ROAD SIGN RECOGNITION USING SHALLOWNET, LENET, VGGNET AND DETECTION USING FASTER R-CNN

A THESIS SUBMITTED TO
THE FACULTY OF ARCHITECTURE AND ENGINEERING
OF
EPOKA UNIVERSITY

BY

MARKO MAKSUTI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR BACHELOR DEGREE
IN SOFTWARE ENGINEERING

JUNE, 2024

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Marko Maksuti

Signature:

ABSTRACT

ROAD SIGN RECOGNITION USING SHALLOWNET, LENET, VGGNET AND DETECTION USING FASTER R-CNN

Maksuti, Marko

B.Sc., Department of Computer Engineering

Supervisor: M.Sc. Sabrina Begaj

Road sign recognition and detection are critical components of advanced driver assistance systems (ADAS) and autonomous vehicles, ensuring safe and efficient navigation. This thesis explores the application of deep learning techniques, particularly convolutional neural networks (CNNs), for road sign detection and detection from images.

The proposed approach consists of two main stages: road sign recognition and detection. In the recognition stage, various CNN-based models will be tested to classify road signs present in the input image. The models will be trained on a large dataset of annotated road sign images, enabling them to learn discriminative features and patterns associated with different road sign categories.

For the detection stage, the thesis will explore and evaluate the Faster R-CNN algorithm to recognize and then determine the spatial coordinates of the recognized road signs within the image.

Keywords: Classification; Object Detection; Object Recognition; Deep Learning; Autonomous Vehicles; Computer Vision;

ABSTRAKT

KLASIFIKIMI I SHENJAVE TË QARKULLIMIT DUKE PËRDORUR SHALLOWNET, LENET, VGGNET DHE ZBULIMI I TYRE DUKE PËRDORUR FASTER R-CNN

Maksuti, Marko

B.Sc., Departamenti i Inxhinierisë Kompjuterike

Supervizor: M.Sc. Sabrina Begaj

Klasifikimi dhe zbulimi i shenjave të qarkullimit janë komponentë kritikë të sistemeve të avancuara për asistencën e shoferave, dhe automjeteve autonome, për të siguruar drejtim të sigurt dhe efikas të mjetit. Kjo tezë eksploron aplikimin e teknikave të deep learning (shqip: "mësimit të thellë"), në veçanti rrjeteve nervore konvolucionale (CNN), për klasifikim dhe zbulim të shenjave të qarkullimit në imazhe.

Rruga e propozuar përbëhet nga dy faza kryesore: klasifikimi i shenjave të qarkullimit dhe zbulimi. Në fazën e klasifikimit, disa modele të bazuar në CNN do të testohen për te klasifikuar shenjat e qarkullimit që gjenden në imazhin hyrës. Modelet do të trajnohen në një grup të madh të dhënash me imazhe të shënuara të shenjave të qarkullimit, duke i lejuar ato të mësojnë tipare diskriminuese dhe strukturën e lidhur me kategori të ndryshme te shenjave të qarkullimit.

Në fazën e zbulimit, teza do të eksplorojë dhe vlerësojë algoritmin Faster R-CNN për të njohur dhe më pas për të gjetur koordinatat e shenjave të qarkullimit të njohura brenda imazhit.

Fjalë kyçe: Klasifikim; Zbulim objektesh; Deep Learning; Automjete autonome; Shikim kompjuterik;

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
CONV	Convolutional
POOL	Pooling
FC	Fully Connected
RPN	Region Proposal Network
STSD	Swedish Traffic Sign Dataset
GTSRB	German Traffic Sign Recognition Benchmark
mAP	Mean Average Precision
RSV	road-signs-vanga

CHAPTER 1

INTRODUCTION

Road sign recognition and detection systems are crucial components of advanced driver assistance systems (ADAS) and autonomous vehicles, helping to ensure safety and compliance with traffic regulations. Traditional computer vision approaches to this problem often relied on hand-crafted features and heuristics, which can lack the robustness and generalization required for real-world driving scenarios. In recent years, deep learning techniques, particularly convolutional neural networks (CNNs), have demonstrated remarkable performance on image classification and object detection tasks, learning powerful feature representations directly from data. This thesis explores the application of multiple CNN architectures for road sign classification, followed by an object detection algorithm to localize signs within images. By leveraging the powerful feature learning capabilities of deep neural networks, this work aims to implement an accurate and reliable road sign recognition system that can operate under diverse lighting, weather, and occlusion conditions. The implementation and evaluation of such a system using state-of-the-art deep learning models has potential relevance to the ongoing development of intelligent transportation systems and enhancing road safety.

1.1. Motivation and importance

Reliable road sign detection and recognition systems are of paramount importance for advanced driver assistance systems (ADAS) and autonomous vehicles. Failure to properly identify and comprehend traffic signals, warning signs, and regulatory signs can lead to dangerous situations and violations of traffic rules. According to the National Highway

Traffic Safety Administration, in 2019 there were an estimated 36,096 fatal motor vehicle traffic crashes in the United States alone [1]. While human error accounts for a majority of these incidents, equipping vehicles with intelligent computer vision capabilities to automatically detect and recognize road signs can potentially prevent many crashes caused by driver distraction or mistake.

Beyond safety benefits, accurate sign detection and recognition enables autonomous vehicles to effectively comprehend their environment and make appropriate navigational decisions based on traffic signs and road markings. This capability is an essential component for deploying fully self-driving cars at scale. As urban and suburban regions continue developing their smart city infrastructure, robust vision-based perception of traffic signs and signals will be critical for smart transportation systems to function properly.

Furthermore, road sign recognition systems can streamline asset management for municipalities. By continuously monitoring the presence and state of traffic signs within a region, cities can be alerted when signs are missing, damaged, obstructed or due for replacement. This can improve operational efficiency and reduce potential liabilities from non-compliance with traffic regulations.

With the rapid growth of ADAS features and autonomous driving initiatives across the automotive industry and smart cities, there is an urgent need for highly accurate and reliable road sign detection and recognition algorithms that can operate under real-world conditions of varying illumination, weather, occlusions, rotations, and ambient noise. This serves as the key motivation behind implementing robust deep learning solutions for this computer vision task.

1.2. Project objectives

The primary objectives of this bachelor's thesis project are twofold: 1) Implement and test multiple deep learning architectures for accurate road sign classification, and 2) Develop an effective object detection module to localize road signs within images. The goal is to explore the application of these algorithms to the road sign recognition task, rather than to create an optimized production-level system. Through rigorous empirical evaluations on established road sign datasets like DFG Traffic Sign Dataset, the project aims to analyze the behaviors, strengths, and limitations of the implemented models across a diverse range of test data. Model performance metrics like precision, recall, will be measured and reported, but optimizing for high accuracy is not the primary objective.

The task of classification involves inputting an image of a road sign to the network, and getting the respective class label as the output. For this task images are already assumed to be cropped tightly around the subject, as illustrated in Figure 1.2.1. For the task of object detection on the other hand, the image can include a background, other objects, or other forms of noise, as well as multiple road signs. The goal of object detection is to find all road signs in the image and draw a bounding box around them, while also labeling them with the corresponding class and confidence score. This is illustrated in Figure 1.2.2.

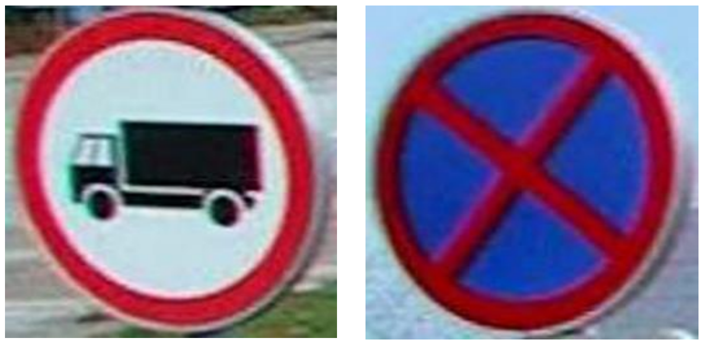


Figure 1.2.1. Examples of images that can be used for classification

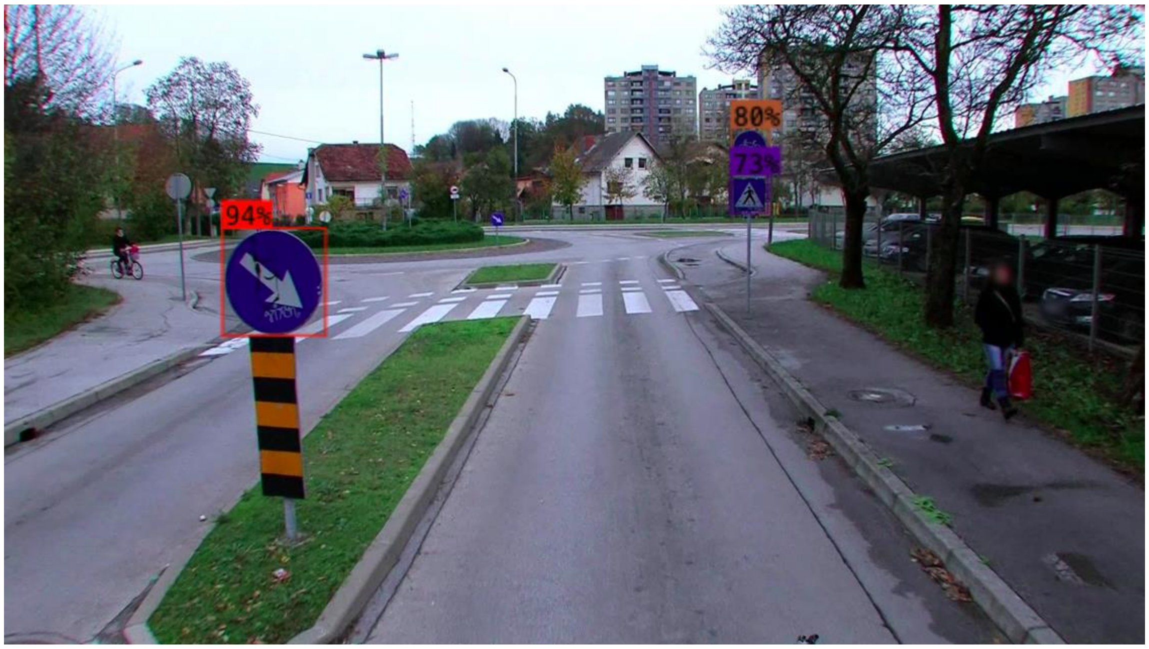


Figure 1.2.2. Example of object detection in action

1.3. Evaluation Metrics

For the classification component, measuring the overall accuracy (percentage of correct predictions) on a held-out test set is a common top-line metric. However, accuracy can be misleading for imbalanced datasets, so metrics like precision (percentage of positive predictions that are correct) and recall (percentage of actual positives identified) provide additional insights. The F1 score, which is the harmonic mean of precision and recall, gives a balanced sense of a model's performance.

For object detection tasks like localizing traffic signs, mean Average Precision (mAP) is a widely adopted metric. It is computed as the average of Average Precision (AP) scores. AP is evaluated per class, it combines the precision and recall values. mAP combines AP scores for all classes across different Intersection over Union (IoU) thresholds into a single number, facilitating comparison across methods. Higher mAP indicates better detection precision and recall. IoU by itself is a measure of how accurate object detection is, it measures how closely the predicted bounding box matches the ground truth. Speed (inference time) is another critical factor, especially for real-time applications. A combination of these classification, detection, and speed metrics provides a comprehensive evaluation of traffic sign recognition systems.

CHAPTER 2

LITERATURE REVIEW

2.1. Convolutional Neural Networks

Convolutional neural networks (CNNs) are a type of deep neural network architecture that has proven highly effective for computer vision tasks like image classification, object detection, and segmentation. The concept of convolutional neural networks (CNNs) was first introduced by LeCun et al. [2] in their seminal 1998 paper "Gradient-Based Learning Applied to Document Recognition". This groundbreaking work proposed a multilayer neural network architecture designed for handwritten digit classification.

A key innovation of CNNs is separating the steps of feature extraction and classification into distinct components. Rather than using hand-engineered features as inputs, CNNs employ a feature learning module consisting of alternating convolutional and subsampling (aka pooling; abbreviated as POOL) layers to automatically extract relevant visual representations from raw pixel data. The convolutional layers apply trainable filters that capture localized patterns like edges or shapes as the filters are spatially swept across the input images. These learned filters effectively act as feature extractors, with their weights treated as trainable parameters during the learning process. The subsampling layers downsample the feature maps to reduce the number of parameters.

The feature maps output from the convolution/subsampling layers then are fed into fully-connected classification layers, similar to traditional neural networks, to associate the extracted features with the final output categories.

This separation of responsibilities enabled CNNs to build powerful hierarchies of localized features, with successive layers learning higher-level representations appropriate for the target task. LeCun et al. demonstrated the effectiveness of this architecture on handwritten digit classification.

2.2. LeNet

To validate their CNN approach, LeCun et al. [2] implemented and evaluated a specific architecture called LeNet, for recognizing handwritten digit images.

LeNet consisted of several consecutive layers: two convolutional layers, two subsampling layers, and two fully-connected layers leading to a final classification representing the digit classes 0-9.

In the first convolutional layer, LeNet extracted localized edge features by convolving the input images with learned 5x5 kernel filters, followed by subsampling. The second convolutional layer then combined these edges into higher-level patterns like corners or slants over broader receptive fields.

After this alternating convolution and subsampling feature extraction module, the resulting feature maps were flattened and fed into two fully-connected layers that integrated the localized features across the entire image to recognize entire digit shapes and output the final classification.

LeNet's elegant yet powerful architecture demonstrated the representations learned by CNNs could outperform hand-engineered features on complex pattern recognition tasks.

Moreover, LeNet exhibited key strengths like translation invariance and required far fewer parameters compared to fully-connected networks.

The success of the relatively small LeNet architecture validated the core CNN principles and paved the way for much larger and more sophisticated models in later years.

2.3. VGGNet

While LeCun et al.'s pioneering work introduced the core CNN concepts, the VGGNet architecture proposed by Simonyan and Zisserman [3] demonstrated the powerful capabilities of much deeper convolutional neural networks on large-scale vision tasks.

In their 2014 paper "Very Deep Convolutional Networks for Large-Scale Image Recognition", the VGG researchers investigated the effect of CNN model depth by constructing extremely deep architectures composed of 16-19 weight layers using very small 3x3 convolutional filters throughout the network.

This simple yet innovative design choice enabled VGGNet models to achieve state-of-the-art accuracies on the challenging ImageNet dataset at the time, which consisted of over 14 million images across 1000 classes [4]. The results showed that representational depth was critical for making accurate predictions in such large-scale visual recognition tasks.

A key advantage of the VGGNet architecture was its uniform convolution and max-pooling layers, with the same small 3x3 filter size across the entire network. This homogeneity made the architecture easy to implement and extend to even deeper depths if required.

Moreover, the use of very small 3x3 convolutional filters offered additional benefits. Stacking several 3x3 layers had an effective receptive field equivalent to larger filters, but with fewer parameters and computational overhead compared to using large kernel sizes directly.

While achieving impressive performance, the downside was that VGGNet contained over 130 million parameters, making the models quite large and memory intensive.

Nonetheless, the VGGNet work highlighted that CNN depth was a vital factor for visual representation learning. It motivated subsequent architectures like GoogLeNet and ResNets to explore even deeper and more efficient network designs.

2.4. Object Detection and Faster R-CNN

Prior to the adoption of deep learning, traditional object detection pipelines relied on sliding window methods that evaluated a classifier across different regions of an image in a brute-force manner. While achieving reasonable accuracy, these approaches were extremely computationally inefficient and incapable of scaling to large numbers of object classes.

The R-CNN model introduced by Girshick et al. [5] was one of the pioneering works that integrated deep convolutional neural networks for object detection. By repurposing CNN features trained on ImageNet classification (usually the VGG architecture), R-CNN could localize objects within region proposals generated by an external algorithm and classify their categories. Despite its accuracy improvements, the multi-stage pipeline was quite slow due to repeated CNN evaluations. Fast R-CNN [6] streamlined this process by computing CNN features just once per image, then mapping region proposals to their feature representations via RoI pooling layers. This eliminated redundant convolution computations, significantly accelerating detection speeds.

Building upon Fast R-CNN, the Faster R-CNN architecture proposed by Ren et al. [7] introduced a Region Proposal Network (RPN) that allowed for end-to-end training and object detection in a single unified model, without the need to rely on an external region proposal algorithm anymore.

The key innovation was using the RPN to share convolutional features from the backbone network to rapidly generate high-quality region proposals. These proposals were then fed into the remainder of the Fast R-CNN detection pipeline for classification and bounding box regression.

By eliminating the reliance on external region proposal methods, Faster R-CNN models became more accurate and efficient. Faster R-CNN demonstrated the benefits an end-to-end trainable architecture could have on unifying the tasks of region proposal and object detection into one jointly optimized model. Its impact led to the widespread adoption of this two-stage proposal + detection framework as a fundamental approach for object localization tasks including road sign detection.

Mask R-CNN is an extension of Faster R-CNN proposed by Kaiming He et al. in 2017, which adds a branch for predicting segmentation masks for each object detected in the image; it can not only detect objects but also segment them at the pixel level.

2.5. Related Work

Numerous studies have investigated the application of deep learning techniques, particularly convolutional neural networks (CNNs), for accurate road sign classification and detection in recent years. Tabernik and Skočaj [8] conducted an extensive evaluation of CNN approaches on this problem in their work "Deep Learning for Large-Scale Traffic-Sign Detection and Recognition.". The authors evaluate different algorithms to tackle the task of detection and recognition. They propose several improvements to enhance performance, which are evaluated on a novel dataset (DFG) containing 200 traffic-sign categories, each of at least 200 instances after data augmentation. Importantly, this dataset includes challenging traffic-sign categories that have not been previously considered in other works.

The authors focus on evaluating two algorithms for object detection: Faster R-CNN and Mask R-CNN. They use the Swedish Traffic Sign Dataset (STSD) as a baseline comparison against their newly proposed DFG dataset. Their Faster R-CNN architecture uses VGG16, whereas the Mask R-CNN one uses ResNet-50 [9]. Both networks are initialized with a model pre-trained on ImageNet. For Faster R-CNN they use a learning rate of 0.001 and a weight decay of 0.0005. For Mask R-CNN they use a learning rate of 0.0025 and a weight decay of 0.0001. In both they use a momentum of 0.9.

On the STSD dataset, the authors achieve over 94% scores on mAP, recall, f-measure and precision using Faster R-CNN, and slightly lower precision and f-measure scores on the normal Mask R-CNN. With their adaptations on the algorithm they achieve 95.2 mAP, 97% f-measure, 96.7% recall and 97.5% precision.

On their DFG dataset, the authors first evaluate the RPN network separately for every class, and then they evaluate the full pipeline, that is RPN + classification. All algorithms achieve 90%+ mAP scores, with Faster R-CNN doing the worst and their proposed adapted Mask R-CNN doing the best, achieving a mAP^{50} score of 95%, mAP$^{50:95}$ of 84.4%, and recall of 96.5% on the augmented dataset. The authors then do an evaluation considering different traffic sign sizes, and they conclude that smaller objects perform worse.

Several other studies have explored deep learning approaches for traffic sign perception beyond the work of Tabernik and Skočaj. For instance, A.D. Kumar et al. [10] proposed a novel deep learning method using capsule networks for traffic sign classification, achieving an accuracy of 97.6% on the German Traffic Sign Recognition Benchmark (GTSRB) . Yang Gu and Bingfeng Si [11] proposed a novel lightweight framework based on YOLOv4, achieving a mAP score 80.62% on the GTSDB dataset, demonstrating a balance between accuracy and computational efficiency; with the original more heavyweight YOLOv4 they achieved a mAP score of 89.11%. Wang et al. [12] improved YOLOv4 by integrating a Triplet attention mechanism and BiFPN, which enhanced small object detection and achieved an mAP of 60.4%, an 8% improvement over the original YOLOv4 on the TT100K-COCO dataset. Another study by Zhu Mao et al. [13] presented a deep CNN-based pipeline for embedding road sign models, achieving an mAP of 93.5% for road sign detection using oblique aerial images.

CHAPTER 3

THE DATASETS AND ALGORITHMS

3.1. The DFG Traffic Sign Dataset

The first dataset we'll be exploring is the same dataset discussed in section 2.5. of this thesis, Tabernik and Skočaj's DFG Traffic Sign dataset. As mentioned previously, this dataset contains 200 classes, including classes not commonly explored. The original dataset contains 5254 training images and 1703 testing images, of which 6758 are in 1920x1080 resolution and 199 in 720x576 resolution [14]. After data augmentation performed by the authors of this dataset, there are in total 16264 images. In the original dataset there are at least 20 instances per class, as shown in Figure 3.1.1. After augmentation, the dataset is more balanced and has roughly 200 instances per class as shown in Figure 3.1.2.

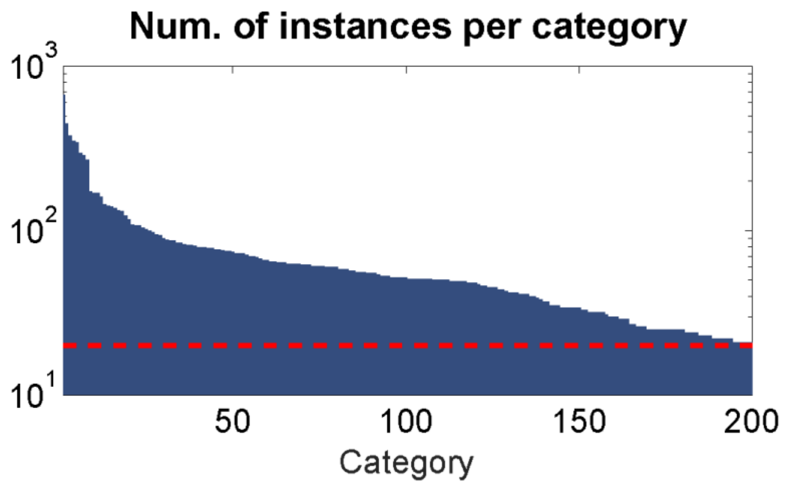


Figure 3.1.1. Data distribution in the original DFG dataset [14]

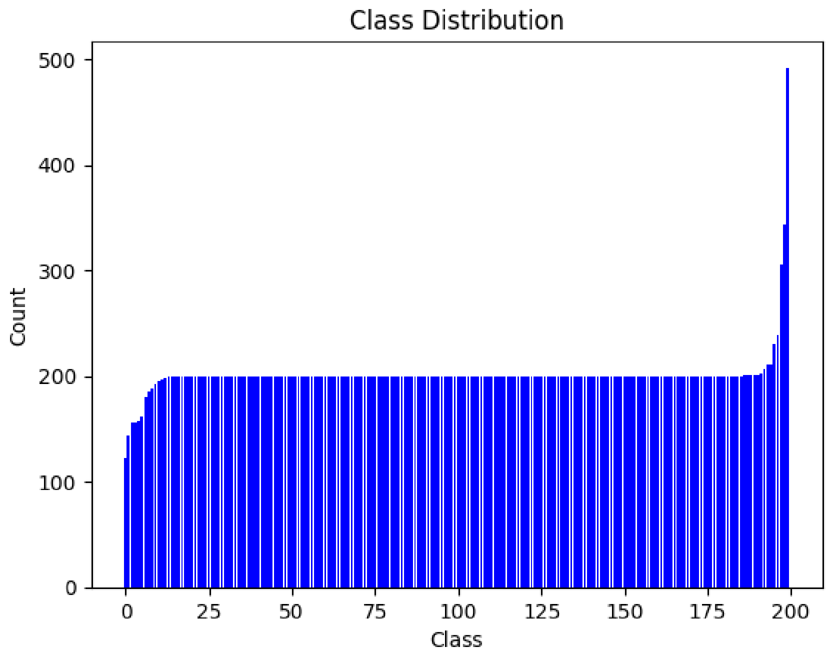


Figure 3.1.2. Data distribution in the augmented DFG dataset

Each image has one or more instances of various traffic signs in it. The original images are captured with a camera, the augmented ones are artificially constructed by placing cropped instances of traffic signs on random places of another image which serves as the background. Figure 3.1.3. shows an example of each type of image.

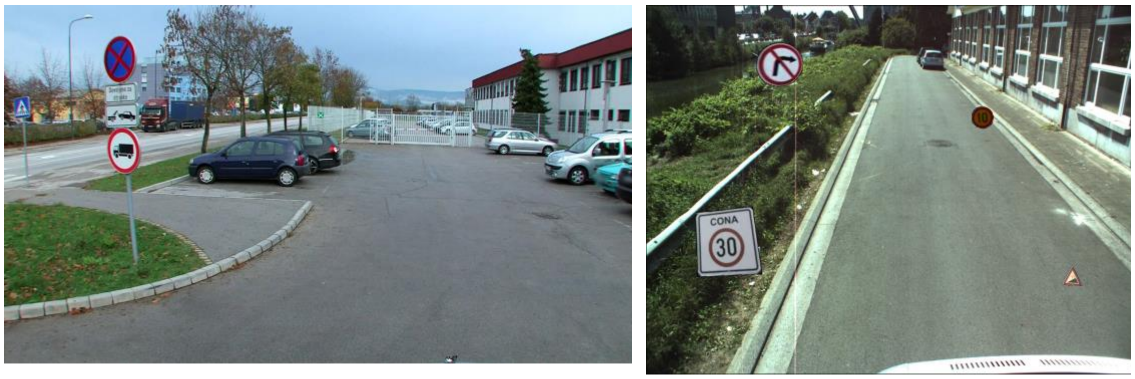


Figure 3.1.3. Image samples from the DFG dataset, natural (left), artificial (right)

3.2. Road-signs-vanga

For comparison, we'll also be testing the algorithms in another dataset from Roboflow, a website providing various open-source datasets and pre-trained models. The dataset is found in this link: https://universe.roboflow.com/midstem/RSV/. We'll be calling it according to the identifier link, road-signs-vanga (RSV). This is a simpler dataset with fewer images, fewer classes and a smaller resolution of 640x640. The road signs also take a bigger portion of the image. Overall those features may make this dataset faster to train, however this dataset is also more imbalanced, which will likely lead to certain classes performing better than others. It also has overall less images. Figure 3.2.1. shows the data distribution per class.

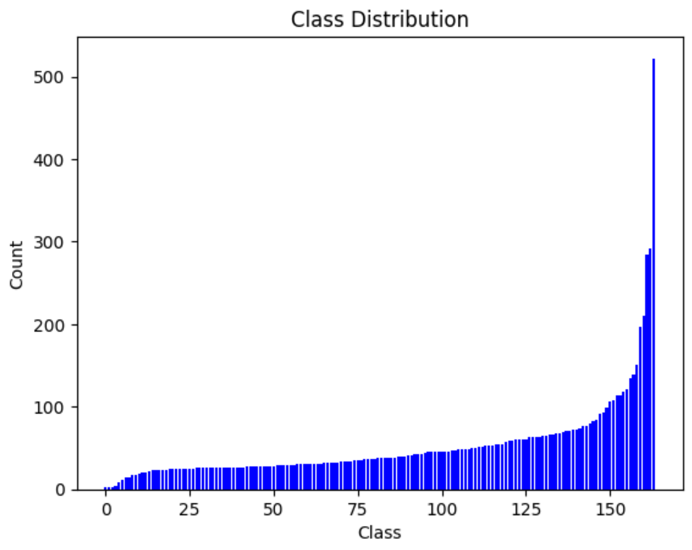


Figure 3.2.1. Data distribution in the RSV dataset

3.3. Road sign classification

For the first task of road sign classification, we will be modifying our datasets. Rather than work directly on the images that contain multiple classes of objects to detect, we will be extracting the objects themselves, each into a single square image. Both datasets contain an annotation file in JSON format, which specifies information about each image, such as the coordinates of each object in the image (the ground truth bounding boxes) and the class labels corresponding to them. We will leverage this annotation file and use the bounding box coordinates to crop every object out of the bigger image. Each road sign image will be resized to 64x64 (stretching it in the process), which should be a good enough resolution to accurately represent that image, while also avoiding a large training time. For this same reason we will also be converting the images to grayscale. Examples from the resulting images are shown in Figure 3.3.1.

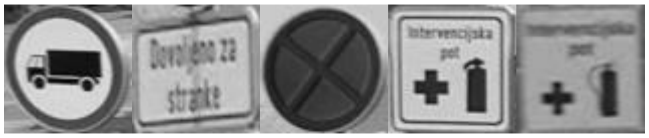


Figure 3.3.1. Images after being preprocessed for the image classification algorithms

Next we will be evaluating multiple CNN algorithms for classification, starting with a simple architecture with only one convolutional (CONV) layer, commonly referred to as ShallowNet, then continuing with the algorithms discussed in chapter 2, LeNet and VGG.

For ShallowNet the one and only CONV layer will have 32 filters of size 3x3. For LeNet we will start with a CONV layer consisting of 20 5x5 filters and a padding to preserve image size, followed by a max pooling layer of size and stride of 2x2. These two layers will be followed by another pair of CONV and pooling layers of the same parameters. At the end they'll be connected to two fully connected (FC) layers, one consisting of 500 neurons and the other of how many classes there are. We'll use ReLU as the activation function for all layers except the last, which will use softmax to give us the probability of the input image belonging to a class.

Lastly we will be testing a variant of the VGGNet family of architectures, not VGG16 but a smaller variant, MiniVGGNet, due to the fact that images are relatively small (64x64) and to save computational effort. Instead of 16 weighted layers, we will have 6, 4 CONV layers and 2 FC layers. We'll start with 32 3x3 filters in the first CONV layer, followed by the ReLU activation function and a batch normalization (BN) layer. This set of layers will be repeated before applying a max pooling layer of size 2x2 and a Dropout layer of 0.25 probability for regularization. Then we will repeat the same (CONV + ReLU + BN) * 2 + POOL + Dropout, but this time with double the amount of filters in the CONV layers, 64. Then we will add a FC layer of 512 neurons, again followed by ReLU + BN + Dropout of 0.5 probability before finishing off with another FC layer with as many neurons as the number of classes, which is fed to the softmax activation function for the final classification.

3.4. Road sign detection

For object detection, we will use the Faster R-CNN algorithm outlined in section 2.4. We will use a library called Detectron2 (further outlined in chapter 4) which facilitates the process of training and provides pre-trained weights to use as a backbone for feature extraction. Specifically Detectron2 provides weights of the ResNet-50 architecture, pre-trained on the ImageNet dataset.

CHAPTER 4

MATERIALS AND METHODS

This project implements and evaluates multiple deep learning models for the tasks of road sign classification and detection. The implementation leverages state-of-the-art tools and libraries that facilitate rapid experimentation, and deployment of neural network architectures tailored for computer vision applications.

4.1. Python

All the code for this project will be written in the Python programming language. Python has found a widespread adoption in the machine learning community and offers a rich ecosystem of supporting libraries. Its clean and intuitive syntax, and its strong data processing capabilities make it well-suited for developing and deploying deep learning systems.

4.2. Google Colab

Deep Learning for computer vision problems can be computationally intensive. For this reason this project will be taking advantage of Google Colaboratory's (Colab) hardware acceleration to perform the model training process. Colab provides free access to powerful GPU and TPU resources hosted in the cloud, eliminating the need for local potentially expensive hardware. Google Colab uses a Jupyter notebook interface which tightly integrates code, documentation, and visualizations within a single environment. Furthermore, Colab offers seamless integration with Google Drive, streamlining data management and sharing across collaborators. Colab can be accessed for free at https://colab.research.google.com.

4.3. Deep Learning Libraries

At the core of this implementation are two key deep learning libraries: Keras and TensorFlow. Keras serves as a high-level neural network API that enables rapid prototyping of deep learning models. Its user-friendly, modular architecture simplifies the process of defining and training various CNN architectures. Utilities for data preprocessing, model visualization, and performance evaluation are also provided through the Keras ecosystem.

While Keras abstracts much of the low-level implementation details, it builds upon the robust and highly scalable TensorFlow framework. TensorFlow is a production-grade system for deploying machine learning models in a variety of environments, from cloud platforms to web services and mobile/embedded devices. Its flexible architecture and comprehensive documentation make it suitable for transitioning models from research to real-world applications.

4.4. Detectron2

The task of object detection is a problem too complex to solve from scratch. While the deep learning frameworks mentioned above provide the building blocks for constructing neural network models, we will need something more to handle the complexity of object detection. Detectron2 is one such library that offers high-level utilities tailored specifically for the object detection domain.

Detectron2 provides utilities for easily integrating custom datasets like the road sign datasets used in this project. Earlier on this thesis we mentioned the datasets contain a JSON annotations file. This file actually follows the COCO (Common Objects in Context) format to store class & bounding box information for the dataset, which Detectron2 natively supports.

Lastly Detectron2 encapsulates not just the model architectures, but also evaluation metrics like mAP that are essential for benchmarking detection performance on the road sign task. Its visualization tools may also aid in qualitative analysis.

4.5. NumPy

The NumPy library serves as the core utility for numerical computing in Python. Its module provides support for large, multi-dimensional arrays and matrices, along with a vast library of high-level mathematical functions to operate on these arrays.

In the context of this project, NumPy plays a crucial role in efficiently representing and manipulating the image data and annotations across the various datasets. The road sign images are stored as NumPy arrays, enabling vectorized operations like resizing, normalization to be performed rapidly in batch processing.

4.6. HDF5

The RSV dataset is small enough to fit entirely in memory after preprocessing for the classification task, however the DFG dataset is not. In order to accommodate loading large datasets in chunks from disk to memory, the HDF5 (Hierarchical Data Format) file format and supporting libraries may be employed for efficient data storage and retrieval.

HDF5 provides a robust way to organize large amounts of heterogeneous data, including images and annotations, within a single file. Its hierarchical structure and compression capabilities enable seamless handling of datasets that may be too large to comfortably load entirely into memory. By storing the images and labels in HDF5 containers, the data can be either accessed directly from disk as a NumPy array or streamed in batches to the deep learning models during training, alleviating potential memory constraints.

4.7. Matplotlib

To effectively visualize and interpret the image data, model architectures, and the results, this project utilizes the Matplotlib plotting library. As one of the most comprehensive visualization packages in Python, Matplotlib provides a rich set of functions for generating visualizations across a wide range of platforms and output formats.

Matplotlib will be initially used to plot the class distribution, to gain an understanding of the dataset and how balanced it is. When analyzing CNN architectures, Matplotlib's plotting capabilities will be used to visualize the model's performance, by training loss and/or accuracy, which will enable us to detect if there's overfitting or underfitting.

4.8. OpenCV

Arguably the most important library in the topic of computer vision, OpenCV offers various image processing capabilities. OpenCV (Open Source Computer Vision Library) is an optimized cross-platform library aimed at accelerating computer vision applications. It provides a Python interface that enables efficient execution of low-level operations on image and video streams.

For this road sign recognition project, OpenCV will be used during the data loading and preprocessing stages. It will initially be used in the image recognition stage to modify the datasets in order to crop each road sign out of the image, using bounding box ground truths from the annotations file. It will also be used to resize the images and convert them to grayscale.

CHAPTER 5

IMPLEMENTATION AND RESULTS

5.1. Setting up the work environment

The first stage of the actual implementation is setting up the work environment. We will put all the code in a Google Colab notebook. Given that Google Colab's resources are temporary and cleared after the session is terminated, we will need a persistence mechanism. Google Colab offers native support for connecting it to Google Drive, which we will use to persist our work.

After we connect to Google Drive, we can download the datasets and copy them there. The DFG dataset will be downloaded directly from the link: https://box.vicos.si/skokec/villard/JPEGImages.tar.bz2 using the wget command line utility, then extracted and decompressed using tar. The second dataset will be downloaded using the Roboflow Python package, specifying the version as 'coco' to ensure compatibility with Detectron2 for object detection. After downloading, the dataset will be unzipped with the unzip tool.

The DFG dataset also has the annotations in a separate file, so we will download that too. The annotations for the augmented dataset come in two versions, one of which marks the images that are too small to be ignored, we will use this one for the classification and the normal one due to incompatibility with Detectron2, for object detection.

5.2. Pre-processing

The next step is data pre-processing. We will begin with a function named ‘extract_bounding_boxes’, taking a list of images as input, an output directory and the annotations file path. This function will go through all the images, read the respective annotations for that image and crop out every bounding box that is not set as ‘ignore’ (in the DFG dataset). Each cropped bounding box will be resized to 64x64 and converted to grayscale, then saved as a separate image in the output directory.

5.3. Loading the data

As mentioned above the DFG dataset is too big to fully fit in memory, so we will use the HDF5 library to merge all images into a single hdf5 file, a format that organizes the files for maximum access efficiency directly from disk. In order to pass the data to Keras for training, we will use a generator function. We will use the normal train_test_split function on a list of indices, then pass those indices to the generator function to sample the actual image data.

The RSV dataset fits in memory so we do not need to use HDF5. We will load all the images as numpy arrays. Those arrays will be serialized to disk and copied to Google Drive to avoid having to load the images one by one again, which generates a huge time overhead.

5.4. Classification

As our first experiment we will train ShallowNet on both datasets, we will set the number of epochs to 100 as a start and learning rate as 0.005. This model did really badly on both datasets, as is shown on Figure 5.4.1 there was barely any learning. In the first dataset the maximum accuracy reached is about 40% on the training accuracy, in the case of the second dataset there is also significant overfitting with the training accuracy being significantly higher than the validation accuracy after around 40 epochs

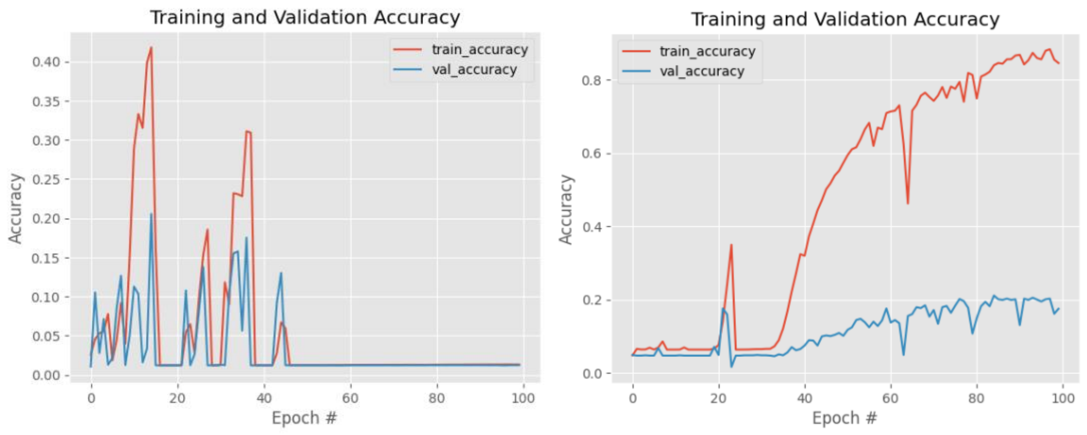
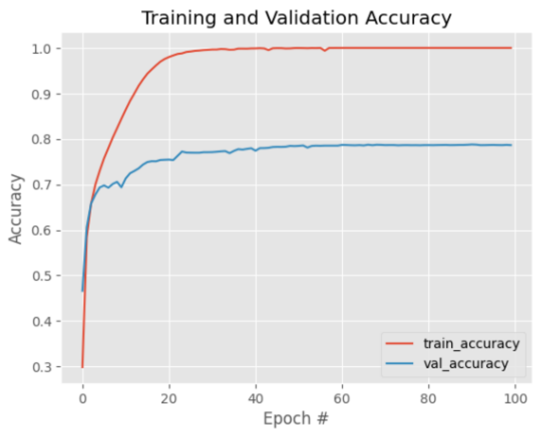


Figure 5.4.1. ShallowNet training and validation accuracy over time, DFG dataset (left), RSV (right)

For our next experiment we will use normalization, specifically we will divide all pixel values by 255, which is the highest possible (white). This immediately results in a huge improvement, right from the first 10 epochs the validation accuracy reaches 70% on the DFG dataset. However it doesn’t get better than 80%, whereas the training accuracy goes higher and higher until it reaches 100%, indicating serious overfitting. The RSV dataset behaves in a similar way, only reaching 69% accuracy. The plots of this experiment are shown below on Figure 5.4.2. From the graphs can also be seen that the number of epochs does not need to be higher than 20 or so, therefore for the next experiments we are going to determine the optimal number of epochs by observing the training vs validation accuracy during training, the moment overfitting starts to happen we can assume we do not need more epochs, or else overfitting is only going to get worse.



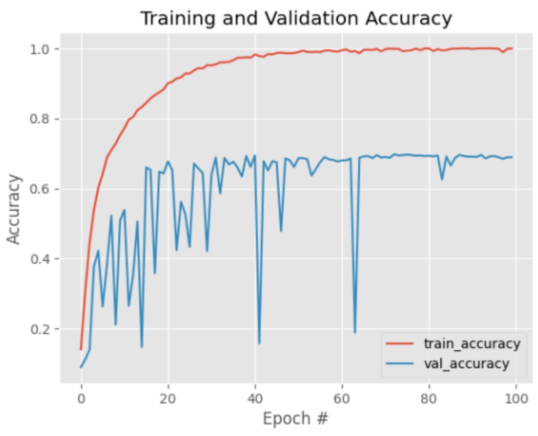


Figure 5.4.2. ShallowNet training and validation accuracy over time after data normalization, DFG dataset (left), RSV (right)

Next we will be evaluating LeNet, a model that has shown good results in small image datasets, although our images are a little higher resolution than the MNIST dataset which LeNet is famous for. We will set the number of epochs lower this time, given the results of the previous experiment.

The results (Figure 5.4.3) show a huge improvement over the previous model. We reach a validation accuracy of over 89% on the DFG dataset and 76% on the RSV dataset. The lower accuracy on the second dataset can be attributed to the lower number of images and the dataset imbalance, problems which could be improved for with data augmentation, which will not be covered in this thesis.

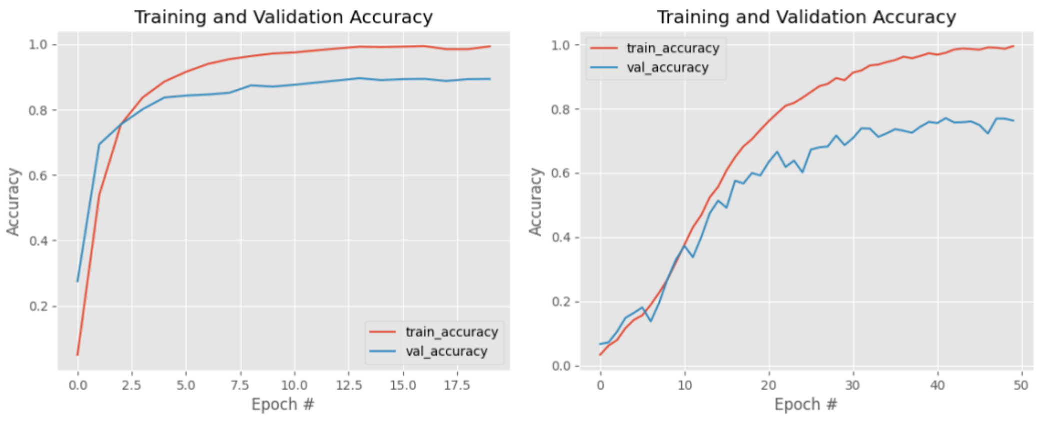


Figure 5.4.3. Training and validation accuracy over time using the LeNet architecture, DFG dataset (left), RSV (right)

Finally we test the MiniVGGNet architecture. The results here (Figure 5.4.4) got even better, reaching over 96% accuracy on the DFG dataset and over 82% on the RSV dataset. The effects of overfitting are way less significant, which can be attributed to the regularization that the VGG architectures introduce through the Dropout and BN layers.

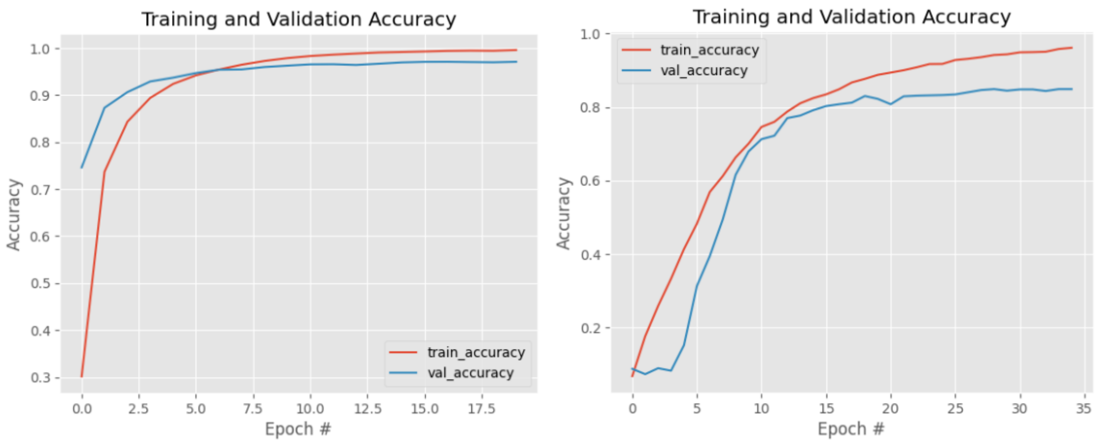


Figure 5.4.4. Training and validation accuracy over time using the MiniVGGNet architecture, DFG dataset (left), RSV (right)

Given that this is our best classification model yet, let's take a look at the other metrics such as precision and recall. The model on the DFG dataset managed to reach perfect 100% precision for most classes, 82%+ for a few and interestingly enough only one outlier of 18% precision and 100% recall, that is class 0, the I-1 sign: "Bend to the left.". A low precision but high recall means the model managed to find all instances of this class, but with a lot of false positives. Figure 5.4.5 shows some of the images that the model confused with this class, and Figure 5.4.6 shows an example of the actual class.

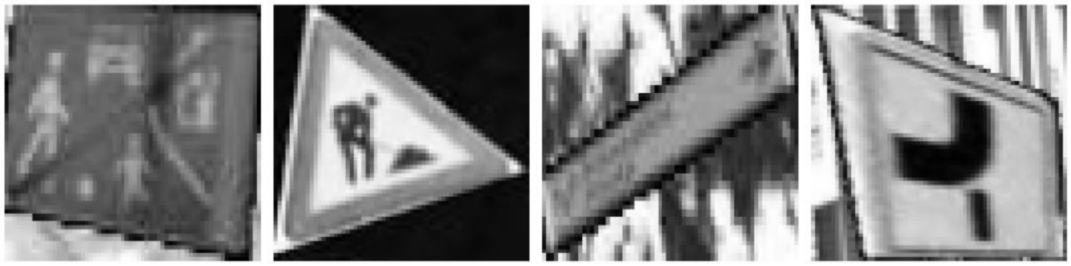


Figure 5.4.5. Images that the VGG model for the DFG dataset confused for class 0

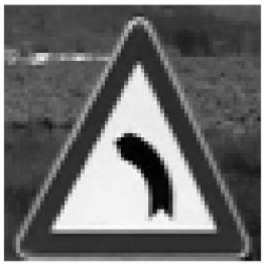


Figure 5.4.6. A correct identification of class 0

To further improve our model, we can try changing the optimizer from the classical SGD to a more sophisticated one, Adam; short for "Adaptive Moment Estimation" it is an optimizer which, rather than working with a fixed learning rate, adapts it automatically based on past gradient information. Doing this change results in a slightly higher accuracy of 97% for the DFG model (Figure 5.4.7). Furthermore, class 0's precision went up from

18% to 48% percent. Figure 5.4.8 shows a histogram for the sorted precision values for the DFG dataset. Recall scores on the other hand do not seem to show much difference; those are shown in Figure 5.4.9.

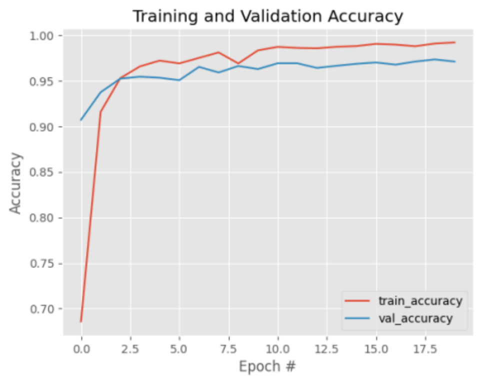


Figure 5.4.7. Training vs validation accuracy over time using the MiniVGGNet architecture, Adam optimizer, DFG dataset

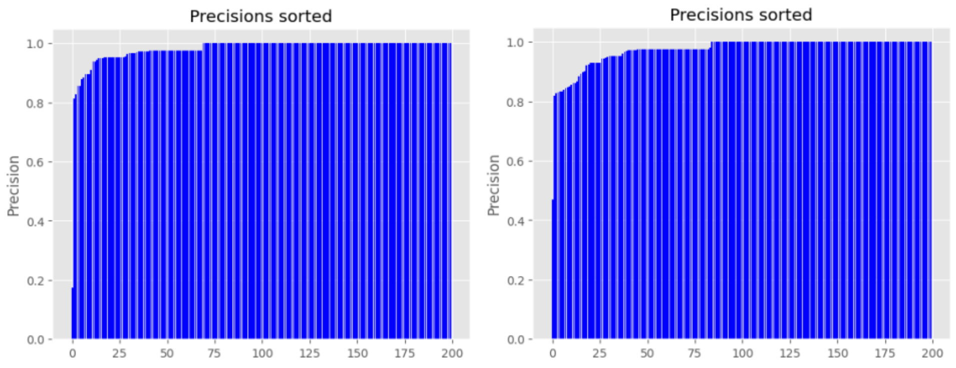


Figure 5.4.8. MiniVGGNet precision values sorted by lowest to highest for DFG dataset, SGD (left), Adam (right)

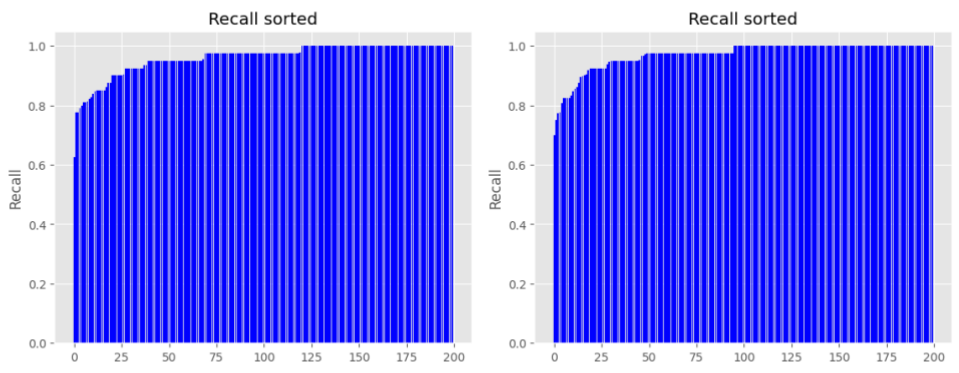


Figure 5.4.9. MiniVGGNet recall values sorted by lowest to highest for DFG dataset, SGD (left), Adam (right)

Changing the optimizer to Adam had a more significant improvement for the RSV dataset, accuracy increased to 87% from the previous 83%, as can be seen on figure 5.4.10.

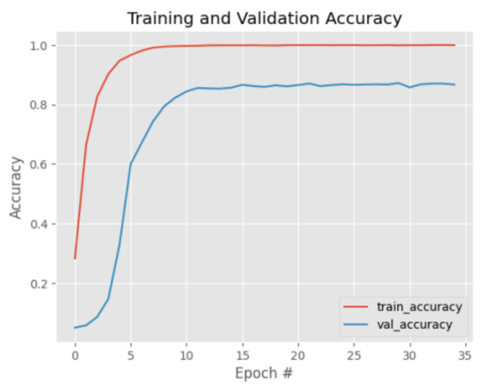


Figure 5.4.10. Training vs validation accuracy over time using the MiniVGGNet architecture, Adam optimizer, RSV dataset

Now looking at precision and recall values, the results are significantly worse than on the DFG dataset, with both SGD and the Adam optimizer. Certain classes have 0 precision and recall, which is attributed to the small number of samples they have in the testing set, that is less than 10 samples. Recall section 3.2. in which it was shown that the dataset was also heavily imbalanced. Figures 5.4.11 and 5.4.12 show the recall and precision values for both the SGD and Adam optimizers. You can see the lowest recall values are 0 and then less than 30% for a good chunk of classes, with Adam having better results.

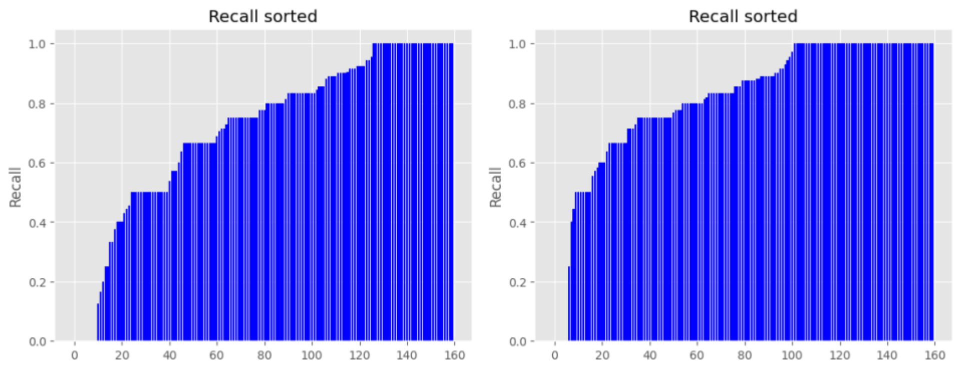


Figure 5.4.11. MiniVGGNet recall values sorted by lowest to highest for RSV dataset, SGD (left), Adam (right)

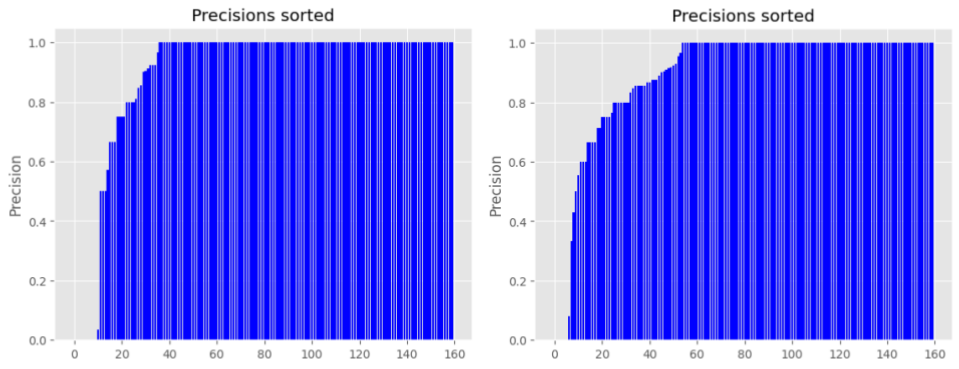


Figure 5.4.12. MiniVGGNet precision values sorted by lowest to highest for RSV dataset, SGD (left), Adam (right)

The precision values have shown better results (Figure 5.4.12), indicating that the model is guessing positives with a high accuracy but not finding all of them, thus having a high number of false negatives for those classes. And while the smallest precision values are higher for the Adam optimizer, overall SGD has more classes with 100% precision.

5.5. Object detection

In our next experiments we will focus on training an object detection model. Object detection involves not only classifying but also finding where an object of a certain class is located. As explained earlier we will be using an object detection library for Python, Detectron2.

All we need to do to start working with Detectron2 is provide the dataset path and the COCO annotations file. We will specify the number of epochs as 500 to start, and see how the results during training are to determine if this number is too big or too low. As is common with object detection using Faster R-CNN, it is initialized with weights from a pre-trained model, in this particular instance we'll be using weights from ResNet50 trained on ImageNet.

On the DFG dataset the model starts with a total loss of around 5.656 on the 20th epoch and quickly drops to around 1 in about 100 epochs, however after this point it starts to fluctuate, going up and down between 1 and 2, eventually settling to 1.838 at the end of the 500 epochs. It is clear (and expected) the model trains very slowly, it took around 6 minutes to run this experiment on the Google Colab GPUs. Table 5.1.1 shows some of the loss values for select iterations.

Epoch	total_loss	loss_cls	loss_box_reg	loss_rpn_cls	loss_rpn_loc
19	5.656	5.452	0.144	0.06291	0.007046
99	1.091	0.8564	0.2695	0.02669	0.004603
139	1.601	0.9981	0.5296	0.01305	0.005976
199	1.781	1.097	0.6732	0.01029	0.00593
219	1.708	1.053	0.6789	0.00457	0.003498
379	1.925	1.156	0.7369	0.005239	0.00518
499	1.838	1.122	0.6963	0.005557	0.0049

Table 5.5.1. Object detection model progress for a few chosen epochs, original DFG dataset

Here, **loss_clss** refers to the classification loss, that is how well the model is predicting the correct class of the detected objects. **loss_box_reg** refers to the bounding box regression loss, how well the model is predicting bounding box coordinates. **loss_rpn_cls** and **loss_rpn_loc** are metrics related to the first stage of the model, RPN. **loss_rpn_cls**, the RPN classification loss, shows how well the model is classifying region proposals into foreground and background, **loss_rpn_loc** shows how well the model is generating relevant region proposal coordinates. Finally, **total_loss** is an average of all the other values.

As can be seen in the table the loss at the end of 500 iterations is quite high and there are fluctuations in the way. The fluctuations are mostly attributed to the second stage, as the RPN stage has a consistently falling loss.

Google Colab offers limited GPU usage, averaging to about 3 hours per session. To simplify the problem and achieve better results in less time, we will be reducing the number of classes. We will pick only the warning signs, which have a triangular shape and should be easy for the model to learn features of. We'll write a function called filter_coco_annotations which will create a new annotations file, filtering only the selected classes. There are 33 warning signs, out of the 200 classes in total.

Now if we run the same experiment again, the improvement is visible. Table 5.5.2 shows the new results. The model now starts with a loss of 3.797 rather than 5.656 and similarly drops to 1.6 after 100 epochs. However after this point the loss starts dropping faster than in the previous experiment, reaching 1.228 at the end of the 500 epochs compared to 1.838. It is clear however that the loss still fluctuates and we need more iterations, so we'll train it for 5000 epochs in the next experiment.

Epoch	**total_loss**	**loss_cls**	**loss_box_reg**	**loss_rpn_cls**	**loss_rpn_loc**
19	3.797	3.695	0.02332	0.06041	0.004545
99	1.612	1.441	0.08656	0.02197	0.003263
139	0.6887	0.4334	0.2248	0.0103	0.002295
199	0.9231	0.5265	0.3916	0.008922	0.002485
219	0.9692	0.549	0.3995	0.00825	0.002355
379	1.268	0.6612	0.6059	0.003396	0.002825
499	1.228	0.6256	0.5847	0.001941	0.003138

Table 5.5.2. Object detection model progress for a few chosen epochs, simplified DFG dataset

After 5000 epochs, taking around 50 minutes to train, the model manages to reach a loss of around 0.2. At this point we can evaluate the model. Table 5.5.3 shows the results of the model after 5000 epochs.

AP	**AP50**	**AP75**	**APs**	**APm**	**APl**
0.588	0.693	0.690	0.006	0.037	0.680

Table 5.5.3. Object detection results, simplified DFG dataset

Note that Detectron2 uses the term Average Precision (AP) interchangeably with mean Average Precision (mAP), rather than as the per-class score it is also known as. In the table we can see the model got a 58.8% mAP score, evaluated as an average across IoU thresholds from 50-95%. Detectron2 also gives us mAP scores at 50% and 75% IoU thresholds, as well as mAP scores considering different object sizes, small, medium and large. It can be seen on the table above that large objects did relatively well (68% mAP) while small and medium ones did very badly.

We can further verify the above by testing the model on a few images and seeing the results. We will tell the evaluator to ignore predictions with a confidence less than 15%. Figures 5.5.1 and 5.1.2 show some examples of correctly detected and classified images. Figure 5.5.3 shows an example of an incorrect detection, there is in fact no road sign at that bounding box, the model got confused due to the triangular shape. And Figure 5.5.4 shows an example of correct detection, but incorrect classification, the true label should've been I-10.

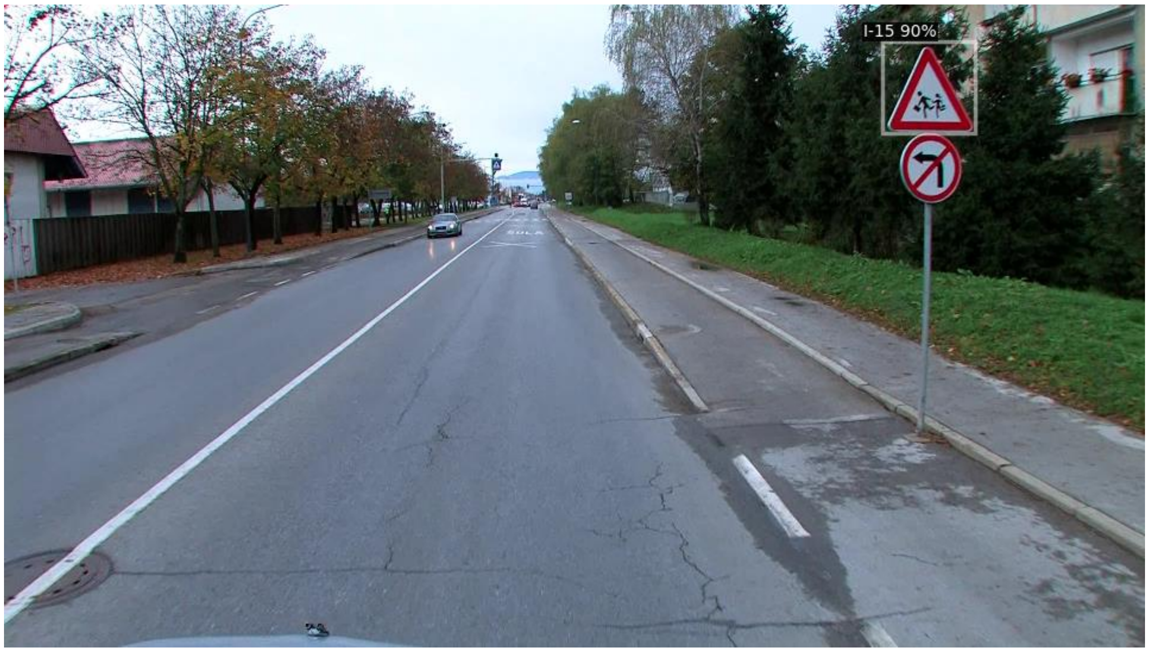


Figure 5.5.1 Example of correctly detected road sign, I-15 with 90% confidence

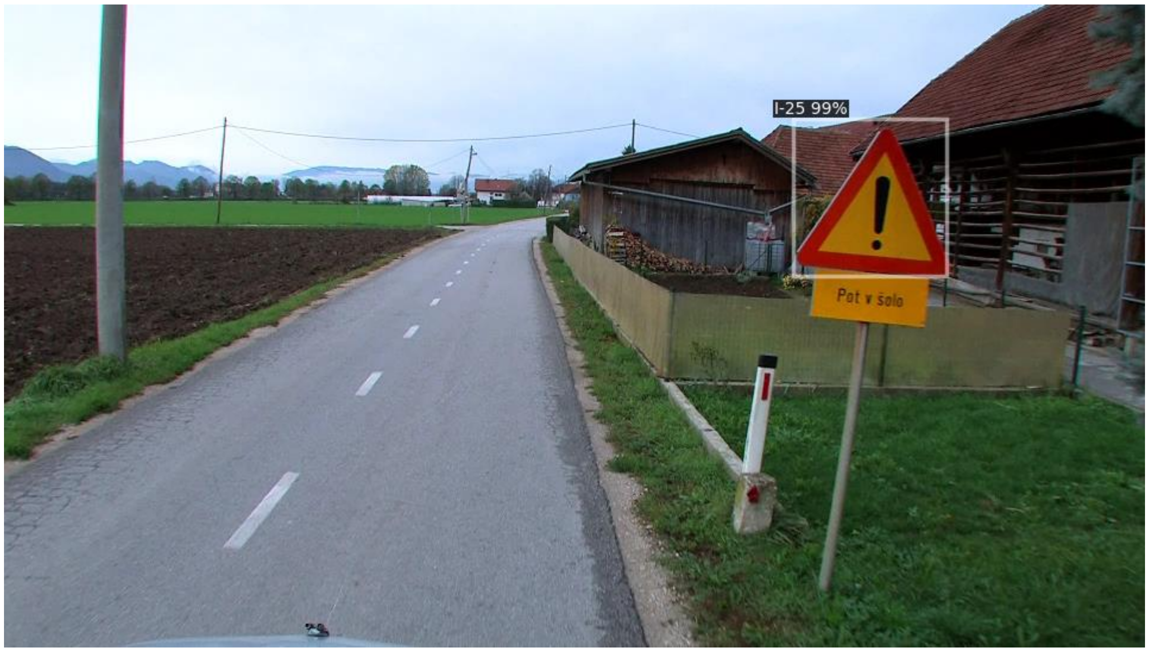


Figure 5.5.2. Example of correctly detected road sign, I-25 with 99% confidence

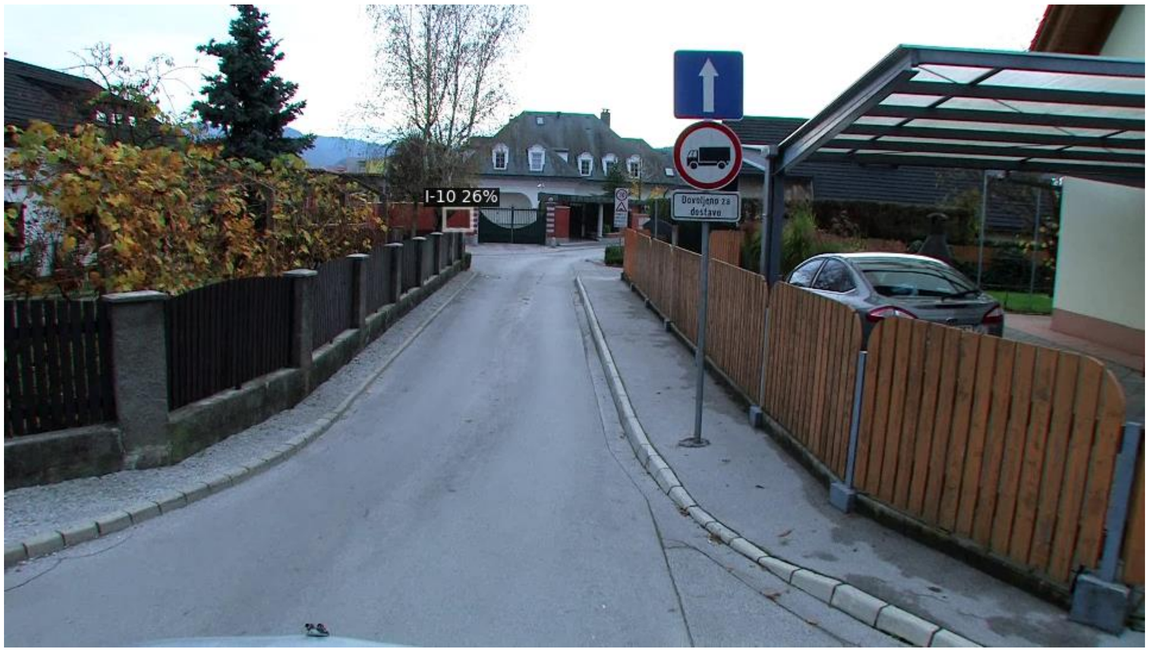


Figure 5.5.3. Example of incorrect detection

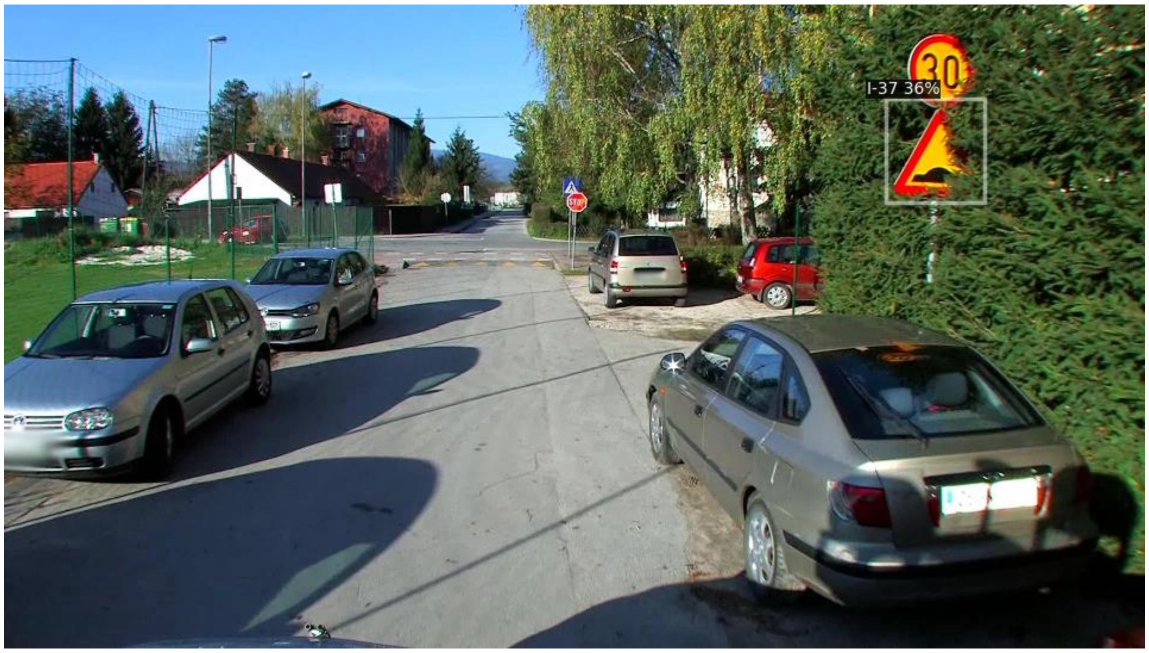


Figure 5.5.4. Example of correct detection, but incorrect classification of I-10

Now we can train a similar model on the second dataset. We will use the same configuration and similarly only train on the 33 warning signs.

Epoch	**total_loss**	**loss_cls**	**loss_box_reg**	**loss_rpn_cls**	**loss_rpn_loc**
19	3.565	3.339	0.268	0.0328	0.003781
99	1.083	0.7095	0.3948	0.02815	0.004246
139	1.297	0.739	0.5272	0.01815	0.004006
199	1.302	0.6997	0.5824	0.01244	0.003361
219	1.163	0.6406	0.5396	0.01804	0.005327
379	1.133	0.5684	0.5454	0.005425	0.002471
499	1.339	0.7073	0.6194	0.004857	0.003

Table 5.5.4. Object detection model progress for a few chosen epochs, simplified RSV dataset

This dataset behaves similarly to the first in the 500 epochs test. It starts with a high 3.5 loss, quickly drops to 1 then fluctuates before ending up on 1.339 at the 500th iteration.

We'll similarly train a model for 5000 epochs, Table 5.5.5 shows the results

AP	**AP50**	**AP75**	**APs**	**APm**	**APl**
0.364	0.487	0.464	0.176	0.341	0.501

Table 5.5.5. Object detection results, simplified RSV dataset

Just like in the classification experiments this dataset did worse on average, with a mAP score of 36.4% compared to 58.8%. Interesting however how the mAP score for small and medium objects is slightly higher; this is likely an effect of the dataset having less such small objects.

Now let’s see some example predictions.

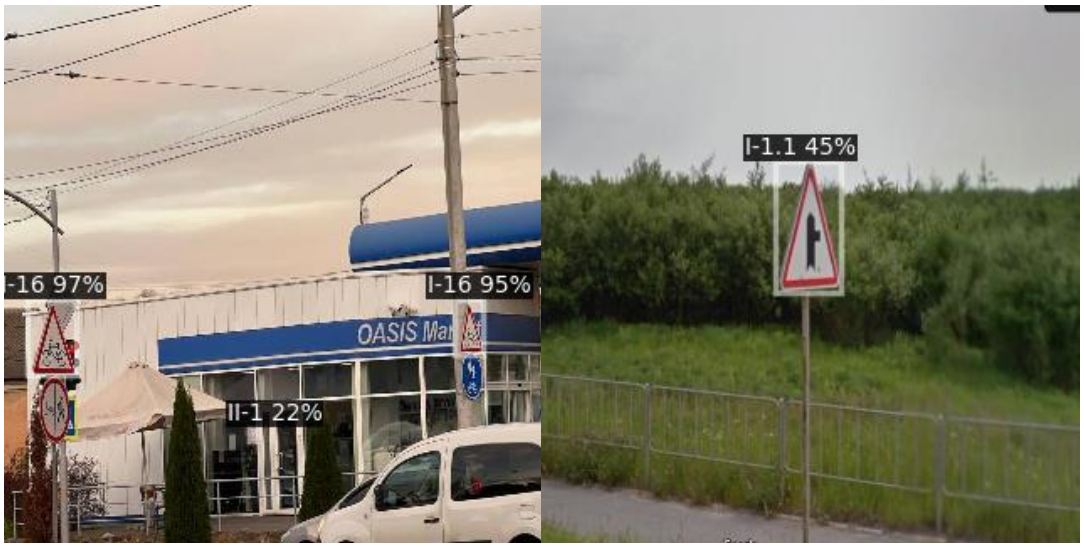


Figure 5.5.5. Example results from object detection model in RSV

Figure 5.5.5 shows some results. In the first picture you can see 3 detections, two of which are correct and have a high confidence too. II-1 is the incorrect prediction. The second figure shows a correct detection but incorrect classification, the sign on the picture is I-28.1 rather than I-1.1.

As our final experiment, we’ll try doing what we did for the classification, resizing and making images grayscale. We’ll resize the images to 256x256 from 640x640.

AP	**AP50**	**AP75**	**APs**	**APm**	**APl**
0.352	0.482	0.425	0.109	0.373	0.591

Table 5.5.6. Object detection results, simplified grayscale RSV dataset

As can be seen on Table 5.5.6, the mAP scores did not significantly change. The most significant change is the mAP score for small objects, and this is expected since we’re making images smaller, small objects will be hard to detect. However the training time was significantly lower, about 26 minutes compared to 50.

CHAPTER 6

CONCLUSION

This thesis explored the application of various deep learning models and techniques to the tasks of road sign classification and detection. The primary objective was to implement and evaluate the performance of multiple deep learning architectures on these important computer vision problems.

For the classification task, three convolutional neural network architectures of increasing complexity were implemented and tested: ShallowNet, LeNet, and a variant of VGGNet called MiniVGGNet. The models were trained and evaluated on two road sign datasets - the DFG Traffic Sign Dataset and the road-signs-vanga (RSV) dataset. The results showed that model depth and capacity had a significant impact on classification accuracy. The simple, single-layer ShallowNet struggled to learn meaningful features, achieving validation accuracies of only around 70-75% on both datasets. In contrast, the deeper LeNet and MiniVGGNet architectures attained much higher accuracies, with MiniVGGNet reaching 97% on the DFG dataset and 87% on the RSV dataset when using the Adam optimizer.

Beyond overall accuracy, class-level metrics like precision and recall were analyzed. The DFG dataset generally had very high precision and recall scores across most classes using MiniVGGNet, with a few exceptions like the I-1 sign class. The RSV dataset had more inconsistent results, with some classes having perfect precision/recall while others had scores of zero, due to the imbalanced nature of the dataset.

For object detection, the Faster R-CNN algorithm was used in conjunction with a ResNet50 backbone, leveraging pre-trained weights. Due to computational constraints, the datasets were simplified to only include warning sign classes. After training for 5000 epochs, the model achieved a mean average precision (mAP) of 58.8% on the DFG dataset and 36.4% on the RSV dataset, evaluated across IoU thresholds from 0.5 to 0.95. Performance was significantly better on larger objects compared to small and medium ones. Qualitative analysis of the detection results showed the models could often localize signs well but sometimes struggled with classification, especially for visually similar classes.

In summary, this thesis demonstrated the power of convolutional neural networks for road sign recognition tasks, while also highlighting challenges like class imbalance, detection of small objects, and classification of similar classes. Very deep models like VGGNet variants proved most effective for classification, while the two-stage Faster R-CNN framework showed promise for detection. However, the experiments also underscored the significant computational resources and training time required for object detection compared to classification.

Potential future extensions could include evaluating more modern backbone architectures like ResNeXt or EfficientNet, using techniques like FPN to better detect small objects, leveraging data augmentation to improve performance on imbalanced datasets, and exploring one-stage detectors like SSD or YOLO for faster inference. Downstream tests on embedded hardware would also be valuable for assessing real-time performance in driver assistance deployments. Nonetheless, this thesis provides a solid foundation for understanding the application of deep learning to traffic sign recognition.

REFERENCES

[1] "Summary of Motor Vehicle Crashes" National Highway Traffic Safety Administration, https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/813209. Accessed 8 May. 2024.

[2] LeCun et al. "Handwritten Digit Recognition with a Back-Propagation Network"

[3] Simonyan and Zisserman "Very Deep Convolutional Networks for Large-Scale Image Recognition"

[4] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. "Imagenet: A large-scale hierarchical image database."

[5] Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation"

[6] Girshick "Fast R-CNN"

[7] Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks"

[8] Tabernik, D. and Skočaj, D. "Deep Learning for Large-Scale Traffic-Sign Detection and Recognition"

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition"

[10] A. D. Kumar, R. Karthika, and L. Parameswaran, "Novel Deep Learning Model for Traffic Sign Detection Using Capsule Networks"

[11] Yang Gu, Bingfeng Si, “A Novel Lightweight Real-Time Traffic Sign Detection Integration Framework Based on YOLOv4”

[12] Wang et al. “Real-Time and Efficient Multi-Scale Traffic Sign Detection Method for Driverless Cars”

[13] Zhu Mao et al. “Deep Neural Networks for Road Sign Detection and Embedded Modeling Using Oblique Aerial Images”

[14] Tabernik, D. and Skočaj, D. “DFG Traffic Sign Dataset” https://www.vicos.si/resources/dfg/. Accessed 16 May. 2024.