Night-time Vehicle Detection Based on Observable Light Cues Using Deep Learning

Celine Ivarsson

Jennifer Zacke

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6195
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Abstract

This thesis investigates the issue of computer vision-based detection of oncoming cars during night-time, a critical road safety issue for automated high-beam assist. We propose a holistic image classification approach that uses deep learning methods to detect light artifacts from an oncoming car's headlights before the car is entirely visible. We explore six different model architectures, including both convolutional neural networks and transformer-based models. We train them using transfer learning with both public and internal datasets using models pre-trained on ImageNet. We evaluate the generalization ability of the models and find that they can achieve up to 71% accuracy when trained on the public dataset and evaluated on the class-balanced internal dataset. Our results show that both convolution-based and transformer-based models have potential in performance for this task, with the best models reaching up to 88% accuracy when trained with the full public dataset and evaluated with the class-balanced public test set. Our research contributes to the field by introducing an approach to detection of oncoming cars and comparing different model architectures for this task.

Acknowledgments

We would like to express our deepest gratitude to our supervisors at Qualcomm, Liam Fahlstad, Joar Karlgren Gustavsson, Mats Fridén, Simon Näsfeldt, and Rickard Peterson, for their invaluable guidance and support throughout the project. Their insightful feedback and support have been important to help us achieve our goals. We are also grateful to our supervisor at Lund University, Pontus Giselsson, for providing us with academic expertise and for encouragement and motivation during the course of this master thesis. We also want to thank our examiner Bo Bernhardsson.

In addition, we would like to extend our thanks to everyone in the Lund office at Qualcomm for their assistance and cooperation during our work.

Celine Ivarsson & Jennifer Zacke
Lund, Sweden

Contents

1 Introduction

1.1 Motivation and Aim

Technology evolves at a rapid pace and brings significant advancements to the automotive industry, especially within advanced driver assistance systems (ADAS). This has revolutionized the driving experience, with enhanced safety and efficiency being the primary benefits. A company working to drive the transition toward the future of cars is Qualcomm Technologies, which is a leading wireless technology innovator. Major advancements are made in the company regarding driver assistance technologies, including the products of its Snapdragon Ride Platform, which is a car-to-cloud, connected ADAS infrastructure.

Modern cars with ADAS are equipped with many functions, including safety systems, lane departure warnings, adaptive cruise control, and automatic emergency braking. But, the current high beam assistance systems in modern cars share one limitation; when driving at night, they turn off the high beams once the oncoming car is visible. Humans process this information quicker and can detect the car from its headlamps lighting up the road and its surroundings, or from only seeing parts of the car before it is fully visible. An automatic system for turning off the high beams can therefore be perceived as slow from a human perspective, and drivers often turn off the high beams earlier in order not to glare at the oncoming driver [Oldenziel et al., 2020].

Humans use multiple senses to predict things that are about to happen, for example, sound and visual cues from oncoming cars. Previous experiences also contribute to the process of what is happening around us. Machine learning models do what they are trained to do, which is most often one specific task, and this makes humans superior in performing more generalized tasks. Detecting oncoming cars in the dark is therefore very intuitive for a human and something we do without further thought but can be challenging for a computer.

Many computer vision algorithms rely on object detection, such as the YOLO algorithm, and [Ewecker et al., 2022] investigated the possibility of detecting oncoming cars during night-time before the car is visible. The authors presented an algorithm pipeline that is capable of detecting approaching vehicles providently during the night, almost 1.6 seconds before a conventional vehicle detection system. It does so by detecting the light artifacts caused by the headlight of the oncoming vehicle. [Ewecker et al., 2022] The dataset in [Ohnemus et al., 2021], Provident Vehicle Detection at Night (PVDN), consisting of scenes of oncoming cars in the dark, will be further used in this thesis project.

This thesis investigates the possibility to detect oncoming cars at night, before the cars are visible, by observing visual cues; the light artifacts. The light artifacts look different depending on factors such as road curvature and the surrounding environment. Typically, these artifacts appear along the horizontal centerline of the images in the video clips of oncoming cars. We also noticed that the brightness of that area in the image varies depending on the cars' velocities and whether the detecting car is turning or not. When a vehicle is visible in the image, a clear glare can be seen and the brightness increases even more. Considering all of these factors, three different types of images can be differentiated. The first is when no car, nor light artifact from a car, is visible. The second one is when a light artifact from an approaching car can be seen, for example on the road or on surrounding trees. The last category is when the car, or the headlamps of the car, are visible. These three categories are quite similar, and the primary distinction between them lies in the vertical center of the image, and instead of performing object detection on these light artifacts to solve this problem, we will investigate how image classification can be done using a holistic approach. The holistic approach in this context means to consider the entire image rather than focusing on features or smaller areas.

Convolutional Neural Networks (CNNs) are performing well in the task of image classification [Aggarwal, 2018], and are widely used in many applications such as medical imaging, video analysis, and self-driving cars. A relatively new type of network is the Transformer which is mostly used for natural language processing. However, Vision Transformers (ViTs) have shown promising results in image classification [Dosovitskiy et al., 2021]. The main difference between a Convolutional Neural Network and a Vision Transformer is the architectural building blocks and how they process the data passing through the network. A CNN consists of a series of convolutional layers that extract features from the input data, whereas the ViT uses the attention mechanism to process the image. In Vision Transformers, the image is divided into smaller patches, where the network can learn the spatial relationships among the patches [O'Shea and Nash, 2015] [Dosovitskiy et al., 2021].

There are many networks provided on APIs such as Keras and Tensorflow Hub that are pre-trained on large datasets for everyone to use. Pre-trained networks are capable of transferring the, already learned, ability to other similar tasks. For example, if a network is pre-trained on animals, the network can be adjusted to classify dog breeds. In order to do that, the last output layer, or the 'classifier' of the network, is removed and some additional layers followed by a new output layer, appropriate for the task, are attached. The remaining layers are used to extract high-level features which should be general enough to be able to train another classifier. These layers are called the backbone and the weights of the backbone are frozen as the new classifier is trained. The new classifying head is fed with the features from the backbone and trained to fit the new data containing dog breeds. This thesis will investigate whether this approach works for our task of classifying three categories of oncoming cars during night-time. This will be done using both PVDN and an internal dataset from the company.

The three categories mentioned above will be referred to as class 0, 1, and 2 and are presented in Table 1.1. We will investigate if early detection of vehicles during night-time is possible, in order to turn off the high beams, using deep learning methods suited for image classification.

Table 1.1 Categories used for labeling images in the dataset.

Category	**Description**
0	No car nor light artifact
1	Light artifact from approaching car
2	Headlights from approaching car are visible

The work during this master's thesis has been done in collaboration with Qualcomm in Lund, a leading company in wireless technology. This thesis project was conducted at the department responsible for designing functions for the advanced driver assistance systems being developed by the company. The aim of this project is to develop an algorithm that can serve as part of the foundation for the high beam assist functionality and perhaps for other safety functions in the ADAS. The algorithm will also serve as a proof of concept to demonstrate the feasibility of this functionality. Hence, this project is of interest to the company and the future development of self-driving cars.

1.2 Research Questions

The following questions will be investigated in this thesis.

- Which of the models; DenseNet, EfficientNet, NASNet, CaiT, and DeiT (see Section 2.1.4 and Section 2.1.6), will give the best accuracy and F1 score for class 1? (See Section 2.2.6)

- How does a transformer-based model perform compared to a convolutional-based model for this task?

- Which transfer learning method generates the best model in terms of accuracy and F1 score for class 1?

- To what extent can the public dataset, PVDN, be used to train the models for image classification on the internal dataset (see Section 3.1.2)?

1.3 Delimitations

The scope of this project is delimited by the following factors.

- This thesis will only test a limited number of Convolutional Neural Networks and Vision Transformers.
- The project will only evaluate the holistic approach to image classification.
- Other methods for light artifact detection will not be considered in this thesis.
- This thesis does not consider processing times or other factors, such as the size of the network, that might affect the future system function.
- The thesis will focus on image classification using only images and not videos. Videos as input data could be beneficial since looking at changes in the light artifacts could make it easier to differentiate between static lights and moving cars.

2

Classification Tools

This project aims to solve a holistic image classification task where the goal is to predict oncoming vehicles by identifying light artifacts. The images are labeled with one of the three classes; "No car nor light artifact", "Light artifact from approaching car" and "Headlights from approaching car are visible". Image classification is the task of labeling or classifying images depending on the content of the image and is a fundamental task in computer vision and image processing. In supervised learning, a parameterized model is trained using labeled images to map features from the images to their corresponding labels. During training, the model's parameters are updated and optimized to improve its performance on the given problem. Once trained, the model can predict the labels of new images based on its learned knowledge. There are many different types of neural network models suitable for solving multi-class image classification problems, such as convolutional neural networks and attention-based transformer models. This chapter provides the necessary theoretical background to understand the content of this thesis.

2.1 Classification Models

Artificial Neural Networks (ANNs) are computational models, often used for classification tasks, designed to work similarly to a human brain. ANNs contain an input layer, one or more hidden layers of neurons (computational nodes), and an output layer. The hidden layers extract patterns and are responsible for most of the internal processing. The output layer of a neural network is responsible for producing the final prediction or output of the network, which represents the result of the processing performed by the neurons in the previous layers [Silva et al., 2017].

A neuron is the part of an ANN that acts as the computational unit to extract features. It takes an input vector and returns a single output value. The inputs are individually weighted depending on the relevance of the input. The typical approach involves introducing a bias term to determine the appropriate threshold for the output, which will then activate the neuron and produce a trigger value.

The output from the neuron is

$$y = g(\sum_{i=1}^{n} w_i x_i - \theta) \tag{2.1}$$

where $g(\cdot)$ is the activation function, w is the weights, x is the input and θ is the bias [Silva et al., 2017].

A feed-forward neural network is a type of neural network in which the flow of information occurs in only one direction. Each layer receives its input from the layer directly below and sends its output to the layer directly above. During the training process, the network's weights and biases are randomly initialized and then updated iteratively using a supervised learning algorithm that adjusts the weights and biases based on the difference between the predicted and actual output [Silva et al., 2017].

2.1.1 Activation Functions

Activation functions are applied to the output of the weighted sum plus bias and decide the output of the neuron. The activation function is fundamental for the neural network and introduces non-linearity into the output of the neuron allowing the neural network to learn more complex patterns [Nwankpa et al., 2018].

ReLU. ReLU stands for Rectified Linear Unit and is a commonly used, non-linear activation function. The function returns the input value for all positive values, and 0 for all negative input values, see Equation 2.2. Due to the simplicity of the mathematical operation, it is computationally efficient and the function also addresses the vanishing gradient problem, where the gradient can become so small during backpropagation that it hinders effective learning [Nwankpa et al., 2018].

$$f(x) = max(0, x) \tag{2.2}$$

GeLU. The Gaussian Error Linear Unit, or GeLU, is a high-performing activation function similar to ReLU. Neuron inputs tend to follow a normal distribution which is something GeLU takes advantage of by scaling the input with the cumulative distribution function of the standard normal distribution. The output from the GeLU function is calculated through

$$GeLU(x) = x\Phi(x) \tag{2.3}$$

where $\Phi(x)$ is the cumulative distribution function of Gaussian distribution [Hendrycks and Gimpel, 2020].

Softmax. The softmax activation function maps the output of a network to a probability distribution over predicted classes and is therefore often used in the output layer of neural networks for multi-class classification problems. The formula for the softmax function is

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K \tag{2.4}$$

where z_i is the output value of the i^{th} neuron in the output layer, e is the Euler's number, and K is the number of classes [Nwankpa et al., 2018].

2.1.2 Loss Function

A loss function is used to calculate the error rate of the current state in the network. It is used as part of the optimization process when updating the weights during training, to obtain a low loss in the model. A common loss function for multiple-class problems is categorical cross-entropy, where the model output is a probability over the classes for each image. Categorical cross-entropy is defined as:

$$L_{cls} = -\sum_{c=1}^{M} y_c \log(p_c) \tag{2.5}$$

where M is the number of classes, y_c is 1 if the prediction is correct and 0 otherwise, and p_c is the predicted probability that the observation belongs to class c. For multi-class problems, the probability is usually computed with a softmax layer, as in Equation 2.5, that converts the output to probabilities.

2.1.3 Convolutional Neural Networks

A Convolutional Neural Network is a type of Artificial Neural Network that is specifically designed for image recognition tasks. Like many ANNs, CNNs consist of neurons that perform computations. CNNs have a specific architecture that is optimized for extracting features from images and consists of convolutional layers, pooling layers, and fully-connected layers. The neurons within the layers in CNNs are organized in three dimensions, the height, width, and depth of the input image. [O'Shea and Nash, 2015]

Convolutional layers are a crucial part of CNNs and consist of a set of learnable kernels. The kernels are small matrices that slide across the image. The dot product of the kernel and image is computed for every spatial position. Each kernel generates an activation map, and stacking these maps along the depth dimension creates the output volume of the convolutional layer [O'Shea and Nash, 2015].

Pooling layers are usually added immediately after convolutional layers and reduce the size of the feature maps and the computational complexity while retaining the most important information. The downsampling is done using a sliding window approach, where a small window of fixed size moves across the input tensor and collects the values in that region using an operation such as max or average pooling.

Fully connected layers, also known as dense layers, are a type of neural network layer where each neuron in the layer is connected to every neuron in the previous layer. In other words, all the outputs from the previous layer are fed as inputs to each neuron in the fully connected layer [Bezdan and Bacanin, 2019].

2.1.4 Convolution-Based Networks

Over the years, several CNN architectures have been proposed that show competitive accuracy, efficiency, and scalability. DenseNet, EfficientNet, and NASNet, described below, are popular CNN architectures that have demonstrated state-of-the-art performance in various computer vision tasks.

DenseNet. DenseNet was first introduced in [Huang et al., 2017]. The paper proposes a network where each layer is connected to every other layer in a feed-forward manner to establish maximum information flow between the layers. Each layer takes input from all preceding layers and passes its own feature maps to all following layers. With this approach, an L-layer network gets L(L+1)/2 connections instead of just L. Dense connectivity requires fewer parameters than traditional convolution networks since each layer receives the feature maps of all preceding layers as inputs and improves the flow of information and gradients throughout the network, making them easier to train. The dense connectivity also has a regularizing effect which reduces overfitting when using smaller training sets.

EfficientNet. EfficientNet is a family of neural networks, designed to achieve higher accuracy while reducing the number of parameters. The architecture was first introduced in [Tan and Le, 2020] and is based on compound scaling which uniformly scales the depth, width, and resolution using a compound coefficient. EfficientNet reaches high accuracies on ImageNet (84.3% top-1) and also transfers well with standard transfer learning datasets such as CIFAR-100 and Flowers, with accuracies of 91.7% and 98.8% respectively.

NASNet. Neural Architecture Search Network, NASNet, is a neural architecture search approach that was first introduced in [Zoph et al., 2018]. NASNet is a machine learning algorithm that learns to design neural network architectures that perform well on a given task by trying out different architectures and evaluating their performance. The algorithm consists of a controller network that learns to generate new architectures by sampling from a search space. This means that NASNet automatically can discover architectures that are optimized for a given task.

2.1.5 Transformers

Transformers were introduced in [Vaswani et al., 2017], where the network architecture is entirely based on attention layers to compute representations of the input and output. The attention mechanism allows for input parallelization and selective focus on the different parts of the input sequence while processing it. A transformer usually consists of an encoder part and a decoder part, where the encoder extracts features from the input sequence, and the decoder produces an output sequence from the features. An input sequence of symbol representations $\mathbf{X} = (x_1, ..., x_n)$ is mapped to a sequence of continuous representations, $\mathbf{Z} = (z_1, ..., z_n)$, by the encoder part. The decoder then generates an output sequence $\mathbf{Y} = (y_1, ..., y_m)$ from $\mathbf{Z}$. Both the encoder and decoder are built of multiple layers containing self-attention and feed-forward sub-layers [Vaswani et al., 2017]. Transformers are mainly used for natural language processing but have recently proven to also be effective for image classification tasks, using the Vision Transformer (ViT) [Dosovitskiy et al., 2021].

Attention, Self-Attention & Multi-Head Attention. Attention is a method of mapping a query and a set of key-value pairs, to an output. Self-attention, which is a type of attention, represents how the input sequence's elements relate to one another, instead of one sequence's relevance to another. The input, consisting of the queries and keys, is of dimension d_k. The weighted sum is computed for the values which become the output, where weight is computed by a compatibility function of the query, and the key is assigned to each value. The dot product between the queries and keys is then computed, these are divided by $\sqrt{d_k}$, and a softmax function is applied, in order to obtain the weights of the values. This type of attention is called 'Scaled Dot-Product Attention' [Vaswani et al., 2017].

As mentioned, while processing the input sequence, attention is used to selectively focus on the input's different parts. The attention function is computed simultaneously on a set of queries that are packed together in a matrix Q (for queries) with K, a matrix containing the keys, and V, a matrix with all values [Vaswani et al., 2017]. The attention function is defined as

$$\text{Attention}(Q,K,V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

and allows the model to learn where the important parts of the input are, in order to generate the output [Vaswani et al., 2017].

Multi-head attention, which is a variation of self-attention, is when the attention function is applied in parallel for all input values. The self-attention operation is instead performed multiple times, each time with a different set of the learned query, key, and value matrices, also known as 'heads'. The model can then improve the ability to attend to different parts of the input sequence and also learn different features from it in parallel [Vaswani et al., 2017].

2.1.6 Transformer-Based Networks

The breakthrough for Vision Transformers (ViTs) was in [Dosovitskiy et al., 2021]. The approach presented appeared as an alternative to the established CNNs and outperformed them in accuracy and computational efficiency. Subsequently, researchers have developed several ViTs, for example, CaiT and DeiT, customized to address different requirements and tasks.

ViT. The Vision Transformer's model architecture follows the original transformer architecture in [Vaswani et al., 2017] as closely as possible due to its efficient implementation and ease of deployment. It was adapted to handle 2D images instead of text sequences as input and consists of only the encoder part of the transformer. The output of the Vision Transformer is instead a class prediction from a classifier head attached at the top [Dosovitskiy et al., 2021].

In order to adapt the original transformer to handle 2D images, the images have to be pre-processed in several steps before it's provided to the network. Firstly, the input image is divided into a sequence of smaller, fixed-size 'patches'. The sequence is then flattened, where each patch is linearly embedded, with the size of the projection dimension determined by a hyperparameter. The embedded patches are then assigned a positional embedding. The resulting sequence is then ready to be fed into the transformer network encoder, which consists of multiple alternating multi-head self-attention layers and Multi-Layer-Perceptron (MLP) blocks of two GeLU-activated layers. Layer normalization is applied before each block and there are residual connections after each block. The output from the encoder is then classified through an MLP head [Dosovitskiy et al., 2021].

DeiT. DeiT, Data-Efficient Image Transformer, is an image transformer that was first introduced in [Touvron et al., 2021a]. The model uses a distillation procedure to minimize the amount of data required for training. Knowledge distillation is a training method where a student model learns from labels generated by a strong teacher network. To help the model learn from the output from the teacher a distillation token is used. The token is added to the initial embeddings through self-attention to minimize the loss from the distillation. By distilling knowledge from teachers, DeiT can effectively extract features of the input image and generalize better to unseen data.

CaiT. Class-Attention in Image Transformers, CaiT, is based on the idea that attention should not just be applied to spatial features, but also to class information and consists of two distinct processing stages; the self-attention stage and the class-attention stage. The class-attention stage has two alternating layers: a multi-head class-attention layer and a feed-forward network layer. The role of the class-attention layer is to extract the information from the set of processed patches. The method was evaluated by fine-tuning on small and large datasets, consisting of images from 10 to approximately 8000 classes, and the result showed excellent generalization capabilities [Touvron et al., 2021b].

2.2 Training

2.2.1 Optimization Method

During the training process, the network's weights and biases are randomly initialized and then updated iteratively using a supervised learning algorithm called backpropagation. Gradient descent is an optimization algorithm that uses backpropagation, and adjusts the weights and biases based on the difference between the predicted and actual output, with the goal of minimizing the error or loss function. The optimization process involves calculating the gradient of the loss function with respect to each weight and bias and using this gradient to update the corresponding parameter values. By iteratively adjusting the weights using the optimization algorithm, the network aims to minimize the loss, which in turn improves the accuracy. Deep learning models generally consist of millions of parameters which raises the need for an optimization method that is able to efficiently train the model.[Lu, 2022]

Optimizers use various techniques to update the weights. A common technique that helps optimizers retain speed during the optimization process is called momentum. Momentum incorporates past gradients into the weight-updating process which helps the process retain speed.

Adam, Adaptive Moment Optimization, is a commonly used optimization method that uses “signal-to-noise” normalization and also smoothes the first-order gradient to get momentum to the weight update [Aggarwal, 2018].

2.2.2 Transfer Learning

Pre-trained networks that are already performing well in certain tasks can be used to create new networks to perform other similar tasks. Transfer learning enables existing knowledge from a model trained on data from one domain, to be reused for new tasks with data from another similar domain. A network that can distinguish between different animals can also be suitable for classifying dog breeds. To do so, the final layer, or layers, are removed and a new untrained head is attached. This head can consist of only one dense layer for the new classes, or a few more layers such as fully connected, drop out, batch normalization, and pooling layers. The base model (pre-trained model) then works as a feature-extractor for the new classifying head. This method is particularly effective when a model is trained on a large dataset, and the knowledge from this model can be transferred when using a smaller dataset [Sarkar et al., 2018]. This method of network training is called transfer learning and can be used in many different ways, for example with feature-extraction and fine-tuning mentioned below.

A way of optimizing a pre-trained network for a new task is to start with feature-extraction, and to train the head of the network until it converges. The second step is to fine-tune the network carefully so it doesn’t overtrain, which will be further described below. Improvements can be achieved when the model’s pre-trained weights are adapted to the new data [Keras, 2023].

Feature-Extraction. One way of conducting transfer learning is to ’freeze’ the weights of the base model, i.e. the pre-trained model, and only train the classifying head. In other words, the weights of the base are unchanged and the classifying head uses the output features from the base as input [Sarkar et al., 2018]. In this report, we will refer to this method as feature-extraction.

Fine-Tuning. A pre-trained network can also be fine-tuned, which means ‘unfreezing’ some, or all, of the base model’s layers and re-training them together with the head [Sarkar et al., 2018]. The network is then usually trained with a low learning rate to adapt the pre-trained weights to the new task. In other words, the pre-trained weights are used as a starting point, and the network is re-trained on the new dataset, updating the weights to fit the new task.

2.2.3 Hyperparameters

Hyperparameters are the different settings of the model that are not learned from the data, as opposed to the regular parameters (weights). These are set before the training process and control the behavior and performance of the model. Some of the hyperparameters of the models in this paper include batch size, learning rate, number of epochs, and, for the transformer models, patch size.

2.2.4 Performance Optimization

During the training process of machine learning models, the optimization algorithm adjusts the weights in the model in order to minimize the loss function which takes steps toward finding the optimal set of weights. If the learning rate is too large, the algorithm may not be able to find the optimal set of weights and if the learning rate is too small the algorithm may take too long to converge. Therefore, the learning rate needs to be carefully chosen to ensure that the algorithm efficiently finds a good solution. Employing a learning rate scheduler is a method for optimizing and adapting the learning rate as the training process advances. The purpose of learning rate schedulers is to help the model to find the optimal set of parameters faster and to prevent overfitting [Ghayoumi, 2022].

Step Decay Learning Rate. A step decay learning rate scheduler changes the learning rate in discrete steps at specific epochs during the training. By starting with a high learning rate and reducing the learning rate in steps, the model can more efficiently reach optimal solutions and potentially avoid local minima.

Cosine Annealing. Cosine annealing is a learning rate scheduler that starts with a high learning rate and gradually decreases until it reaches a minimum value. At this point, the learning rate starts to increase again, following a cosine function, until it reaches the maximum value. The learning rate will oscillate between two values and is calculated through the formula:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})(1 + \cos(\frac{T_{cur}}{T_i}\pi)) \tag{2.6}$$

where η_t is the learning rate at iteration t, η_{min} is the minimum learning rate and η_{max} is the maximum learning rate. T_{cur} is the current iteration within the current restart period, $2T_i$ is the number of iterations in the current restart period and cos is the cosine function. By alternating between high and low learning rates, the model can avoid getting stuck in local minima [Loshchilov and Hutter, 2017].

Early Stopping. A way to prevent overfitting when training neural networks is to use early stopping. During training, there might be a point where the validation loss stops improving and the model starts to overfit to the training dataset. To prevent overfitting, early stopping is used to stop the training process before the model starts overfitting by monitoring the validation accuracy or loss and stop the training process if the model has not improved for a specified number of epochs.

2.2.5 Data Manipulation

The amount and the quality of data are one of the biggest impeding factors when working with neural networks. Machine learning models need large amounts of data to learn the underlying patterns of the dataset and to make accurate predictions of unseen data [Yang et al., 2022]. Below are a few different methods that we have used for compensating for too few training samples or class imbalances in the dataset.

Augmentation. Image augmentation has become a necessary part of the success of neural network models for image classification. Augmentation is modifications of the input samples to obtain a larger distribution within the dataset by creating synthetic data. This will in turn lead to better generalization for the model, prevention of overfitting, and a larger training sample group. Various transformations are applied to the original data, in this case, augmentation is done on the images. When working with images, flipping, rotating, cropping, and adjusting contrast and blur are some of the basic techniques [Yang et al., 2022].

Class Weights. It is important to train the model with an equally balanced dataset, which means providing it with approximately the same number of samples from each category. If the dataset is imbalanced, the training model will spend too much time on the majority classes, and too little on the minority classes. This will in turn lead to the model not learning enough from the samples from the minority classes [Developers, n.d.] When working with an imbalanced dataset, one can weigh the loss function during training. This then tells the model to pay more attention to input samples of underrepresented classes [*TensorFlow 2.0 API Docs* n.d.] To counteract the imbalanced class distribution, we will use class weighting, which involves assigning a higher weight to the loss function for under-represented classes.

Selection. Another way of regulating imbalance within the dataset is to downsample and remove samples, from the over-represented classes. However, downsampling means decreasing the amount of data in the dataset. Alternatively, upsampling can be performed by duplicating samples from the under-represented classes, but in this report, we only perform downsampling.

2.2.6 Model Evaluation

In this section, we describe the evaluation metrics used to assess the performance of the models, both when evaluated using the full test set and the downsampled test set.

Class Specific Performance. Precision, recall, and F1 score are class-specific metrics that should be considered in conjunction when evaluating a model's class-specific performance, see [Powers, 2020]. They give indications of how a model is performing on each class in the dataset.

Precision indicates what fraction of the positive predictions for one specific class are correct. It is defined as the number of true positive predictions divided by the total number of positive predictions made by the model. The equation for precision for class c is

$$Precision_c = \frac{TP_c}{TP_c + FP_c} \tag{2.7}$$

where TP_c refers to the number of positive instances that were correctly classified by the model for class c, and false positives FP_c refer to the number of negative instances that were incorrectly classified as positive for class c. Generally, high precision indicates that the model is good at identifying the positive class correctly and that the rate of false positives is low.

Recall, also known as sensitivity, indicates how many of all positive values are predicted positive. It measures the ability of a classification model to correctly identify positive samples and is of use when it is important to not miss any positive samples and is defined as

$$Recall_c = \frac{TP_c}{TP_c + FN_c} \tag{2.8}$$

where FN_c is the number of positive instances that were incorrectly classified as negative for class c.

Class-specific F1 is a trade-off between precision and recall and is defined as the harmonic mean of precision and recall. It can be calculated through

$$F1_c = \frac{2PrecisionRecall}{Precision + Recall} = \frac{2TP_c}{2TP_c + FP_c + FN_c} \tag{2.9}$$

and is useful when the cost of false positives and false negatives is equal.

General Metrics. Classification accuracy is defined as the number of correct predictions divided by the total number of predictions. It measures the percentage of correct predictions made by the model, with all classes included. Accuracy is defined as

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.10}$$

where TP is the true positives, TN the true negatives, FP the false positives, and FN the false negatives [Powers, 2020].

The confusion matrix is a matrix that displays the number of occurrences of the actual and predicted classifications. In this paper, the actual categories will be shown in the rows, and the predicted categories in the columns. The correctly classified instances will appear on the diagonal, from the top left corner to the bottom right corner. An example of a confusion matrix for the three classes used in this paper is presented in Table 2.1 where it can be seen that one image of category 0 is falsely predicted as category 2.

When evaluating using the full, imbalanced test sets, weighted average F1 score and confusion matrices are used. The weighted average F1 score is an evaluation metric that takes both the F1 score of each class and the proportion of the classes

Table 2.1 Example of a confusion matrix for the three classes used in this thesis.

		Predicted		
		category 0	category 1	category 2
Actual	category 0	8	0	1
	category 1	2	6	1
	category 2	0	2	7

Table 2.2 Calculation of weighted average F1 score

Category	Class Specific F1 score	Support	Support Proportion	Weighted Average F1 score
0	0.93	1464	0.376	(0.93 * 0.376) +
1	0.69	420	0.108	(0.69 * 0.108) +
2	0.95	2010	0.516	(0.95 * 0.51)
Total	-	3894	1.0	$\approx$ 0.91

into account. This means that the F1 score of the classes with more samples will be given more weight and the classes with fewer samples less weight. A calculation of the weighted average F1 score of the classes used in this paper can be seen in Table 2.2.

3

The Light Detection Problem

Our project aimed to find an optimized model for detection of light artifacts from oncoming cars during night-time. In the process of model optimization, we explored many areas of a machine learning workflow. We started by modifying and adapting a public dataset, PVDN, described in Section 3.1.1, for our project's specific task, as well as creating an entirely new internal dataset. A significant portion of the project focused on data retrieval, preparation, processing, and analysis. To run experiments, we implemented an end-to-end pipeline, which included data retrieval and processing, model creation, training, and evaluation. To achieve this, we relied heavily on Tensorflow, Keras, and Tensorflow Hub for machine learning functionality, as well as OpenCV for image processing.

3.1 Datasets

Two different datasets were used in this paper; one public (PVDN) and one internal. Both datasets contain images of oncoming cars during night-time from the data collecting car's ego perspective.

A third dataset, ImageNet, has been used to pre-train the models in the project's experiments. ImageNet is a large database containing around 14 million manually annotated images with more than 20000 categories.

Since the main goal of this project was to optimize the models for the internal dataset, the public dataset, PVDN, was used to fine-tune the pre-trained models before further fine-tuning them with the internal dataset.

3.1.1 Public dataset

The public dataset PVDN (Provident Vehicle Detection at Night) consists of 59,746 grayscale images with a resolution of 1280x960, obtained from 346 different sequences from a rural environment during night-time. On average, the sequences contain 167 images each and all sequences contain at least one oncoming vehicle. The dataset is divided into two subsets, where the onboard camera of the test car captured images with two different exposure times, "Day" (short exposure time) and "Night" (long exposure time). Both subsets are divided into train, validation, and test subsets [Ohnemus et al., 2021].

Since there are 7 different categories in the original annotations, all images have been re-annotated with labels matching our categories; 0: "no car nor light artifacts", 1: "light artifacts from a car" and 2: "car or direct light from a car". The dataset has been modified to fit our task and some sequences and images have been removed due to inadequate distribution within the sequence with an over-representation of images labeled with categories 0 and 2. Images from all three classes from the subset "Night" can be seen in Figure 3.1

Due to the low exposure time used for the subset "Day", the images are dark and contain fewer details. The images in the subset "Night" are more similar to the internal dataset and therefore, the images from "Day" needed to be heavily modified when used, for example adjusting the brightness. This is further described in Section 4.2.2, but the results from these modifications can be seen in Figure 3.2. The models were evaluated using both subsets separately, as well as together, to better understand the models' ability to classify images with different characteristics.

The public dataset has been divided into two datasets called PVDN-s, consisting of the images from the subset "Night", and PVDN-l consisting of images from both subsets, "Day" and "Night". The distributions of the classes, sets, subsets, and sequences can be seen in Table 3.1 and in Figure 3.3.

Figure 3.1 Images from PVDN "Night" with classes 0, 1, and 2 (from left to right).

3.1.2 Internal Dataset

The primary objective of this project is to create a model with strong generalization abilities, that is performing well on unseen data and most importantly, on the company's internal data. In order to adapt the model for the company's internal data and evaluate whether the model is performing well in these settings, a new dataset was created. To ensure that the sequences chosen for the internal dataset are appropriate for the task of detecting oncoming cars, sequences recorded during the night and with at least one oncoming car were selected. We were also looking for surroundings with no or few streetlights since the main purpose of the algorithm implemented in

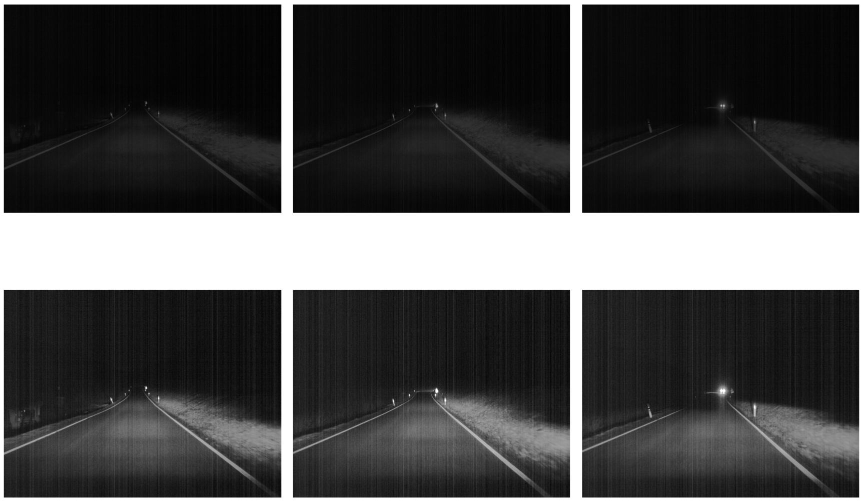

Figure 3.2 Images from PVDN "Day" with classes 0, 1, and 2 (from left to right), before and after increasing the brightness and contrast. In the darker images at the top, it might be hard to visually see any light artifact from the oncoming car.

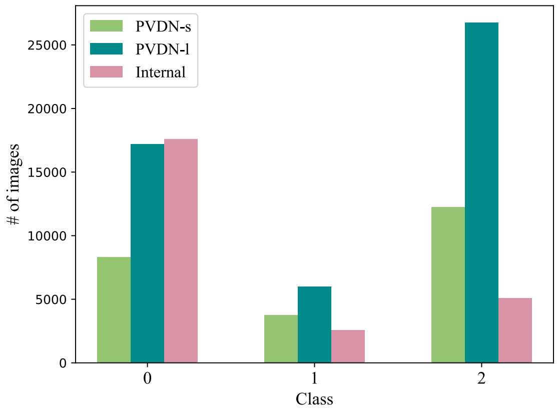

Figure 3.3 A class distribution comparison between PVDN-s, PVDN-l, and the internal dataset.

this project is to turn off the car's high beams. The selected sequences of images then had to be annotated according to our set categories; 0 (no car nor light arti-

Table 3.1 A summary of all images in the PVDN dataset.

Set	Subset	# of sequences	# of 0s	# of 1s	# of 2s	# of images
Train	Day	100	5161	1484	8915	15560
	Night	123	7456	2898	11394	21748
	Total	223	12617	4382	20309	37308
Validation	Day	13	981	443	990	2414
	Night	21	1394	439	1787	3620
	Total	34	2375	882	2777	6034
Test	Day	16	731	307	1651	2689
	Night	23	1464	420	2010	3894
	Total	39	2195	727	3661	6583
Total		296	17187	5991	26747	49925

fact), 1 (light artifact from a car), and 2 (car or direct light from a car). The final dataset consisted of 40 sequences and 25241 images. On average, the sequences in the internal dataset contain 631 images, which is a significantly greater number of images per sequence than in PVDN. The original images were very dark compared to PVDN "Night", and required a lot of adjustments before they were suitable for training. One of the modifications made was increasing the brightness and contrast, and the result from this can be seen in Figure 3.4. The aspect ratio of the images of the internal dataset was 19:9, which also is quite different from PVDN's 4:3.

Table 3.2 The internal dataset's distribution of images and classes.

Set	# of sequences	# of 0s	# of 1s	# of 2s	# of images
Train	19	8072	1315	2822	12209
Validation	10	4519	772	1113	6404
Test	11	4995	494	1139	6628
Total	40	17586	2581	5074	25241

Each sequence only existed in one of the train, validation, and test sets. This was because of the small changes between each frame in the sequences, which would lead to a 'cheating' effect if a model had been trained on a very similar image that it is predicting. The distribution of sequences, subsets, and classes can be seen in Table 3.2. A comparison between PVDN and the internal dataset in terms of class distribution can be seen in Figure 3.3.

Annotation Process. When creating the internal dataset, many images had to be annotated according to the three classes used. To speed up the process, the images that were not yet annotated were fed through a trained model to create predictions or pre-annotations. This was done with a model that was fine-tuned with the PVDN dataset and had achieved a high enough performance (in this case 83% in test accuracy). Afterward, the pre-annotations were manually checked and modified by

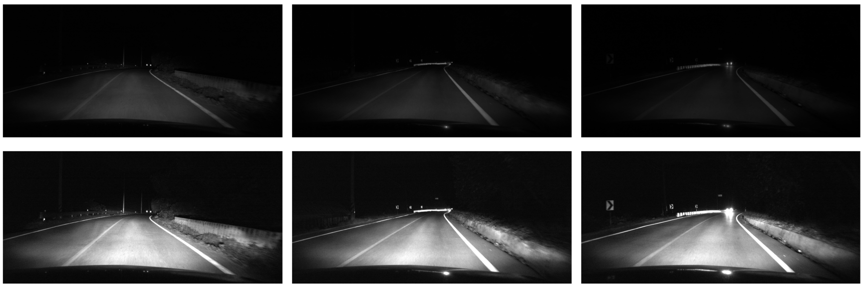

Figure 3.4 Images from the internal dataset with classes 0, 1, and 2 (from left to right), before and after increasing the brightness and contrast. In the left-hand image, no car nor light artifact can be seen. The middle picture shows light artifacts from the oncoming car on the roadside barriers and then the car appears in the image to the right.

us. To make this pre-annotation process as objective and non-biased as possible, the method was controlled by comparing annotations done when using the pre-annotation as a base with annotations done on the same data by people unaware of the predictions. The difference in result between the two methods was negligible, therefore we consider the pre-annotation method to be legitimate.

4

Classification Solution Approach

In this chapter, we will provide a detailed explanation of the models and the training processes used.

4.1 Model Design

The experiments were conducted using six different models, five pre-trained on the ImageNet dataset and one vision transformer with no pre-trained weights. The pre-trained CNNs; DenseNet, EfficientNet, and NASNetMobile were fetched through the Keras Applications API [Chollet et al., 2015–2021], and the transformer models, CaiT and DeiT, were downloaded from [TensorFlow Hub Authors, 2018–2023]. We also developed a custom vision transformer by testing and combining various architectural designs.

4.1.1 Pre-Trained CNNs

The smallest possible architectures were chosen for both NASNet and DenseNet with 5.3M parameters for NASNet Mobile and 8.1M for DenseNet 121, see Table 4.1. This was due to the size of the dataset and the complexity of the designated task for the model, for example, the number of classes. For comparison, EfficientNet B3 was chosen with a total number of parameters of 12.3M, see Table 4.1. The input size for NASNet mobile had to be 224x224 instead of 480x360 like DenseNet and EfficientNet, because of Tensorflow version 2.6.0 bugs.

4.1.2 Pre-Trained Transformers

For DeiT and CaiT, the smallest available architectures, ’tiny’ and ’xxs’ respectively, were picked for the same reasons mentioned above. The DeiT ’tiny’ architecture has a patch size of 16x16 and the input size is 224x224. The patch size chosen is the smallest available but instead entails higher computational cost and possibly better results [Dosovitskiy et al., 2021]. The input size of CaiT 'xxs' is 384x384 [Touvron et al., 2021b].

Below is a table of all pre-trained models used in the experiments.

Table 4.1 All pre-trained models. The parameters are without the added head's/classifier's parameters. Reference name refers to the specific name assigned to each model in the report, which is used throughout the report to refer to each model.

Model	**Ref. name**	**# of Params**	**Input size (x, y)**
DenseNet121	DenseNet	8.1M	480, 360
EfficientNetB3	EfficientNet	12.3M	480, 360
NASNetMobile	NASNet	5.3M	224, 224
DeiT Tiny Distilled	DeiT	5.5M	224, 224
CaiT Xxs24	CaiT	12.0M	384, 384

4.1.3 Classification Head

To perform feature-extraction on all pre-trained models, the last dense layer of the model was removed and a classification head was attached at the top. A combination of dense layers, drop out layers, pooling layers, and batch normalization layers were used depending on the model. Common for all classification heads was the last dense layer containing three nodes for the three classes to be predicted.

4.1.4 Customized Vision Transformer

A Vision Transformer was created with custom architecture to fit this particular task. The best architectures were found by manually tuning the number of layers and heads, size of projection dimension, patch size, input size, and drop out factor. The hyperparameters, such as learning rate, were also extensively tested to find the optimal training setting for the models. The top three models' architectures are shown below in Table 4.2.

4.2 Experiment Design

The experiment pipeline consists of fetching the dataset information (sequences and labels), sorting and selecting the data, loading and pre-processing the images (resize, brightness, multi-channel, etc), loading or creating the model, training the model, and evaluating with the test data. Only the model, training of said model, and dataset were changed between experiments. During the experiments, both the internal and the public datasets were used, and the goal was not to compare them against one another but to improve the model's generalization ability. Three datasets were used for training and evaluating the models; PVDN-s, PVDN-l, and the internal dataset.

Table 4.2 Three Vision Transformers' custom architectures, referred to as ViT 1, 2, and 3. Proj. dim. is the projection dimension. Cosine(η_{max}, η_{min}, T_i) is the cosine annealing settings with η_{max} and η_{min} as learning rate max & min, and T_i is the maximum number of iterations/epochs of the cosine period. Scheduler(η_{max}, η_{min}, $start_{epochs}$) is the learning rate scheduler, with η_{min} and η_{max} as learning rate min & max, and $start_{epochs}$ is the number of epochs with the maximum learning rate.

ViT	1	2	3
# of layers	5	5	12
# of heads	4	4	4
Proj. dim.	20	100	140
Input size	(480, 360)	(480, 360)	(480, 360)
Patch size	(60,80)	(60,80)	(32, 32)
Learning rate	Cosine(1e-5, 1e-9, 50)	Scheduler(1e-5, 1e-9, 3)	1e-5

4.2.1 Dataset Split

The models were trained on three different datasets; a subset of images only from PVDN "Night" called PVDN-s, a subset of images from both PVDN subsets ("Day" and "Night") called PVDN-l, and the internal dataset. PVDN-s only consists of the "Night" set, whose images are more similar to the internal images than the "Day" set. PVDN-l contains images from both "Day" and "Night" sets, which means that there is a difference in image qualities, details, and illumination within the subset. The reason for training on both of these training sets was that even though the internal images are more similar to "Night", it might increase the models' generalization ability to train with a larger set of images. As mentioned in Section 2.5.5, having a large dataset with a broad variety of data often improves performance. The brightness of the images in the internal dataset had to be increased in order to match PVDN's images.

The internal dataset was split into three subsets; train, validation, and test, with 19, 10, and 11 video clips respectively. The split was done so that each set got a fair amount of samples from class 1, as this is the critical class. As mentioned earlier, the PVDN dataset was already split when downloaded.

Due to the imbalance of the number of images of each class in the datasets, the validation set in all three datasets, PVDN-s, PVDN-l, and the internal dataset had to be truncated to match the number of samples of class 1. We decided to keep all images in the training set, and instead weigh the classes during training to get more diverse training data. The evaluation was done with two different test sets, one truncated to balance the number of images of the classes, and one with all images. When comparing the class-specific performance of models, the balanced test set is used. In contrast, the full test set is used for comparing the overall performance of models.

The models' performances were then evaluated on test sets from the three datasets PVDN-s, PVDN-l, and the internal dataset.

4.2.2 Data Pre-Processing

To ensure that our models can be effectively fine-tuned with pre-trained weights, it was necessary to match the input size and channels of our dataset to those used during pre-training. This involved resizing and converting all images from their original grayscale format to three-channel images, with values replicated across all three channels. In addition, for the two transformer models, the pixel values were normalized to a range between -1 and 1 to further align the input with the pre-trained models' data. Pixel values and image size for each model can be seen in Table 4.3. Additionally, the brightness of the images from the internal dataset and from the subset "Day" in PVDN was increased. As mentioned in Section 3.1, examples of images before and after brightness increase can be seen in Figure 3.2 for the PVDN subset "Day" and Figure 3.4 for the internal dataset. This was accomplished by applying a constant multiplication factor to the pixel value with a constant followed by truncation of any values outside of the pixel range. Different constants were used for the two datasets and the constants were chosen to match the subset "Night". Our approach was not solely based on visual inspection, but also involved predicting sequences with various constants and models, selecting the parameter that leads to the highest accuracy.

Table 4.3 All models and their corresponding pixel value range and image sizes that were used in the study.

Model	Pixel values	Input size (x, y)
DenseNet	[0, 255]	480, 360
EfficientNet	[0, 255]	480, 360
NASNet	[0, 255]	224, 224
DeiT	[-1, 1]	224, 224
CaiT	[-1, 1]	384, 384

4.2.3 Data Augmentation

Data augmentation was used as a layer in all model architectures. We developed a customized layer utilizing Keras Layers, that was designed for our datasets' images' properties. It contained transformations for contrast, brightness, and Gaussian blur. The adjustments were applied using a random distribution, which means that the function can skip augmentation entirely, or augment with one augmentation feature, two features, or all features. As the augmentation is part of the models' architecture, all samples from all sets (train, validation, and test) passing through the model will be augmented.

4.2.4 Training Processes

The processes consists of two main training methods; feature-extraction from a pre-trained model (FE) and fine-tuning a pre-trained model (FT). The steps taken during the training processes were combinations of these two in different orders and applied to the datasets used. Two training processes for creating an optimized model for the PVDN dataset were tested and evaluated in parallel, FT and FE-FT. After this, the internal dataset was used to further fine-tune the resulting models trained on PVDN. The steps of each process can be seen below. Table 4.4 shows a summary of all processes and their datasets, and Figure 4.1 shows a flowchart of the steps of each process. The models' performances were evaluated after each step with the test sets mentioned in Section 4.2.1.

During all fine-tuning steps in the processes, all layers of the models were retrained, instead of just unfreezing a few of the last layers. This was done because of the number of models we trained and for efficiency reasons. Only retraining some of the last layers could be tested as future work based on this project.

Since ImageNet and our datasets contain data from such different domains, we will not have a process only consisting of FE steps. If a backbone trained on a dataset from a related domain was used, the processes' steps could have been different.

Process FT (PVDN). A model that is pre-trained on the ImageNet dataset was fine-tuned on the PVDN dataset, both PVDN-s and PVDN-l. The hyperparameters for this step varied depending on the model that was being fine-tuned. This was done on all pre-trained models mentioned in Section 4.1.

Process FT (Internal). This process is connected to the previous one, where the resulting model from FT (PVDN) was fine-tuned with the internal dataset. Only the models trained on PVDN-l was used for this process. The reason for this is that the accuracies from the models trained on PVDN-l exceeded the ones trained on PVDN-s. Here, the hyperparameters were once again changed to keep the structure of the weights and to not overtrain on the new dataset. This process is also referred to as Int-FT throughout this report.

Process FE-FT (PVDN). The process of feature-extraction and fine-tuning is as follows:

1. Feature-extraction from pre-trained model using the PVDN dataset

2. Fine-tuning the resulting model from step 1. with PVDN dataset

Feature-extraction with PVDN datasets (PVDN-s and PVDN-l) was first performed on a pre-trained model. The classifier head's architecture and the hyperparameters varied depending on the model. The next step was to take this model, with the trained head, and fine-tune it on the same dataset. As mentioned in Section 2.2.2, it might be beneficial to train the head first, and then unfreeze the base model's layers and train the model again. To validate this, we evaluated both methods, only fine-tuning and feature-extraction then fine-tuning, to see whether training the classifier head first makes any difference.

Process FE-FT (Internal). As in process Int-FT, this process builds upon the previously generated model from FE-FT (PVDN), trained on PVDN-1, and will be referred to as Int-FE-FT. The model was fine-tuned again with the internal dataset and the hyperparameters were also modified to preserve the structure of the pre-trained weights and not overtrain when fine-tuning. The training process in Int-FT is identical to Int-FE-FT, but the base model that is fine-tuned is not identical in the different approaches.

Table 4.4 The training processes, their reference name (Ref.), and the respective training datasets. 'Train' refers to the training dataset during the experiments in this project, and 'PT mod.' is the pre-trained model the process is based on. 'Train PT mod.' is the dataset that the pre-trained model that is being feature-extracted or fine-tuned is trained on.

Process	Ref.	Train	PT mod.	Train PT mod.
FT (PVDN)	FT	PVDN	-	ImageNet
FT	Int-FT	Internal	FT (PVDN)	PVDN
FE-FT (PVDN)	FE-FT	PVDN	-	ImageNet
FE-FT	Int-FE-FT	Internal	FE-FT (PVDN)	PVDN

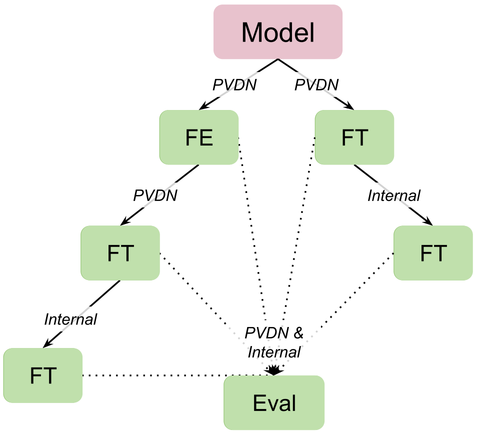

Figure 4.1 The different paths for training the models. On the left-hand side, the first two blocks represent FE-FT with PVDN, with the Int-FE-FT block located at the bottom. FT with PVDN is represented on the right-hand side with Int-FT at the bottom. The majority of all models generated were evaluated with PVDN and the internal dataset, which can be seen in the middle block 'Eval'.

4.2.5 Hyperparameters

The hyperparameters chosen for the training experiments varied depending on the model and method. For example, some models required a higher learning rate, whilst other models were very sensitive to the learning rate. To control the learning rate, cosine annealing, and a step decay scheduler were used during most of the trainings. The final hyperparameters for all trainings were decided on along the way, during the development of the models. The number of epochs was set to 100, with early stopping using 10 to 15 epochs in patience, and the batch size was set to 32. These were common for all training sessions, except for the last step of each process, the final fine-tuning, where the number of epochs was decreased to 20.

4.3 Resources

The end-to-end pipeline was conducted in Visual Studios Code using Python and Shell. The implementation was done using frameworks such as TensorFlow, Keras, TensorFlow Hub, Open CV, and Numpy. An internal computation cluster with several CPUs and GPUs was used for the model trainings.

5

Results

This chapter presents the results of the models, evaluated using both the downsampled, balanced, datasets and the full datasets.

5.1 Balanced Test Sets

When training the models with the dataset PVDN-l, using the FE-FT process of feature-extraction and then fine-tuning, the best-performing model according to the test accuracy and F1 score was DenseNet. It achieved an accuracy of 88.10% and 0.83 in F1 score for class 1 when evaluating with the test set from PVDN-s. EfficientNet followed with an accuracy of 85.79% and an F1 score of 0.81. All models' results for the processes, FE, FT, FE-FT, using the training set from PVDN-l and the test set from PVDN-s can be seen in Figure **??**.

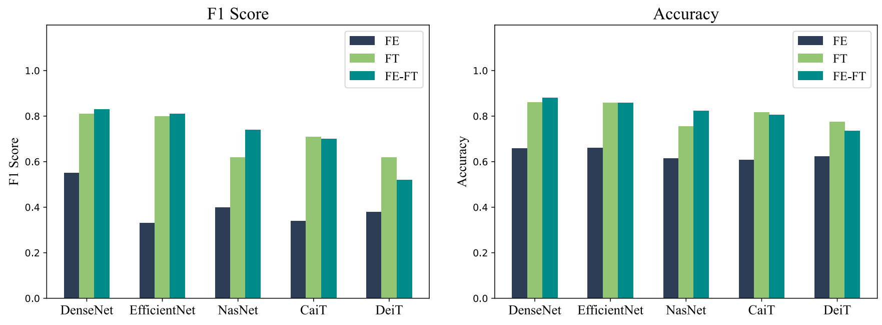

Figure 5.1 Performances of models trained on the training set from PVDN-l with different training processes, FT and FE-FT. The results are from evaluating with the test set from PVDN-s, where the F1 score is for class 1 and the accuracy is for all classes.

The results in Figure **??** show that the convolutional networks perform well for the task when trained on the training set from PVDN-l. When only focusing on FE-

FT results, all convolutional neural networks perform better than the transformer-based networks, both in terms of accuracy and F1 score. The FE-FT also seems to be the better process for the convolutional-based networks. The opposite applies to the transformer networks, where it's beneficial to not pre-train the head in advance, and only fine-tune the models. One reason might be that the trained head gets overfitted to the train data due to being trained twice, and the model then performs poorly on the test data. A transformer usually requires a large amount of data to perform well and PVDN might not be large enough for a transformer network to learn from [Dosovitskiy et al., 2021].

The same evaluation, as in Figure **??**, was done for the models fine-tuned with the internal dataset i.e. the processes Int-FE-FT and INT-FT. The two processes, Int-FT and INT-FE-FT, are compared in Figure 5.2. EfficientNet trained on a model with Int-FE-FT process achieved the best F1 score for class 1, 0.66, and the best accuracy, 76.72%. The evaluation was done with the balanced test set from the internal dataset.

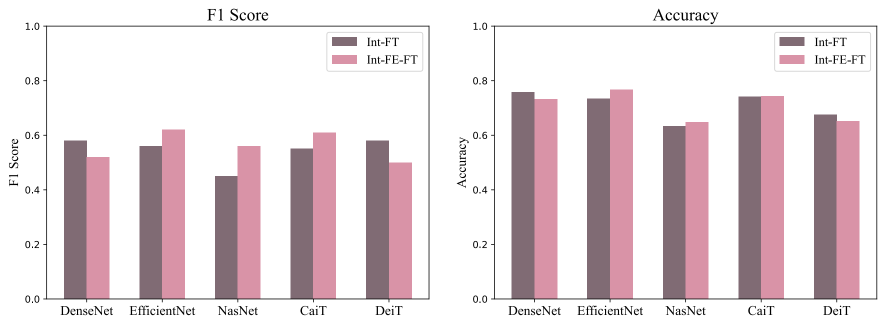

Figure 5.2 Results from models fine-tuned using the internal dataset with different training processes, Int-FT and Int-FE-FT. The F1 score is for class 1, and the accuracy is for all classes. The models were evaluated with the internal test set.

The models trained with Int-FE-FT and Int-FT showed less variation in F1 scores than those shown in Figure 5.1, trained on the train set from PVDN-l and evaluated with the test set from PVDN-s. The range of the F1 scores for the models trained with the internal dataset was 0.17, whereas for PVDN, the range of the F1 scores was 0.31. It seemed like the performance of the pre-trained models, the FT and FE-FT models, had little impact on the performance of the models fine-tuned with the internal dataset, as the transformer models did not perform as poorly compared to the convolutional networks as they did when trained on the PVDN dataset. One potential explanation for this could be that the transformer models were more generalized and better at predicting unseen data than the convolutional. This is also implicated in Table 5.2, where the CaiT model trained on PVDN-l got an accuracy of 70.92% when evaluated with the internal test set. This is only approximately 3.5 percentage points lower than the one trained on the internal dataset.

To compare the performance difference between training on PVDN and the internal dataset, we evaluated all models solely trained on PVDN-l using the internal test set. The resulting F1 score for class 1 and overall accuracy when using PVDN-l with FT and FE-FT, and internal with Int-FT and Int-FE-FT are shown in Figure 5.3.

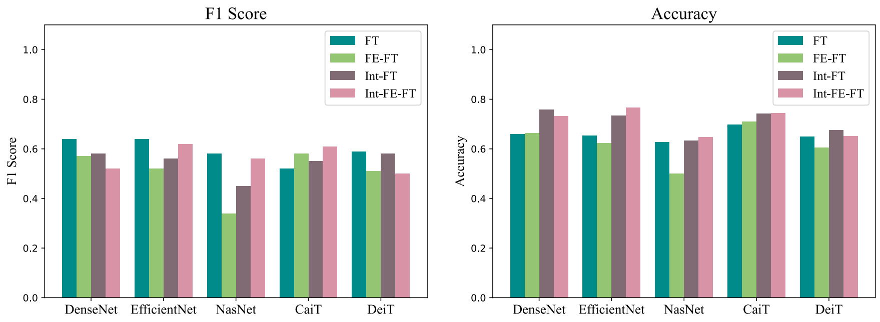

Figure 5.3 Performances of fine-tuned models trained on PVDN-l and the internal dataset using FT, FE-FT, Int-FT, and Int-FE-FT processes, evaluated on the internal test set. The F1 score is for class 1, and the accuracy is calculated for all classes.

Figure 5.4 presents the results of training the models using the FE-FT process and evaluating them when trained using different training datasets. The accuracy and F1 score for class 1 are displayed for all models. The test set from PVDN-s was used to evaluate the models trained on both PVDN-s and PVDN-l datasets, whereas the internal test set was used for evaluating the models trained on the internal dataset. It can be seen that all models, except DeiT, improve when training on the larger dataset, PVDN-l.

Figures 5.5 and 5.6 show the results of the highest-performing models. The models have been evaluated using PVDN-s and the internal test set, however, they were trained using different training processes and datasets. According to Figure 5.6, the models trained on PVDN (FT and FE-FT) showed a similar F1 result for predicting classes 0 and 2, and the difference between them lies in the F1 score for class 1. The difference in F1 score for class 1 is a significant factor that accounts for the variation in accuracy between the models, with higher F1 scores leading to greater accuracy and vice versa. When examining the plot on the right-hand side of Figure 5.6, which displays the results of models evaluated on the internal test set, it can be seen that the F1 score for class 1 remains consistent across all models. However, the F1 scores for categories 0 and 2 are lower for the NASNet and DeiT models, leading to a decrease in accuracy compared to when they were trained on

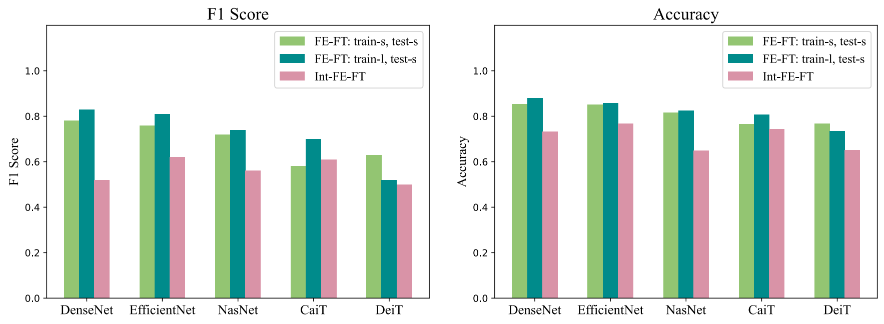

Figure 5.4 Comparison of the models' performances when training on different datasets with the FE-FT and Int-FE-FT processes, and evaluating with the respective test set. The light green bar is for the models trained and evaluated with small PVDN (PVDN-s). The dark green bar, in the middle, is for the models trained on the larger PVDN (PVDN-l) and evaluated with the test set from PVDN-s. The pink bar shows the result from the models being trained and evaluated with the internal dataset.

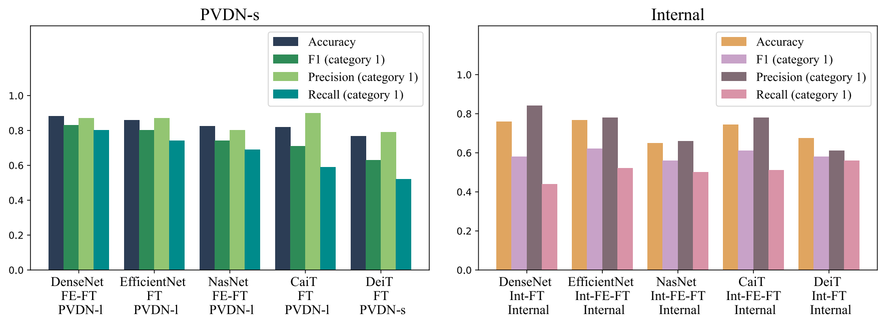

Figure 5.5 Comparison of accuracy, F1-score for category 1, precision for category 1, and recall for category 1. The results shown in the figure correspond to the process generating the highest accuracy for each model evaluated with PVDN-s and the internal test set.

the PVDN dataset. However, the two datasets, PVDN, and the internal dataset are different both when it comes to the number of images, the number of sequences, and the images' properties. This might be a reason for the difference in performance between the models trained and evaluated on PVDN and the models trained and evaluated on the internal dataset.

An unexpected result is that the models trained and evaluated on the internal dataset are better at predicting class 2 than class 0, even though there is an overrepresentation of images with class 0 in the dataset. When visually inspecting the

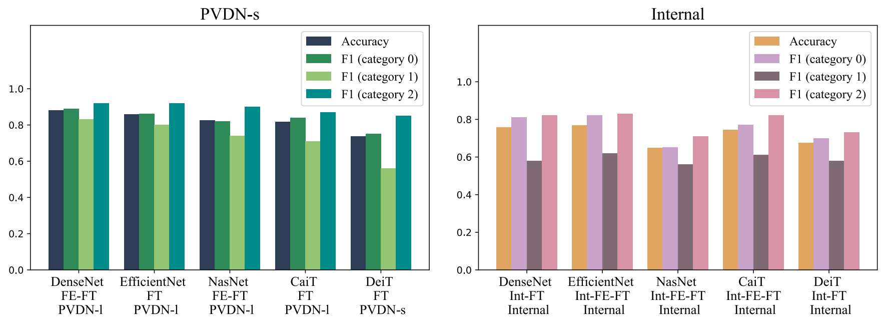

Figure 5.6 Comparison of accuracy and F1-score. The results shown in the figure correspond to the process generating the highest accuracy for each model evaluated with PVDN-s and the internal test dataset.

images from both datasets, we noticed that the indirect and direct light from the oncoming car's headlights in the images from the PVDN subset "Night" is spreading more and affects a larger area in the image. In the internal dataset and in the PVDN subset "Day", the glare is, in many cases, contained in a small area which might lead to difficulties for the models to distinguish between the classes in these datasets.

Table 5.1 and 5.2 show the performance of the top 10 performing models in terms of accuracy, evaluated on both the balanced PVDN-s test set and the balanced internal test set.

Table 5.1 Top 10 performing models, based on their accuracy on the test set from PVDN-s, trained using different methods and displaying their corresponding accuracy and performance metrics.

Model	**Process**	**Train**	**Acc.**	**F1**	**Prec.**	**Recall**
DenseNet	FE-FT	PVDN-l	0.881	0.87	0.83	0.8
DenseNet	FT	PVDN-l	0.862	0.87	0.81	0.75
EfficientNet	FT	PVDN-l	0.86	0.87	0.8	0.74
EfficientNet	FE-FT	PVDN-l	0.858	0.83	0.81	0.79
DenseNet	FE-FT	PVDN-s	0.854	0.84	0.78	0.73
EfficientNet	FE-FT	PVDN-s	0.852	0.87	0.76	0.67
EfficientNet	FT	PVDN-s	0.833	0.84	0.76	0.68
NasNet	FE-FT	PVDN-l	0.825	0.8	0.74	0.69
DenseNet	FT	PVDN-s	0.822	0.83	0.73	0.65
CaiT	FT	PVDN-l	0.818	0.9	0.71	0.59

The results for the top 3 performing ViTs with custom architectures (not pre-trained on any dataset) are shown in Table 5.3. These models were only trained and evaluated with PVDN-s.

Table 5.2 Top 10 performing models, based on their accuracy on the internal test set, trained using different methods and displaying their corresponding accuracy and performance metrics

Model	Process	Train	Acc.	F1	Prec.	Recall
EfficientNet	Int-FE-FT	Int	0.767	0.78	0.62	0.52
DenseNet	Int-FT	Int	0.758	0.84	0.58	0.44
CaiT	Int-FE-FT	Int	0.744	0.78	0.61	0.51
CaiT	Int-FT	Int	0.742	0.82	0.55	0.41
EfficientNet	Int-FT	Int	0.734	0.72	0.56	0.46
DenseNet	Int-FE-FT	Int	0.731	0.85	0.52	0.37
CaiT	FE-FT	PVDN-l	0.709	0.67	0.58	0.51
CaiT	FT	PVDN-l	0.698	0.72	0.52	0.41
DenseNet	FT	PVDN-s	0.696	0.84	0.66	0.54
CaiT	FT	PVDN-s	0.688	0.56	0.59	0.63

Table 5.3 Accuracy and F1 score for all classes, 0, 1, and 2. The ViTs have an architecture created by us, and these are the top 3 performing models in terms of test accuracy.

ViT	Accuracy	F1 - class 0	F1 - class 1	F1 - class 2
1	0.621	0.61	0.46	0.75
2	0.611	0.63	0.37	0.77
3	0.601	0.64	0.38	0.7

NASNet and DeiT achieved lower performance compared to most of the other models tested. It's worth noting that they had the smallest image input size, 224x224, which may have led to a loss of important details in the images. This is supported by the correlation observed in Figure 5.7, which suggests a possible relationship between image input size and performance. However, further data is required to draw a definitive conclusion about this correlation. Additionally, the relatively low number of parameters in their architectures, 5.3 million and 5.5 million respectively, could have contributed to their lower performance. For context, this is approximately 3 million parameters fewer than the DenseNet, which achieved high performance on the same task. However, our analysis did not find a clear correlation between the number of parameters and performance, as shown in Figure 5.8.

While the internal dataset used for fine-tuning and testing contains a large number of images, it has a limited number of sequences. This means that many images in the dataset are quite similar, and may not provide sufficient diversity for the models' training. This lack of diversity might explain why the transformer models performed better when fine-tuned on the internal dataset. The attention mechanism in the transformer models enables them to focus on different aspects of the images, which might help capture variations between similar images in the dataset.

The results indicate that both EfficientNet and DenseNet are viable models for

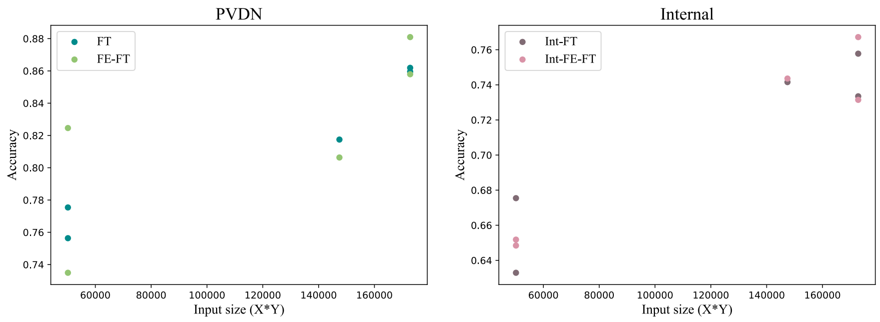

Figure 5.7 Accuracy for the processes FT and FE-FT to the left and INT-FT and INT-FE-FT to the right, plotted against the models' input size. The input size is calculated through the number of pixels in the x-plane multiplied by the number of pixels in the y-plane.

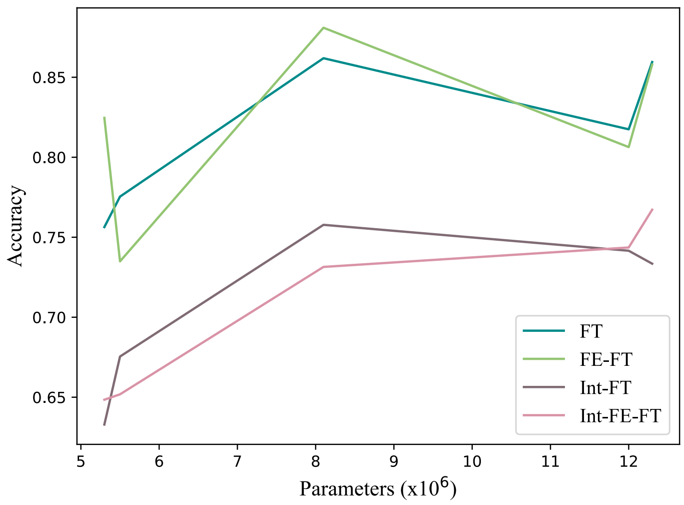

Figure 5.8 Accuracy for the processes FT, FE-FT, INT-FT, and INT-FE-FT plotted against the model parameters in millions. The models included for FT and FE-FT are trained with PVDN-l and tested with PVDN-s. For Int-FT and Int-FE-FT, the models were evaluated with the test set from the internal dataset.

this task. However, CaiT and DeiT also demonstrated strong performance when fine-tuned on the internal dataset, suggesting their potential suitability for this task. Since the models DeiT and NASNet did not achieve as good results as the models with larger input sizes, we believe that all models could benefit from using input images with higher resolutions.

The confusion matrices of the best-performing models, which are displayed in Figure 5.5, can be found in Table 5.4 and Table 5.5. Table 5.4 shows the confusion matrices obtained using the public test set, PVDN-s, while Table 5.5 shows the confusion matrices obtained from the internal test set evaluation.

Table 5.4 Confusion matrices of the best performing models evaluated using the downsampled test set PVDN-s, based on the process and training dataset resulting in the highest weighted average F1 score.

Model	**Process**	**Train set**	**Confusion Matrix** Test set: PVDN-s
DenseNet	FE-FT	PVDN-l	$\begin{array}{cc\|c\|c\|c\|} & & \multicolumn{3}{c}{\text{Predicted}} \\ & & 0 & 1 & 2 \\ \hline \text{Actual} & 0 & 383 & 29 & 8 \\ & 1 & 50 & 334 & 36 \\ & 2 & 5 & 22 & 393 \\ \hline \end{array}$
EfficientNet	FE-FT	PVDN-l	$\begin{array}{cc\|c\|c\|c\|} & & \multicolumn{3}{c}{\text{Predicted}} \\ & & 0 & 1 & 2 \\ \hline \text{Actual} & 0 & 388 & 26 & 6 \\ & 1 & 67 & 331 & 22 \\ & 2 & 14 & 44 & 362 \\ \hline \end{array}$
NASNet	FE-FT	PVDN-l	$\begin{array}{cc\|c\|c\|c\|} & & \multicolumn{3}{c}{\text{Predicted}} \\ & & 0 & 1 & 2 \\ \hline \text{Actual} & 0 & 362 & 46 & 12 \\ & 1 & 91 & 288 & 41 \\ & 2 & 6 & 25 & 389 \\ \hline \end{array}$
CaiT	FT	PVDN-l	$\begin{array}{cc\|c\|c\|c\|} & & \multicolumn{3}{c}{\text{Predicted}} \\ & & 0 & 1 & 2 \\ \hline \text{Actual} & 0 & 379 & 19 & 22 \\ & 1 & 95 & 249 & 76 \\ & 2 & 8 & 10 & 402 \\ \hline \end{array}$
DeiT	FE-FT	PVDN-s	$\begin{array}{cc\|c\|c\|c\|} & & \multicolumn{3}{c}{\text{Predicted}} \\ & & 0 & 1 & 2 \\ \hline \text{Actual} & 0 & 393 & 22 & 5 \\ & 1 & 217 & 161 & 42 \\ & 2 & 29 & 19 & 372 \\ \hline \end{array}$

The confusion matrices in Table 5.4 show that the balanced PVDN-s test set

Table 5.5 Confusion matrices of the best performing models evaluated using the downsampled internal test set, based on the process and training dataset resulting in the highest weighted average F1 score.

Model	**Process**	**Train set**		**Confusion Matrix**		
				Test set: Internal		
				Predicted		
			Actual	0	1	2
DenseNet	Int-FT	Internal	0	435	32	27
			1	127	217	150
			2	13	10	471
				Predicted		
			Actual	0	1	2
EfficientNet	Int-FE-FT	Internal	0	410	62	22
			1	87	258	149
			2	13	12	469
				Predicted		
			Actual	0	1	2
NASNet	Int-FE-FT	Internal	0	306	101	87
			1	88	245	161
			2	56	28	410
				Predicted		
			Actual	0	1	2
CaiT	Int-FT	Internal	0	437	34	23
			1	160	202	132
			2	24	10	460
				Predicted		
			Actual	0	1	2
DeiT	Int-FE-FT	Internal	0	353	60	81
			1	147	192	155
			2	49	24	421

exhibits an over-representation of images labeled as category 1 being predicted as category 0. The opposite is true for the convolutional neural networks when evaluating the balanced internal test set where more images of category 1 are classed as category 2. The internal images are darker than the images in the test set PVDN-s, images from the subset "Night", and the brightness and contrast of the internal images were therefore increased. Too much contrast and brightness might lead to smaller visual differences between images of category 1 and category 2 and could be one reason why more images of category 1 are classed as category 2 for the internal dataset.

5.2 Imbalanced Test Sets

To compare the overall performance of models, the models have been evaluated using the full, imbalanced, test sets. The result of the highest performing models in terms of weighted average of the F1 scores are presented in Table 5.6 and 5.7 using the PVDN-s test set and the internal test set respectively. The models have been trained using different training processes. The corresponding confusion matrices can be seen in Table 5.8 and 5.9.

Table 5.6 Weighted average F1 score for the best-performing process and train set for each model. The result is for evaluation with the PVDN-s test set.

Model	**Process**	**Train set**	**F1 - weighted average Test set: PVDN-s**
DenseNet	FE-FT	PVDN-l	0.91
EfficientNet	FT	PVDN-l	0.90
NASNet	FE-FT	PVDN-s	0.88
CaiT	FT	PVDN-l	0.89
DeiT	FT	PVDN-l	0.86

Table 5.7 Weighted average F1 score for the best-performing process and train set for each model. The result is for evaluation with the internal test set.

Model	**Process**	**Train set**	**F1 - weighted average Test set: Int**
DenseNet	Int-FT	Internal	0.88
EfficientNet	Int-FE-FT	Internal	0.85
NASNet	Int-FE-FT	Internal	0.72
CaiT	Int-FT	Internal	0.87
DeiT	FT/Int-FT	PVDN-l/Internal	0.79

The weighted average F1 scores of the top performing models, presented in Table 5.6 and 5.7, indicate that DenseNet is a highly effective deep learning model, with average weighted F1 scores of 0.91 and 0.88 respectively, that performs well for the task. Notably, the weighted average F1 score of DenseNet when evaluating using the internal dataset is only slightly lower than the score when evaluating using the PVDN test set, even though the internal dataset contains very few sequences. In contrast, NASNet, trained using Int-FE-FT, achieved the lowest score of 0.72 on the internal test set, and its performance was notably worse than when evaluated on the PVDN test set, with a decrease of 0.16 in the F1 score. This suggests that the 19

Table 5.8 Confusion matrices of the best performing models evaluated using the full test set PVDN-s, based on the process and training dataset resulting in the highest weighted average F1 score.

Model	**Process**	**Train set**	**Confusion Matrix**			
			Test set: PVDN-s			
				Predicted		
			Actual	0	1	2
DenseNet	FE-FT	PVDN-l	0	1322	100	42
			1	50	334	36
			2	19	108	1883
EfficientNet	FT	PVDN-l	0	1360	80	24
			1	96	309	15
			2	84	107	1819
NASNet	FE-FT	PVDN-s	0	1301	121	42
			1	107	271	42
			2	65	126	1819
CaiT	FT	PVDN-l	0	1306	58	100
			1	95	249	76
			2	64	42	1904
DeiT	FT	PVDN-l	0	1344	56	64
			1	165	205	50
			2	106	79	1825

sequences in the internal training set may not be sufficient for NASNet. The transformer model CaiT, trained using the Int-FT process, performed second best when evaluating on the internal test set, and achieved only a 0.01 lower weighted average F1 score than the best model, DenseNet. When evaluating CaiT, trained with the process Int-FE-FT, on the PVDN-s test set, the model achieved a higher weighted F1 score than when evaluating on the internal test set, this is shown in Table 7.2 in the Appendix. This suggests that CaiT may have the ability to generalize well and

Table 5.9 Confusion matrices of the best performing models evaluated using the full internal test set, based on the process and training dataset resulting in the highest weighted average F1 score.

Model	**Process**	**Train set**		**Confusion Matrix**		
				Test set: Internal		
				Predicted		
			Actual	0	1	2
DenseNet	Int-FT	Internal	0	4496	251	248
			1	127	217	150
			2	28	21	1090
EfficientNet	Int-FE-FT	Internal	0	4208	582	205
			1	87	258	149
			2	30	27	1082
NASNet	Int-FE-FT	Internal	0	3347	924	724
			1	88	245	161
			2	117	64	958
CaiT	Int-FT	Internal	0	4476	295	224
			1	160	202	132
			2	46	29	1064
DeiT	Int-FT	Internal	0	3952	779	264
			1	142	276	76
			2	126	176	837
DeiT	FT	PVDN-1	0	4051	793	151
			1	138	284	72
			2	255	188	696

perform effectively even with a smaller training set. The performance of the DeiT model did not improve when fine-tuned with the internal dataset, as evidenced by the fact that its weighted average F1 score on the internal test set remained the same as when it was only trained on the external dataset, PVDN. DeiT trained with PVDN-l is the model that correctly predicted the highest number of images of category 1 and is second best at correctly predicting class 1 when trained with the internal dataset. However, this model had a high number of falsely predicted images of categories 0 and 2, predicted as category 1. The second best model when evaluating the downsampled internal test set was EfficientNet, which was also the second best at correctly predicting images of category 1 when using the full internal test set. However, the confusion matrix shows that the model falsely predicted many images of category 0 as category 1, which is one reason for achieving only the third-best weighted average F1 score. These findings highlight the importance of evaluating using both a balanced test set and the full test sets, in order to get a comprehensive understanding of model performance.

6

Future Work & Conclusion

This chapter suggests possible directions for future work and summarizes the main findings.

6.1 Future Work

When designing a neural network model a common challenge is to ensure that the model learns from relevant data and not overfits on specific details. One potential issue that can contribute to overfitting is if the model learns to classify based on information that is not actually relevant to the classification task. In our case, one such issue might be that the model learns to classify based on the state of the own car's high beam. For example, it might learn that there is usually no other car present when the own car's high beam is turned on and the surrounding is illuminated. However, this is not a valid factor since the task is to automatically turn off the high beam and should not depend on the driver's action. During the developing phase, we noticed that the models incorrectly predicted the presence of a car, or light artifacts from a car, in scenes where there was no such object or artifact when the high beam was turned off. In other words, the model was detecting the presence of a car due to the absence of the high beam, even though there was no actual car or artifact present in the scene. By alternating when the high beam is turned off when collecting new data, the impact of this bias could be reduced. This would lead to data with images of all classes both with the high beam on and off, which would help minimize the influence of the high beam. Another way to reduce the effect of whether the surrounding is illuminated or not is to use an object detection method or a segmentation method that only focuses on a small part of the image.

The augmentation layer, mentioned in Section 4.2.3, was part of all models' architectures and was therefore applied on all sets; train, validation, and test. This means that when predicting an input image, the image will be augmented with the same probability as for an input from the train set. Augmenting the validation and test set is not ideal, and should only be done for the training set to diversify the data and the result might have been affected by this.

Since there was no annotated internal data for this project, annotations had to be done by us. Due to time constraints, only a selected number of sequences were chosen for the internal dataset. Figure 5.4 indicates that a larger dataset, PVDN-l, results in better model performance compared to a smaller dataset, PVDN-s. Based on this observation, we believe that more annotated data from a larger amount of sequences would lead to improvements in the models' performances. To expand on this project, collecting and annotating more data would be a crucial part of developing an algorithm with reliable results.

One important consideration when expanding the dataset in the future is the number of classes. Determining whether to turn off or keep the high beam on might be easier with additional classes that describe the situation. For example, the two classes where a car, or light artifact from a car, is present could be further divided into classes that describe if the high beam should be turned off or not.

Using a sliding average when deactivating the high beam or using a probability threshold for the images predicted as class 1 are alternative solutions to determine if the high beam should be kept on or not.

To reach the models' full potential, a hyperparameter grid search should be performed on the models. The values chosen for hyperparameters such as learning rate, batch size, and the number of epochs can greatly affect the performance of the model. By testing the model with various combinations of hyperparameters the model can reach its full potential, resulting in better accuracy and generalization. We were, unfortunately, unable to do this due to constraints in both time and computer resources. We noticed that some hyperparameters had a more significant impact on the transformer models compared to the convolutional neural networks. Based on this, we believe that a hyperparameter grid search would benefit the performance of transformers and lead to improved results.

Several hyperparameters, as well as architectural structures such as the classification head, for the models, were determined based on insights obtained from the training process and its results. Many of these decisions were made along the way as more information became available during the trainings. Some of the training sessions took over a day to run, it was therefore difficult to plan ahead of time. Even though we had access to a number of GPUs, the computational resource was a limiting factor in this project. Another restriction due to constraints in both time and computer resources is the number of models we were able to train and the input image size of each model. Increasing the image size could lead to higher model performance as it allows the model to capture more detailed information, as mentioned earlier. This is something that should be considered for future work, as well as exploring more models.

The datasets consist of sequences extracted from a video clip, and a way of taking advantage of this would be to explore the efficacy of models taking a sequence as input data. A model like this could be classifying sequences by detecting differences between the images within the sequence. Manually annotating the internal dataset was a challenging task when examining the images individually. Annotations were instead made by looking at multiple frames, hence using models that take a sequence as an input, such as 3D Convolutional Neural Networks and Video Vision Transformers, might be a suitable solution for this task.

6.2 Conclusion

The primary goal of this thesis was to investigate the possibility to detect oncoming vehicles during night-time before they become visible. This has been done by evaluating three pre-trained convolutional neural networks and two pre-trained vision transformers by training them using one public dataset and one internal dataset. We utilized two different transfer learning techniques, fine-tuning, and feature-extraction combined with fine-tuning, to leverage the knowledge from the pre-trained models. Consequently, we conducted two training processes for each dataset.

Among the models we evaluated, DenseNet trained using feature-extraction and fine-tuning yielded the best results when evaluated on the balanced public dataset, PVDN-s, with an accuracy of 88.1%. DenseNet trained using only fine-tuning, and EfficientNet trained using feature-extraction and fine-tuning were the top-performing models when evaluated on the balanced internal test set with accuracies of 75.78% and 76.72% respectively. Interestingly, the vision transformer, CaiT, achieved an accuracy of 70.92% when predicting the balanced internal test set and being trained solely on the public dataset. When evaluating the imbalanced test sets, DenseNet was the top-performing model for both the public and the internal test sets, with a weighted average F1 score of 0.91 and 0.88 respectively. CaiT achieved the second-best weighted average F1 score, 0.87, when evaluating the full internal test set.

Overall, our project findings demonstrate that different models, both transformer-based and convolution-based models, are capable of detecting cars during night-time before they become visible, by training them using a holistic image classification approach to detect the light artifacts from the cars' headlights.

7

Appendix

Table 7.1 Results from all model trainings. ’Acc.’ is the test accuracy and ’Prec.’ is the precision of class 1. The F1 score and recall are also calculated for class 1.

Process	Model	Train	Test	Acc.	Prec.	F1	Recall
DenseNet	FE	PVDN-s	PVDN-s	0.655	0.63	0.38	0.27
DenseNet	FE	PVDN-l	PVDN-s	0.658	0.57	0.55	0.53
DenseNet	FE	PVDN-l	PVDN-l	0.62	0.54	0.44	0.37
EfficientNet	FE	PVDN-s	PVDN-s	0.709	0.66	0.52	0.43
EfficientNet	FE	PVDN-l	PVDN-s	0.66	0.66	0.33	0.22
EfficientNet	FE	PVDN-l	PVDN-l	0.63	0.58	0.32	0.22
NasNet	FE	PVDN-s	PVDN-s	0.618	0.61	0.36	0.26
NasNet	FE	PVDN-l	PVDN-s	0.614	0.55	0.40	0.31
NasNet	FE	PVDN-l	PVDN-l	0.605	0.56	0.36	0.27
DeiT	FE	PVDN-s	PVDN-s	0.649	0.63	0.40	0.29
DeiT	FE	PVDN-l	PVDN-s	0.624	0.63	0.38	0.27
DeiT	FE	PVDN-l	PVDN-l	0.607	0.59	0.30	0.20
CaiT	FE	PVDN-s	PVDN-s	0.617	0.56	0.32	0.23
CaiT	FE	PVDN-l	PVDN-s	0.607	0.55	0.34	0.25
CaiT	FE	PVDN-l	PVDN-l	0.589	0.56	0.32	0.22
DenseNet	FE-FT	PVDN-s	PVDN-s	0.854	0.84	0.78	0.73
DenseNet	FE-FT	PVDN-s	Int	0.659	0.74	0.49	0.37
DenseNet	FE-FT	PVDN-l	PVDN-s	0.881	0.87	0.83	0.80
DenseNet	FE-FT	PVDN-l	PVDN-l	0.812	0.83	0.71	0.62
DenseNet	FE-FT	PVDN-l	Int	0.664	0.71	0.57	0.47
EfficientNet	FE-FT	PVDN-s	PVDN-s	0.852	0.87	0.76	0.67
EfficientNet	FE-FT	PVDN-s	Int	0.563	0.61	0.52	0.46
EfficientNet	FE-FT	PVDN-l	PVDN-s	0.858	0.83	0.81	0.79
EfficientNet	FE-FT	PVDN-l	PVDN-l	0.796	0.77	0.69	0.62
EfficientNet	FE-FT	PVDN-l	Int	0.624	0.70	0.52	0.41
NasNet	FE-FT	PVDN-s	PVDN-s	0.815	0.83	0.72	0.65

NasNet	FE-FT	PVDN-s	Int	0.484	0.65	0.39	0.28
NasNet	FE-FT	PVDN-l	PVDN-s	0.825	0.80	0.74	0.69
NasNet	FE-FT	PVDN-l	PVDN-l	0.74	0.75	0.60	0.49
NasNet	FE-FT	PVDN-l	Int	0.501	0.66	0.34	0.23
DeiT	FE-FT	PVDN-s	PVDN-s	0.768	0.79	0.63	0.52
DeiT	FE-FT	PVDN-s	Int	0.624	0.50	0.58	0.68
DeiT	FE-FT	PVDN-l	PVDN-s	0.735	0.80	0.52	0.38
DeiT	FE-FT	PVDN-l	PVDN-l	0.689	0.75	0.40	0.28
DeiT	FE-FT	PVDN-l	Int	0.605	0.55	0.51	0.47
CaiT	FE-FT	PVDN-s	PVDN-s	0.765	0.87	0.58	0.44
CaiT	FE-FT	PVDN-s	Int	0.687	0.61	0.57	0.54
CaiT	FE-FT	PVDN-l	PVDN-s	0.806	0.87	0.70	0.58
CaiT	FE-FT	PVDN-l	PVDN-l	0.731	0.86	0.53	0.38
CaiT	FE-FT	PVDN-l	Int	0.709	0.67	0.58	0.51
DenseNet	FT	PVDN-s	PVDN-s	0.822	0.83	0.73	0.65
DenseNet	FT	PVDN-s	Int	0.696	0.84	0.66	0.54
DenseNet	FT	PVDN-l	PVDN-s	0.862	0.87	0.81	0.75
DenseNet	FT	PVDN-l	PVDN-l	0.816	0.84	0.73	0.64
DenseNet	FT	PVDN-l	Int	0.659	0.56	0.64	0.73
EfficientNet	FT	PVDN-s	PVDN-s	0.833	0.84	0.76	0.68
EfficientNet	FT	PVDN-s	Int	0.601	0.75	0.45	0.32
EfficientNet	FT	PVDN-l	PVDN-s	0.86	0.87	0.80	0.74
EfficientNet	FT	PVDN-l	PVDN-l	0.791	0.80	0.68	0.60
EfficientNet	FT	PVDN-l	Int	0.653	0.70	0.64	0.59
NasNet	FT	PVDN-s	PVDN-s	0.752	0.80	0.62	0.51
NasNet	FT	PVDN-s	Int	0.546	0.48	0.42	0.38
NasNet	FT	PVDN-l	PVDN-s	0.756	0.74	0.62	0.54
NasNet	FT	PVDN-l	PVDN-l	0.716	0.72	0.56	0.45
NasNet	FT	PVDN-l	Int	0.627	0.60	0.58	0.56
DeiT	FT	PVDN-s	PVDN-s	0.737	0.80	0.56	0.43
DeiT	FT	PVDN-s	Int	0.564	0.45	0.55	0.71
DeiT	FT	PVDN-l	PVDN-s	0.775	0.84	0.62	0.49
DeiT	FT	PVDN-l	PVDN-l	0.715	0.81	0.49	0.35
DeiT	FT	PVDN-l	Int	0.651	0.61	0.59	0.57
CaiT	FT	PVDN-s	PVDN-s	0.787	0.81	0.65	0.54
CaiT	FT	PVDN-s	Int	0.688	0.56	0.59	0.63
CaiT	FT	PVDN-l	PVDN-s	0.818	0.90	0.71	0.59
CaiT	FT	PVDN-l	PVDN-l	0.746	0.84	0.56	0.43
CaiT	FT	PVDN-l	Int	0.698	0.72	0.52	0.41
DenseNet	Int-FE-FT	Int	Int	0.731	0.85	0.52	0.37
EfficientNet	Int-FE-FT	Int	Int	0.767	0.78	0.62	0.52
NasNet	Int-FE-FT	Int	Int	0.648	0.66	0.56	0.50
DeiT	Int-FE-FT	Int	Int	0.652	0.70	0.50	0.39

CaiT	Int-FE-FT	Int	Int	0.744	0.78	0.61	0.51
DenseNet	Int-FT	Int	Int	0.758	0.84	0.58	0.44
EfficientNet	Int-FT	Int	Int	0.734	0.72	0.56	0.46
NasNet	Int-FT	Int	Int	0.633	0.59	0.45	0.36
DeiT	Int-FT	Int	Int	0.675	0.61	0.58	0.56
CaiT	Int-FT	Int	Int	0.742	0.82	0.55	0.41

Table 7.2 Results from the model trainings when evaluating with the imbalanced test sets from PVDN-s and internal.

Model	**Process**	**Train set**	**Weighted F1**	**Weighted F1**
			(PVDN)	(Internal)
DenseNet	FE	PVDN-s	0.78	
DenseNet	FE	PVDN-l	0.75	
DenseNet	FE	PVDN-l	0.75	
EfficientNet	FE	PVDN-s	0.81	
EfficientNet	FE	PVDN-l	0.81	
EfficientNet	FE	PVDN-l	0.81	
NasNet	FE	PVDN-s	0.75	
NasNet	FE	PVDN-l	0.74	
NasNet	FE	PVDN-l	0.74	
CaiT	FE	PVDN-l	0.74	
CaiT	FE	PVDN-l	0.74	
CaiT	FE	PVDN-s	0.73	
DeiT	FE	PVDN-s	0.75	
DeiT	FE	PVDN-l	0.74	
DeiT	FE	PVDN-l	0.74	
DenseNet	FE-FT	PVDN-s	0.9	0.81
DenseNet	FE-FT	PVDN-l	0.91	0.76
EfficientNet	FE-FT	PVDN-l	0.89	0.83
EfficientNet	FE-FT	PVDN-s	0.9	0.77
NasNet	FE-FT	PVDN-l	0.88	0.75
NasNet	FE-FT	PVDN-s	0.88	0.66
CaiT	FE-FT	PVDN-l	0.88	0.82
CaiT	FE-FT	PVDN-s	0.86	0.76
DeiT	FE-FT	PVDN-l	0.84	0.72
DeiT	FE-FT	PVDN-s	0.85	0.67
DenseNet	FT	PVDN-s	0.87	0.81
DenseNet	FT	PVDN-l	0.9	0.77
EfficientNet	FT	PVDN-s	0.89	0.82
EfficientNet	FT	PVDN-l	0.9	0.8

NasNet	FT	PVDN-l	0.84	0.7
NasNet	FT	PVDN-s	0.83	0.56
CaiT	FT	PVDN-l	0.89	0.85
CaiT	FT	PVDN-s	0.87	0.73
DeiT	FT	PVDN-l	0.86	0.79
DeiT	FT	PVDN-s	0.83	0.6
DenseNet	Int-FE-FT	Internal	0.79	0.86
EfficientNet	Int-FE-FT	Internal	0.35	0.85
NasNet	Int-FE-FT	Internal	0.35	0.72
CaiT	Int-FE-FT	Internal	0.87	0.84
DeiT	Int-FE-FT	Internal	0.82	0.76
DenseNet	Int-FT	Internal	0.79	0.88
EfficientNet	Int-FT	Internal	0.06	0.81
NasNet	Int-FT	Internal	0.02	0.71
CaiT	Int-FT	Internal	0.83	0.87
DeiT	Int-FT	Internal	0.85	0.79

Bibliography

Aggarwal, C. C. (2018). *Neural Networks and Deep Learning (A Textbook)*. 1st ed. Springer International Publishing.

Bezdan, T. and N. Bacanin (2019). “Convolutional neural network layers and architectures”, pp. 445–451. DOI: `10.15308/Sinteza-2019-445-451`.

Chollet, F. et al. (2015–2021). *Keras documentation: keras applications*. `https://keras.io/api/applications/`. Accessed: March 24, 2023.

Developers, G. (n.d.). *Sampling and splitting data for machine learning*. `https://developers.google.com/machine-learning/data-prep/construct/sampling-splitting`. Accessed on March 24, 2023.

Dosovitskiy, A., L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby (2021). *An image is worth 16x16 words: transformers for image recognition at scale*. arXiv: `2010.11929 [cs.CV]`.

Ewecker, L., E. Asan, L. Ohnemus, and S. Saralajew (2022). *Provident vehicle detection at night for advanced driver assistance systems*. arXiv: `2107.11302 [cs.CV]`.

Ghayoumi, M. (2022). *Deep Learning in Practice*. 1st. [cited 2023 Mar 12]. CRC Press, Taylor and Francis Group.

Hendrycks, D. and K. Gimpel (2020). *Gaussian error linear units (gelus)*. URL: `https://arxiv.org/abs/1606.08415`.

Huang, G., Z. Liu, L. van der Maaten, and K. Q. Weinberger (2017). “Densely connected convolutional networks”, pp. 2261–2269. DOI: `10.1109/CVPR.2017.243`.

Keras (2023). *Transfer learning & fine-tuning*. `https://keras.io/guides/transfer_learning/`. [Accessed on 24th March 2023].

Loshchilov, I. and F. Hutter (2017). *Sgdr: stochastic gradient descent with warm restarts*. arXiv: `1608.03983 [cs.LG]`.

Lu, J. (2022). “Gradient descent, stochastic optimization, and other tales”. *ArXiv* **abs/2205.00832**.

Nwankpa, C., W. Ijomah, A. Gachagan, and S. Marshall (2018). “Activation functions: comparison of trends in practice and research for deep learning”.

O’Shea, K. and R. Nash (2015). “An introduction to convolutional neural networks”.

Ohnemus, L., L. Ewecker, E. Asan, S. Roos, S. Isele, J. Ketterer, L. Müller, and S. Saralajew (2021). *Provident vehicle detection at night: the pvdn dataset*. arXiv: `2012.15376 [cs.CV]`.

Oldenziel, E., L. Ohnemus, and S. Saralajew (2020). “Provident detection of vehicles at night”. In: *2020 IEEE Intelligent Vehicles Symposium (IV)*, pp. 472–479. DOI: `10.1109/IV47402.2020.9304752`.

Powers, D. M. W. (2020). *Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation*. arXiv: `2010.16061 [cs.LG]`.

Sarkar, D., R. Bali, and T. Ghosh (2018). *Hands-On Transfer Learning with Python: Implement advanced deep learning and neural network models using TensorFlow and Keras*. Packt Publishing.

Silva, I. da, D. Hernane Spatti, R. Andrade Flauzino, L. Liboni, and S. dos Reis Alves (2017). *Artificial Neural Networks. A Practical Course*. 1st. Springer International Publishing.

Tan, M. and Q. V. Le (2020). *Efficientnet: rethinking model scaling for convolutional neural networks*. arXiv: `1905.11946 [cs.LG]`.

TensorFlow 2.0 API Docs (n.d.). `https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/Model\#fit`. Accessed on March 24, 2023.

TensorFlow Hub Authors (2018–2023). *Tensorflow hub*. `https://tfhub.dev/`. Accessed: March 24, 2023.

Touvron, H., M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou (2021a). *Training data-efficient image transformers & distillation through attention*. arXiv: `2012.12877 [cs.CV]`.

Touvron, H., M. Cord, A. Sablayrolles, G. Synnaeve, and H. Jégou (2021b). *Going deeper with image transformers*. arXiv: `2103.17239 [cs.CV]`.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017). *Attention is all you need*. arXiv: `1706.03762 [cs.CL]`.

Yang, S., W. Xiao, M. Zhang, S. Guo, J. Zhao, and F. Shen (2022). *Image data augmentation for deep learning: a survey*. arXiv: `2204.08610 [cs.CV]`.

Zoph, B., V. Vasudevan, J. Shlens, and Q. V. Le (2018). “Learning transferable architectures for scalable image recognition”, pp. 8697–8710. DOI: `10.1109/CVPR.2018.00907`.

Lund University **Department of Automatic Control** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER'S THESIS *Date of issue* May 2023 *Document Number* TFRT-6195
Author(s) Celine Ivarsson Jennifer Zacke	*Supervisor* Liam Fahlstad, Qualcomm Joar Karlgren Gustavsson, Qualcomm Mats Fridén, Qualcomm Simon Näsfeldt, Qualcomm Rickard Peterson, Qualcomm Pontus Giselsson, Dept. of Automatic Control, Lund University, Sweden Bo Bernhardsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle

Night-time Vehicle Detection Based on Observable Light Cues Using Deep Learning

Abstract

This thesis investigates the issue of computer vision-based detection of oncoming cars during night-time, a critical road safety issue for automated high-beam assist. We propose a holistic image classification approach that uses deep learning methods to detect light artifacts from an oncoming car's headlights before the car is entirely visible. We explore six different model architectures, including both convolutional neural networks and transformer-based models. We train them using transfer learning with both public and internal datasets using models pre-trained on ImageNet.
We evaluate the generalization ability of the models and find that they can achieve up to 71% accuracy when trained on the public dataset and evaluated on the class-balanced internal dataset. Our results show that both convolution-based and transformer-based models have potential in performance for this task, with the best models reaching up to 88% accuracy when trained with the full public dataset and evaluated with the class-balanced public test set. Our research contributes to the field by introducing an approach to detection of oncoming cars and comparing different model architectures for this task.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280-5316		*ISBN*
Language English	*Number of pages* 1-57	*Recipient's notes*
Security classification		

http://www.control.lth.se/publications/