecisive, and it is hard to draw any decisive conclusion on whether one model works better than another. It is also evident that the project did not manage to significantly improve the performance of the provided pipeline, and as such, it is still unsuitable for deployment due to unsatisfactory performance. In particular, deployment would entail high-stakes decision-making, which would require higher class accuracies than those achieved in the project. However, there is still room for future work regarding the given data and the provided task.

While the project has addressed numerous different hyperparameters and evaluated their impact on the pipeline's performance, many still deserve more attention, which the project could not provide due to time limitations. One of the more prominent is the learning rate while using SGD as an optimiser. According to Karpathy [31], the optimal learning rate region of SGD is relatively small compared to other optimisers such as Adam. It was found that with the base learning rate of 0.01 for SFNet and PIDNet and 0.001 for Deeplabv3plus, and a polynomial learning rate scheduler SGD did outperform Adam in this study, but the search for the best learning rate is relatively shallow for this project. The impact of the base learning rate was studied with large step sizes where it was observed that the learning rate of, for instance, 0.01 beat the learning rate of 0.001 for SFNet in terms of performance. However, assuming that the optimal learning rate region is narrow for this problem, further research is necessary to conclude that the optimal learning rate has been found.

Another prominent hyperparameter that did not get the attention it deserves is data augmentation. It was noticed that augmentation, such as translation in the x-direction, caused undesired behaviour in both datasets while applied to non-bev samples. This could have multiple causes, but one of the more reasonable ones is that the augmentation was applied too aggressively. Perhaps if the magnitude of the translation had been smaller or applied with less probability, it would have provided an increase in performance instead of harming it. As such, it would be interesting to perform further studies while varying the magnitude and probabilities of data augmentations, not just translation, to find which combination and values would lead to the best classification performance. Furthermore, the lack of increased performance while applying data augmentations on non-bev data indicates that the chosen augmentations are not well suited for that kind of data as they may represent scenarios that are unlikely to appear in real life. Consequently, a pursuit to find

other more suitable methods to augment non-bev data would likely benefit the model performance, making it better at generalising to new data.

Another limiting part of this study is the provided data, and since the choice of model did not significantly affect the pipeline's performance, it is clear that the focus should instead be put on improving the data. Firstly, there exists plenty of flaws in the data and their corresponding labels, the most prominent being the motion estimation. The project did not put a lot of effort into sifting through all training or evaluation frames, and therefore, there is no telling how many of these frames were subjected to the flaws found in the data, but it was undoubtedly a non-negligible amount. However, if a more thorough study of the frames in each dataset had been conducted, one could have discarded the frames that were most affected by the flaws and, as such, improved the overall quality of the data.

The sparse labels are arguably the most limiting factor in the provided data. As discussed earlier, the relatively small amount of information in the sparse labels contributed to a plethora of difficulties, which were further multiplied by the limited variation in which these sparse labels were located in the input frames. In the project, there were approaches to address this problem through sparsely annotated segmentation methods such as AGMM and TEL [14, 7]. However, the effectiveness of these was, at best, questionable. While label densification techniques, such as large labels, resulted in a slight performance boost, the scalability of these more primitive methods remains quite limited.

One of the limitations of this project was that there would not be any collection of new data and no decisive effort put into correcting the laser annotations in the provided dataset. However, if the ultimate goal is to create a pipeline with the best possible semantic segmentation performance, pushing towards a more densely annotated dataset would certainly be beneficial. Two approaches to achieve this would be increasing the area covered by laser detectors and manual annotations. Both of these approaches come with their advantages and disadvantages. For instance, manual annotations are subjected to human error and in comparison to other densely annotated datasets that are manually annotated, such as Cityscapes [5], it is arguably more challenging to tell apart different road surface conditions in the provided data used in this project than say a person and a road sign in data used to create Cityscapes. Thus, human error would inflict a larger limitation in model performance than seen for models trained on the Cityscapes dataset.

The other approach, to provide more dense annotations through laser detectors that cover a larger area, would add more information that can be utilised while training, which is beneficial. The laser annotations would also be more widespread, reducing the model's bias towards classifying pixels that are more commonly covered by sparse labels, as seen in the result of this project. However, such a dataset would still not provide as much information as a fully densely annotated dataset, substantially limiting the performance. An idea to lessen the limitations seen in manual annotation and more dense laser annotations is to combine them. The laser annotations could serve as guidelines for the person annotating, which would lessen the limitation imposed by human error and still result in densely annotated data. However, manual annotation requires a lot of tedious labour, and extended laser

annotation would also come at significant costs, which may be a deterring factor for any future work.

Furthermore, the project presents results for two different datasets and argues that while the first dataset contains better-quality data, the second dataset contains a larger quantity of data that is also more varied and should, therefore, lead to better generalising models. However, the project fails to test whether these theories are valid. Consequently, future work could involve developing a test dataset containing sessions not included in any of the training datasets. Such a dataset could then be used to evaluate which of the training datasets is better suited for the task at hand.

This project limited itself to already developed models and, more specifically, models developed for other datasets such as Cityscapes [5], which may share some similarities to the projects provided data but are still vastly different. Consequently, these models include functions specifically designed to improve performance on the Cityscapes dataset but do not work as well in this project. An obvious example would be the D branch in the PIDNet model, which is used to generate a boundary loss. This is done by extracting the boundaries in the ground truth of the densely annotated dataset, but these boundaries do not exist in the sparsely annotated data provided for this project, which severely limits the impact of PIDNet's D branch. Therefore, future work could be to design a custom machine learning model with functions specifically designed to utilise the characteristics found in the provided data for this project.

6

Societal, ecological and ethical aspects

This chapter will briefly discuss any societal, ecological, and ethical aspects that must be considered while working on a project such as the one represented by this report. Do note that much of the discussion in this section revolves around the deployment of the pipeline developed in the project and assumes the pipeline would be ready for deployment. However, as the project results indicate, the pipeline's performance is limited and comes with severe flaws that make it unsuitable for deployment.

6.1 Societal aspects

The societal aspects of this project are significant. The deployment of a pipeline such as the one developed in the project can be a factor in enhanced driving safety as the information gathered will let drivers and potential automated cars adjust their driving behaviour and employ appropriate safety procedures. This could lead to increased road safety by reducing the number of road accidents, injuries, and fatalities.

Furthermore, the information gathered by the pipeline could also be shared, giving real-time data on a road's surface condition. If the system is deployed in a considerable number of vehicles, the shared information could be used to estimate the condition of roads on a larger scale. This could improve overall traffic management by, for instance, informing authorities where road maintenance, such as snowploughing, is necessary.

6.2 Ecological aspects

Deploying a pipeline such as the one developed in the project could have significant benefits from an ecological point of view. The pipeline's purpose is to be a factor in improved road safety by reducing the number of accidents that occur due to hazardous road conditions. Such road accidents could potentially lead to major traffic blockages, which in turn leads to idling time for vehicles or potentially longer detours, both of which increase fuel consumption and emissions.

However, it is also important to take into consideration the energy that is used in

the creation of such a pipeline. The training and deploying of a pipeline like this requires a significant amount of computational power, which in turn comes with a large energy consumption and environmental impact. Consequently, it is important to implement more efficient algorithms and model architectures wherever possible. Furthermore, as utilisation of cloud services is inevitable, another factor to consider is ensuring that the cloud services used are committed to renewable energy sources and using energy-efficient hardware.

6.3 Ethical aspects

A machine learning pipeline like the one proposed in this project has a few ethical implications. One of them being transparency. Transparency is a common villain when working with the black box models of neural network architectures and should be considered, especially when using them for high-stakes decision-making [44]. Such models require a significant amount of data to train, and it is difficult to gather such an amount of data without introducing flaws such as biases. In particular, it was important in this project to investigate the limitations of the pipeline and any biases the pipeline has acquired during training. In the case of deployment, such information should be made public and known to potential users so that necessary precautions can be taken. For instance, it was noticed that the pipeline was prone to making incorrect predictions during nighttime, indicating that, should the pipeline be deployed, the driver should not wholly rely on the pipeline while driving at night.

Another ethical implication is responsibility. Consider a case where the pipeline is deployed into either a semi- or fully autonomous car and has failed to predict the condition of the road surface correctly. This failure has then led to an accident on the road. Is it the driver, the car manufacturer or the developer who should take responsibility? This could, of course, be situation-dependent, but if the responsibility falls on anyone but the developer, it strengthens the incentive for transparency. The user needs to be aware of how the model makes its predictions, any possible limitations and biases that could cause it to make such a failure, and the conditions for which it is prone to do so.

The societal aspects, in section 6.1, discussed the possibility of sharing the information gathered by the model. This brings us to the next ethical consideration, namely privacy. Information sharing also comes with the potential to link the shared information to individual drivers, which is an infringement of privacy. Measures must be taken to safeguard privacy rights. The collector of information should not be able to track individual cars/drivers. There needs to be clear regulations that govern the use of such information, with a focus on protecting individuals' privacy while still being able to use the benefits that shared information comes with.

7

Conclusion

In conclusion, the project performed a quantitative study to evaluate the efficiency of various machine learning models and label enhancement techniques using two different approaches to data parsing: cropped images and bird's-eye view data.

The results show that while the simpler ENet model achieved the highest performance on non-augmented non-bev data in the first dataset, it was outperformed by the more complex models on more augmented bev-data and the more varied non-bev data in the second dataset. This could indicate that the more complex models, such as SFNet and PIDNet, are slightly better at generalising than ENet. However, there is no significant difference in performance, especially for non-bev data, which indicates that the model's complexity does not greatly impact the final performance. This was further reinforced in the results between different-sized PIDNet models, where the large variant of PIDNet achieved essentially the same performance as the small variant, even though it contains more than four times the number of parameters.

An explanation for the limited performance increase of complex models is that the provided data does not benefit from the deeper layers in more complex models. The most important parameters when determining the condition of a road are the colour and texture. These are examples of low-level features extracted in the early layers of a neural network. In contrast, deeper layers capture high-level features, such as shapes or parts of objects, which are not prominent in the provided data. Consequently, the high-level features extracted from the more complex models are not utilised to classify the road surface conditions, limiting their impact on performance. Furthermore, the complex models may be more prone to overfit the flaws identified in the provided data.

Moreover, the results indicate that the efficiency of label enhancement techniques is highly dependent on the data used while training. The Gaussian and large square labels, which propagate the properties of labelled pixels to unlabelled pixels in close vicinity, were more efficient on the second dataset than the first, likely due to the second dataset containing more homogeneous data. Additionally, the more sophisticated methods of AGMM and TEL, which were used to extract supervision from unlabeled pixels, did not perform well for the provided data. The cause of these methods' lack of performance is likely due to the characteristic difference between the provided data and the data they were developed for. For instance, the methods are highly dependent on the fact that the sparse labels in the image represent all

conditions present in the input image. If that is not the case, the methods can not find feature similarities between labelled and unlabelled pixels, making the methods fall flat and could even introduce unreliability in the added losses, causing harm during training.

The sparsely annotated dataset is a significant limiting factor and the project's focal point. It has flaws, such as faulty motion estimations, and the sparse labels severely constrain the amount of supervision that can be provided to the models. The limited spread at which the sparse labels appear in the data also contributes to significant biases in the model's field of view. Most labels are in a straight line in the centre of the image, constraining the model's field of view. Consequently, the models will struggle to predict the condition of pixels outside of this region. An additional factor is that the features of pixels outside the region of sparse labels likely differ slightly from pixels inside the region due to factors such as viewing angle, which complicates the prediction of unlabelled pixels even further. This could also be a significant factor in why the AGMM and TEL methods underperformed, as they are primarily based on comparing features between labelled and unlabelled pixels.

Furthermore, the sparse labels limit the ability to evaluate the models' semantic segmentation performance. Conventional evaluation metrics, such as accuracy and mIoU, can only be computed on the labelled pixels and, therefore, do not completely reflect the model's ability for semantic segmentation. Consequently, a need for manual evaluation arose with the primary goal of quickly sifting out methods that present significant flaws in semantic segmentation.

Future research should conduct more extensive studies on learning rates and data augmentations. When applied correctly, these factors can heavily impact the pipeline's performance. Nevertheless, the project focused more on the impact of different machine learning models and label enhancement techniques. Consequently, the impact of learning rate and data augmentations was not studied sufficiently. Additionally, it would be beneficial to put more effort into identifying flawed frames in the dataset and discarding them, improving the overall data quality. Furthermore, pushing toward a more densely annotated dataset would be beneficial. This would not only provide more supervision to the model but could also diversify the locations at which sparse labels appear and would better imitate the densely annotated datasets on which most research within semantic segmentation is conducted. More dense annotations could be introduced by adding more laser detectors to the recording vehicles or through manual annotation. Combining both would yield the best result as the methods complement each other in a desired way. More laser information would assist the manual annotation by reducing the impact of human error. Lastly, developing a custom machine learning model tailored to the specific characteristics of the dataset, rather than utilising models developed for other data, could potentially lead to significant performance gains.

Overall, This project underscores the need for tailored approaches in machine learning applications, emphasising that more complex models are not always the best choice for every dataset. Furthermore, label enhancement techniques such as the Gaussian labels and large square labels led to a trade-off between more information and reduced average data quality, which is largely dependent on the characteristics

of the dataset. The more sophisticated label enhancement methods of TEL and AGMM were not well-suited for the provided data, leading to undesired behaviours or decreased performance. Future research should focus on tuning hyperparameters such as learning rate and data augmentations, which were limited in the project. Additionally, it would be beneficial to focus more on improving the data quality, either through discarding flawed frames in the training data or adding more annotations via more laser detectors or manual annotation.

Bibliography

[1] OpenAI, "ChatGPT," 2024, large language Model. [Online]. Available: https://www.openai.com

[2] Grammarly Inc., "Grammarly," 2024, writing assistant software. [Online]. Available: https://www.grammarly.com/about

[3] Y. LeCun, C. Cortes, and C. J. C. Burges, "The mnist database of handwritten digits," 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist

[4] Roboflow, "Mnist 42000 images dataset," https://universe.roboflow.com/roboflow-jvuqo/mnist-42000-images-u0qdg, apr 2023, accessed 26-May-2024. [Online]. Available: https://universe.roboflow.com/roboflow-jvuqo/mnist-42000-images-u0qdg

[5] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. [Online]. Available: https://www.cityscapes-dataset.com/

[6] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognition Letters*, vol. 30, no. 2, pp. 88–97, 2009, video-based Object and Event Analysis. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167865508001220

[7] Z. Liang, T. Wang, X. Zhang, J. Sun, and J. Shen, "Tree energy loss: Towards sparsely annotated semantic segmentation," 2022. [Online]. Available: https://arxiv.org/abs/2203.10739

[8] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jul 2017, pp. 6230–6239. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2017.660

[9] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "Enet: A deep neural network architecture for real-time semantic segmentation," *ArXiv*, vol. abs/1606.02147, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:16231053

[10] J. Xu, Z. Xiong, and S. P. Bhattacharyya, "Pidnet: A real-time semantic segmentation network inspired from pid controller," 2022. [Online]. Available: https://arxiv.org/abs/2206.02066

[11] X. Li, A. You, Z. Zhu, H. Zhao, M. Yang, K. Yang, S. Tan, and Y. Tong, "Semantic flow for fast and accurate scene parsing," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 775–793. [Online]. Available: https://doi.org/10.1007/978-3-030-58452-8_45

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[13] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," 2018. [Online]. Available: https://arxiv.org/abs/1802.02611

[14] L. Wu, Z. Zhong, L. Fang, X. He, Q. Liu, J. Ma, and H. Chen, "Sparsely annotated semantic segmentation with adaptive gaussian mixtures," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2023, pp. 15 454–15 464. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR52729.2023.01483

[15] FHWA, "How do weather events impact roads?" 2017, accessed: 2024-02-01. [Online]. Available: https://ops.fhwa.dot.gov/weather/q1_roadimpact.htm

[16] "Real-time semantic segmentation." [Online]. Available: https://paperswithcode.com/task/real-time-semantic-segmentation

[17] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, pp. 1–48, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:195811894

[18] R. Adrian, "Convolutional neural networks (cnns) and layer types," 2021, accessed 22-May-2024. [Online]. Available: https://pyimagesearch.com/2021/05/14/convolutional-neural-networks-cnns-and-layer-types/

[19] Arc, "Convolutional neural network," 2018, accessed 22-May-2024. [Online]. Available: https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05

[20] P. Petru, "What is a convolutional neural network?" 2023, accessed 22-May-2024. [Online]. Available: https://blog.roboflow.com/what-is-a-convolutional-neural-network/

[21] P. Paul-Louis, "An introduction to different types of convolutions in deep learning," 2017, accessed 23-May-2024. [Online]. Available: https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d

[22] A. Aqeel, "What is transposed convolutional layer?" 2020, accessed 23-May-2024. [Online]. Available: https://towardsdatascience.com/what-is-transposed-convolutional-layer-40e5e6e31c11

[23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015. [Online]. Available: https://arxiv.org/abs/1502.03167

[24] G. MK, "Basic cnn architecture: Explaining 5 layers of convolutional neural network," 2022, accessed 22-May-2024. [Online]. Available: https://www.upgrad.com/blog/basic-cnn-architecture/

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," 2015. [Online]. Available: https://arxiv.org/abs/1502.01852

[26] Ahmadsbry, "A perfect guide to understand encoder decoders in depth with visuals," 2023, accessed 23-May-2024. [Online]. Available: https://medium.com/@ahmadsabry678/a-perfect-guide-to-understand-encoder-decoders-in-depth-with-visuals-30805c23659b

[27] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," 2017. [Online]. Available: https://arxiv.org/abs/1612.03144

[28] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," 2017. [Online]. Available: https://arxiv.org/abs/1706.05587

[29] Wikipedia contributors, "Stochastic gradient descent — Wikipedia, the free encyclopedia," 2024, [Accessed 15-May-2024]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Stochastic_gradient_descent&oldid=1222784950

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. [Online]. Available: https://arxiv.org/abs/1412.6980

[31] A. Karpathy, "A recipe for training neural networks," 2019, accessed: 2024-01-31. [Online]. Available: https://karpathy.github.io/2019/04/25/recipe/

[32] M. Sujatha, "Weight decay in deep learning," 2023, accessed 06-June-2024. [Online]. Available: https://medium.com/@sujathamudadla1213/weight-decay-in-deep-learning-8fb8b5dd825c

[33] H. N. B, "Confusion matrix, accuracy, precision, recall, f1 score," 2019, accessed 05-July-2024. [Online]. Available: https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-score-ade299cf63cd

[34] A. Rosebrock, "Intersection over union (iou) for object detection," 2016, accessed 05-July-2024. [Online]. Available: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

[35] Wikipedia contributors, "Simulated annealing — Wikipedia, the free encyclopedia," 2024, accessed 06-May-2024]. [Online]. Available: https://en.wikipedia.org/wiki/Simulated_annealing

[36] A. Clark, "Pillow (pil fork) documentation," 2015. [Online]. Available: https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf

[37] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[38] J. Ansel, E. Yang, H. He, N. Gimelshein, A. Jain, M. Voznesensky, B. Bao, P. Bell, D. Berard, E. Burovski, G. Chauhan, A. Chourdia, W. Constable, A. Desmaison, Z. DeVito, E. Ellison, W. Feng, J. Gong, M. Gschwind, B. Hirsh, S. Huang, K. Kalambarkar, L. Kirsch, M. Lazos, M. Lezcano, Y. Liang, J. Liang, Y. Lu, C. Luk, B. Maher, Y. Pan, C. Puhrsch, M. Reso, M. Saroufim, M. Y. Siraichi, H. Suk, M. Suo, P. Tillet, E. Wang, X. Wang, W. Wen, S. Zhang, X. Zhao, K. Zhou, R. Zou, A. Mathews, G. Chanan, P. Wu, and S. Chintala, "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation," in *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, Apr. 2024. [Online]. Available: https://pytorch.org/assets/pytorch2-2.pdf

[39] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. [Online]. Available: https://doi.org/10.1109/CVPR.2009.5206848

[40] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, jun 2010. [Online]. Available: https://doi.org/10.1007/s11263-009-0275-4

[41] J. Bokhorst, H. Pinckaers, P. van Zwam, I. Nagtegaal, J. van der Laak, and F. Ciompi, "Learning from sparsely annotated data for semantic segmentation in histopathology images," in *Proceedings of The 2nd International Conference on Medical Imaging with Deep Learning*, ser. Proceedings of Machine Learning Research, M. J. Cardoso, A. Feragen, B. Glocker, E. Konukoglu, I. Oguz, G. Unal, and T. Vercauteren, Eds., vol. 102. PMLR, 08–10 Jul 2019, pp. 84–91. [Online]. Available: https://proceedings.mlr.press/v102/bokhorst19a.html

[42] F. Gao, M. Hu, M.-E. Zhong, S. Feng, X. Tian, X. Meng, M. yi-di-li Ni-jia ti, Z. Huang, M. Lv, T. Song, X. Zhang, X. Zou, and X. Wu, "Segmentation only uses sparse annotations: Unified weakly and semi-supervised learning in medical images," *Medical Image Analysis*, vol. 80, p. 102515, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1361841522001621

[43] S. Roychowdhury, M. Zhao, A. Wallin, N. Ohlsson, and M. Jonasson, "Machine learning models for road surface and friction estimation using front-camera images," in *2018 International Joint Conference on Neural*

Networks (IJCNN), 2018, pp. 1–8. [Online]. Available: https://doi.org/10.1109/IJCNN.2018.8489188

[44] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Machine Intelligence*, vol. 1, no. 5, pp. 206–215, May 2019. [Online]. Available: https://doi.org/10.1038/s42256-019-0048-x

A

Appendix 1

This chapter will contain a link to a Google Drive folder containing videos where the trained models are applied to an example session recorded by the same collaboration company that recorded the data in dataset 1. Consequently, the models trained on data from the first dataset are likely to perform slightly better, but the videos show that the models trained on dataset 2 do not perform that badly either. There will not be a thorough discussion of these videos, but the discussion in section 5 covers most of what is seen in these videos.

However, in the videos, one can notice that the labelling techniques that are based on soft labels cause more sporadic predictions where they are commonly changed between two conditions in between frames. While this appears inappropriate behaviour, the soft labels can easily explain it. The models trained with soft labelling techniques will try to predict the probability distribution of the pixels in the image, while the models trained on hard labels instead try to predict the most likely condition and will, therefore, typically have a large probability towards that condition. Consequently, since the image's colouring is based on the condition with the highest probability, the hard labels will appear to be more stable. Conversely, the models trained on soft labels will frequently encounter scenarios where the predictions have multiple conditions with high probability, which causes a sporadic nature since the highest probability condition often changes between frames.

The videos can be accessed through the following link: Link to the videos. Note that not all labelling techniques trained with all models on both datasets are present. Still, there should be enough videos of how the labelling techniques, models and datasets affect the semantic segmentation of the pipeline. The main idea behind including these videos is for the reader to understand better how the pipeline would perform should it be deployed in real life.

Additionally, the videos are named with model, data parsing method, labelling technique and dataset. The best way to study the videos is to open multiple of them at once, beginning with the original view of the road. Then, it depends on what aspect of the pipeline one wants to study. Should one be interested in how the choice of model impacts the semantic segmentation, open videos on the same dataset with the same labelling technique but different models; if the labelling technique's impact is more interesting, open videos with the same model on the same dataset but different labelling techniques and so on.

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY