universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

A Cluster-Driven and Satellite Image-Based Framework for Truck Activty Classification

verfasst von | submitted by

Alexander Vollstädt BSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2025

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 856
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Kartographie und Geoinformation
Betreut von | Supervisor:	Ass.-Prof. Mag. Dr. Andreas Riedl

Content

List of Figures

List of Tables

Abstract

As global supply chains grow more complex, understanding truck activity patterns through GPS data analysis has become indispensable for freight operation planning and optimization. This thesis introduces a four-step truck activity classification framework combining traditional threshold and clustering methods with novel satellite image-based classifications techniques across diverse geographical settings. A Truck Activity Classification Image Dataset (TACID) containing 1,000 satellite images per class was constructed using POI and image extraction. Two ResNet50 image classification models (three and four-class) were trained to distinguish loading/unloading, rest, traffic, and fuel stops, achieving 82% (three-class) and 69% accuracy (four-class) on the test data. A two-stage stop identification pipeline based on speed thresholding and DBSCAN clustering reduced the number of GNSS points requiring classification by 99.8%, enabling efficient handling of large datasets. The best pretrained model categorized the detected stop locations into the predefined classes, revealing important challenges due to strong inter-class similarities. Temporal analysis demonstrated that irregular transmission frequencies and large temporal gaps limit precise dwell-time calculation. Overall, the framework extends satellite image-based truck activity recognition to multi-vehicle datasets, offering a scalable methodology for freight mobility analysis while highlighting key limitations and directions for future research.

Kurzfassung

Mit der zunehmenden Komplexität globaler Wertschöpfungsketten ist es für Transportunternehmen essentiell geworden, LKW-Aktivitäten durch GPS-Datenanalyse besser zu verstehen. Diese Arbeit präsentiert einen vierstufigen Ansatz zur LKW-Aktivitätsklassifikation, der auf traditionellen Schwellenwerts- und Clustermethoden beruht. Ein spezieller Datensatz (TACID) mit 1,000 Bildern pro Kategorie wurde durch POI und Satellitenbilddatenextraktion erstellt. Zwei ResNet50 Bilderkennungsmodelle (drei und vier Klassen) wurden trainiert um zwischen Lade-, Pausen-, Verkehrs- und Tankstopps zu unterscheiden und erreichten 82% (drei Klassen) und 69% (vier Klassen) Genauigkeit auf den Testdatensatz. Durch zweistufige Stopperkennung, basierend auf einem Geschwindigkeitslimit und DBSCAN, wurde die Anzahl der zu untersuchenden GNSS Punkten um 99.8% reduziert. Das Beste vortrainierte Modell machte bei der Klassifizierung der Stopps starke Ähnlichkeiten zwischen Klassen offensichtlich, die eine eindeutige Zuordnung schwierig machten. Die zeitliche Analyse der LKW-Aufenthalte zeigte, dass unregelmäßige Übertragungsraten und lange zeitliche Unterbrechungen bei der GPS-Signalübertragung, die genaue Berechnung von Stoppzeiten stark erschweren. Alles in Allem hat diese Masterarbeit bewiesen, dass Satellitenbild-gestützte Aktivitätserkennung auf große Datensätze, die mehrere Fahrzeuge abdecken, anwendbar ist, und somit eine skalierbare Methodik für Mobilitätsanalysen präsentiert, welche sowohl Einschränkungen, aber auch potenzielle Ansätze für zukünftige Studien hervorgehoben hat.

Preface

Before you lies the master thesis “A Cluster-Driven and Satellite-Image-Supported Framework for Truck Activity Recognition“. I completed this dissertation to fulfill the graduation requirements of the Cartography and Geographic Information Science Master program at the University of Vienna. I worked on this study from February to December 2025.

During my studies I got introduced to various programs and approaches in GI Science. By writing this thesis and conducting the necessary analysis, I chose a subject which required skills I did not yet have. I have created a distinct dataset for machine learning purposes by extracting geospatial data from different data providers. The training of the image recognition model for this study confronted me with challenges I had not encountered before. By implementing the data processing pipeline in Python, I gained more experience with this powerful tool I was only familiar with before. Numerous struggles along the way made me realize the complexity of writing a scientific study, but also taught me valuable lessons both personally and professionally.

I want to thank my old boss, Martin Schweizer, for coming up with the underlying idea and granting access to real world truck data for this thesis. With operational application in mind, he motivated me to develop the framework for satellite image-based truck activity classification for this thesis. Throughout this thesis I gained experience in machine learning applications and GPS data analysis, for which I am grateful. I also want to thank my supervisor, Dr. Andreas Riedl for the guidance and support during the process. Another person I want to thank is Florian Krottenthaler, a good friend of mine, who helped me by proof reading my final thesis and giving me his unbiased feedback as an experienced scientist. Finally I want to thank my family and friends for being there for me.

1 Introduction

Modern society depends on continuous access to a wide range of resources essential for daily life. Industries such as construction, electronics, chemicals, and agriculture rely on materials sourced both locally and from distant regions across the globe. As individuals, we regularly purchase goods shipped from around the world. While cross-border trade is not new, globalization has given rise to increasingly complex supply chains that are far more interconnected, sourcing goods and raw materials from multiple locations simultaneously.

Transport is therefore fundamental to economic functioning and requires careful monitoring and analysis to secure, plan, and optimize access to goods and resources. Approximately 78% of freight throughout Europe is transported on trucks (International Union of Railways, 2024), using urban and rural road networks daily. Regular stops for loading, unloading, refueling, and mandatory rest periods are inherent to trucking operations. Modern vehicles are typically connected to logistics companies through fleet telematics systems, generating substantial volumes of data that can be leveraged to identify patterns in freight transport and optimize routing and planning.

This shift towards intelligent transportation technologies creates significant opportunities for freight movement analysis. Instead of relying on traditional trajectory recordings like driver logbooks and travel diaries, researchers now have access to high-resolution data that allows inexpensive processing at a much larger scale (Choudhry & Qian, 2023). Relevant features contained in this type of data are the ping information, which includes a timestamp for each entry, and the latitude and longitude position of the vehicle at the time of the recording (Patel et al., 2022). Based on this information, interesting locations and patterns within the data can be detected through analysis.

To understand GPS trajectories, an essential step is extracting knowledge about vehicle stop events, which is indispensable for trip planning, monitoring, driver comfort, and operational efficiency (Aziz et al., 2016). Information about stop locations enables inference about stop purposes, which is crucial for transportation planning and freight movement strategies (Patel et al., 2022). Numerous approaches to extracting insights from GPS data have emerged in recent years, introducing different methodological frameworks. Despite promising results, considerable challenges remain. While GNSS data provides substantial potential for understanding freight movement patterns, it also presents significant analytical and interpretive challenges (Gingerich et al., 2016). This trade-off highlights the need for continued research on trajectory data in general and truck trajectory data in particular.

The goal of this thesis is to develop a framework for truck activity classification by analyzing fleet GPS trajectory data without additional information such as driver logs or travel diaries. The study is based on 64 separate datasets, each covering seven days of truck trajectories in raw form, acquired directly from the fleet management system of a transport company in Tirol, Austria, operating across central Europe. The underlying idea is to identify truck stop locations from raw GNSS data, to extract satellite imagery for these places, and to use an image classification model (ICM) to classify the images into

predefined activity classes. After identifying stop activities, additional knowledge about temporal distribution and frequency is derived from the processed data. Such a framework could be a valuable contribution to the field of mobility research and freight transport planning.

1.1 Background and Problem Statement

Truck activity classification has been studied extensively using various approaches. Most related work analyzes GPS truck trajectory data directly, but some frameworks incorporate additional information sources such as point of interest data (Siripirote et al., 2020), land use information (Choudhry & Qian, 2023), and road networks (Yang et al., 2021).

The inclusion of satellite imagery for truck stop classification, as intended in this thesis, was proposed by Hu et al. (2024). Their framework is divided into three steps. First, three categories of truck activity were defined: loading/unloading, resting, and driving. By extracting satellite imagery from the study area, a Satellite Image Dataset of Truck Activity (SIDTA) was created. Second, an image classification model was fine-tuned for truck activity classification using SIDTA as training data. Evaluation showed that ResNet50, a convolutional neural network, achieved 98.04% accuracy on the training dataset. Third, the fine-tuned model classified all 2,000 points from a GPS trajectory into the predefined categories, with neighboring point results used to improve identification, yielding 96.95% accuracy for single points.

This novel approach is very promising but has not yet been applied to larger datasets, covering the movements of multiple vehicles in varied rural and urban settings. Bridging this **research gap** is the aim of this thesis.

A relevant approach was introduced by Patel et al. (2022), who proposed a cluster-driven classification method to determine truck stop purposes. After detecting stop points within raw GPS data by implementing distance-based dwell-time calculations, DBSCAN was used to group neighboring activity locations into clusters. For each cluster, features such as truck counts and carrier diversity were derived to classify stops using a Random Forest classifier, achieving an overall accuracy of 83%. This framework showed that machine learning methods can handle large data volumes and accurately classify truck stop locations and types.

Gong et al. (2015) developed a two-step framework to classify activity stop locations in GPS trajectories. In a first step, a modified DBSCAN algorithm identified stop and moving points, which were then classified into activity and non-activity stops using support vector machines. Time sequence and direction change constraints improved DBSCAN results, and three sizable features per stop location were derived for classification: stop duration, mean distance to cluster center, and shortest distance to home/workplace. This method achieved 90% accuracy in identifying stop locations and 96% accuracy in classifying them.

The method presented by Hu et al. (2024) required adaptation for this study by combining it with preprocessing steps such as stop detection and clustering to determine the applicability to larger datasets and varied geographical settings. Several shortcomings of

the original approach needed to be addressed. First, the authors classified every single point of their dataset (2,000 points), which was feasible given the small sample size but would be problematic for larger datasets due to computational constraints and image acquisition rates. Another limitation was the lack of preprocessing within the proposed data pipeline. Rather than detecting stop events before classification, which is the prevalent approach in literature, the authors identify false stops after classifying, leaving room for improvement. Third, the authors did not derive any additional information, such as stop duration, from the data, despite its value for planning purposes. Including driving as a separate class for image recognition was an additional weak point, since GPS data allows to distinguish between stationary and moving sections efficiently with proper preprocessing (Aziz et al., 2016; Choudhry & Qian, 2023; Gingerich et al., 2016; Gong et al., 2014; Gong et al., 2015; Gong et al., 2018; Luo et al., 2017; Luong et al., 2015; Milaghardan et al., 2018; Nogueira et al., 2018; Patel et al., 2022; Sarti et al., 2017; Taghavi et al., 2019; Thierry et al., 2013; Wu et al., 2021; Yang et al., 2014; Hu et al., 2024).

1.2 Research Questions and Hypotheses

To address these shortcomings, the following research questions and hypotheses that define the scope of this study were established. While the primary focus concerns the applicability of such a framework, several subsidiary questions need to be considered. These questions will test whether the SIATAR framework (Hu et al., 2024) can be extended to different geographical settings and larger datasets. If the hypotheses are confirmed, the outcome of this thesis should be an optimized truck activity classification methodology based on data preprocessing and image recognition.

- ***RQ 1***: Is a satellite image-based framework for truck activity classification applicable to large datasets covering multiple vehicles operating in varied geographical settings at a local scale, including urban and rural trajectories over a one-week-period?
- **Hypothesis 1**: Data covering different geographical settings and multiple vehicles will require a more varied image dataset for model training, a method to reduce input data volume for the classification model, and an approach to identify multiple stops at identical locations.
- ***RQ 2***: Does a two-stage preprocessing pipeline for stop identification (speed threshold filtering and spatial clustering) reduce the amount of satellite images required for classification significantly?
- **Hypothesis 2**: Dividing the GNSS data into moving and stationary segments should significantly reduce the GPS points needing to be classified. Speed thresholds offer a simple but efficient segmentation method. Spatial clustering of the filtered points should reveal local patterns leading to the detection of interesting locations, further reducing necessary image data.

- ***RQ 3***: Can an ICM distinguish between loading/unloading, rest, and traffic stops with satisfying accuracy, and can it further distinguish between rest and fuel stops with similar accuracy?
- **Hypothesis 3**: The focus on diverse locations will demand heterogeneous training data, reducing the accuracy compared to the demonstrated 96% (Hu et al., 2024). However, if the data shows clear, class-specific features, the model should be able to learn differentiating with reasonable accuracy. The distinction between rest and fuel stops will likely prove to be challenging due to strong inter-class similarities.
- ***RQ 4***: Will GPS transmission frequency limit trajectory segmentation?
- **Hypothesis 4**: If the transmission frequency is constant (up to every 5 minutes), detailed segmentation should be achievable. Frequency variations and large temporal gaps would make satisfactory segmentation challenging.

1.3 Methodological Approach

To answer these questions and test the stated assumptions, a methodological framework based on established techniques from the literature was designed. The adaptation of the SIATAR method (Hu et al., 2024) required creating a multi-step structure where each part enables subsequent analysis stages. Each step relies on different data sources and methods selected specifically for each separate task and research question.

Step 1: Image Dataset Creation

The first part of the framework involved creating a **T**ruck **A**ctivity **C**lassification **I**mage **D**ataset (**TACID**) to train and fine-tune ResNet50 for classification. Several sub-steps were combined to achieve this:

- Class definition: **Loading/Unloading, Rest, Traffic, Fuel**
- **POI data acquisition** for all four classes
- **Satellite image extraction** for each class
- Visual inspection for **data cleaning**

The result was a dataset containing **1000 satellite images per activity class**.

Step 2: Image Classification Model Training

TACID was used to **train and fine-tune ResNet50** to recognize and predict activity classes. This process was conducted separately for two configurations to address RQ 3:

- **Three-Class** Model: Loading/Unloading, Rest, Traffic
- **Four-Class** Model: Loading/Unloading, Rest, Traffic, and Fuel

Step 3: Data Classification Pipeline

With the trained ResNet50 model, the next challenge was extracting stop events and locations from raw GPS data and determining the activities performed at these places. This involved:

- **Data preparation**

- **Speed threshold** application for stop candidate detection
- Stop candidate **DBSCAN** clustering for stop identification
- Satellite image extraction and **activity classification** using the fine-tuned models for cluster centroids

***Step 4*: Temporal Pattern Analysis**

The final processing step was designed to derive interesting temporal patterns regarding stop duration and frequencies, if possible.

- **Dwell-time** calculation

Since Hu et al. (2024) demonstrated that image classification models can identify truck activity from GPS trajectory data, this thesis does not aim to further validate this framework, but rather to optimize the process regarding input data volume so the framework can analyze larger datasets in various geographical settings. The main research question (RQ 1) and corresponding hypothesis depend on the complete framework. Step 2 addresses RQ 3, revealing whether the traffic class is recognized correctly and whether ResNet50 can distinguish rest from fuel stops. RQ 2 is investigated in Step 3, testing the data reduction hypothesis. Postprocessing in Step 4 examines the impact of transmission frequency on stop duration calculation and visiting pattern recognition, addressing RQ 4.

1.4 Thesis Outline

Following the introduction, Chapter 2 reviews state-of-the-art research on the topics of image dataset construction, image classification using convolutional neural networks, and truck activity classification, highlighting approaches and methods relevant to this study. Chapter 3 describes the methods applied for each step within the data processing pipeline, following the structure of the developed workflow. This includes the creation of TACID, depicting the POI and satellite image extraction for each activity class. The Three-Class and Four-Class Model training parameters and strategy will be explained before introducing the GNSS data processing pipeline, including speed threshold and DBSCAN application and temporal analysis. Chapter 4 presents results from each section with supporting visualizations. It focuses on inter-class similarities within TACID by highlighting image examples for each class. The Model's struggle with the rest and fuel class, leading to 82% (three-class model) and 69% (four-class model) overall accuracy on the test data gets depicted. The significant data reduction achieved through data processing (99.8%), and the negative impact of irregular transmission frequency and large temporal gaps on temporal segmentation are also highlighted. Chapter 5 puts these results in context, interpreting the relevance and highlighting limitations and implications of the findings. Finally, Chapter 6 will answer the research questions and suggest directions for future research on satellite image-based truck activity classification.

2 Literature Review

This chapter reviews the scientific literature underlying the methodological framework developed for this thesis. It is structured according to the three main components of the approach: image dataset construction for deep learning applications, image classification using convolutional neural networks (CNN), and truck activity classification using trajectory data. Each section examines relevant methods, highlighting strengths and limitations that influenced the decisions for system design.

2.1 Image Datasets

Image datasets play a central and crucial role in machine learning applications and remote sensing. They are collections of labeled and standardized images that make training, evaluating, and comparing image classification models such as CNNs possible. Without such datasets, meaningful research and reproducibility of scientific findings would not be possible. In remote sensing, they are used for various tasks such as earth observation and classification of land use and land cover (Cheng et al., 2017). A specific dataset was constructed for this thesis, which is why this section will give a brief overview of the relevant literature.

2.1.1 Dataset Construction

A variety of image datasets have been generated by remote sensing and computer vision scientists over the last decades, which helped in advancing the field (Hu et al., 2024). Usually, the process consists of a few steps, such as label definition, data acquisition by crawling the web, and image annotation, in order to create a highly accurate dataset (Russakovsky et al., 2015).

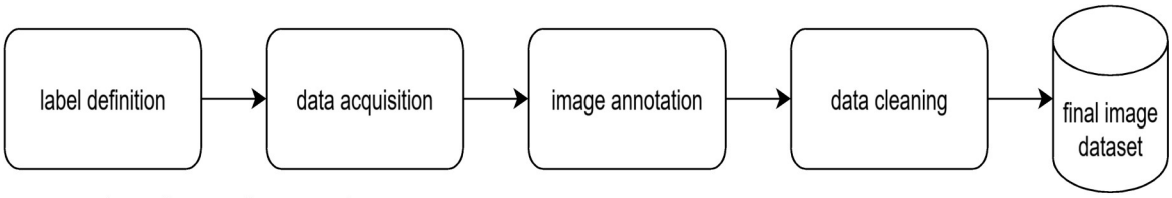

Figure 1: Classical approach to image dataset creation

The first step of choosing labels for dataset construction depends on the intended application of the images, and is conducted by analyzing existing similar sets of imagery to form widely used categories. Data acquisition for remote sensing projects relies on satellite imagery such as Sentinel-2 or SPOT data, or aerial imagery from providers like Google Earth (Cheng et al., 2017). Concerning annotation of large datasets for machine learning, researchers typically rely on humans to verify each candidate image. This is often achieved using the service of Amazon Mechanical Turk (AMT), an online platform that offers tasks for users to complete while getting financial benefits (Deng et al., 2009), or is conducted in a very time consuming way by the researchers (Helber et al., 2019). Another time-consuming and challenging task is the visual inspection that is necessary to spot poor-quality images containing cloud cover, strong reflections, and mislabeled scenes. This is usually performed manually and needs to be double-checked for

meaningful results (Zhu et al., 2019). Then, especially in remote sensing applications, other processing steps such as atmospheric and radiometric correction are applied (Sumbul et al., 2019). Data quality is a challenge when it comes to dataset construction and minimizing labeling errors, which is why data cleaning can be useful when curating a dataset (Chollet, 2021, p. 142).

2.1.2 Dataset Examples

An example of this kind of dataset is ImageNet, a large-scale image database designed to provide input data for machine learning and object recognition purposes. It consists of twelve sub-trees with 5,247 synsets and a total of 3.2 million images. After candidate image collection by means of searching multiple image search engines, the dataset was cleaned and annotated using AMT and is widely used for object recognition and image classification tasks (Deng et al., 2009). MS COCO, containing 91 object types with a total of 2.5 million labeled instances in 328,000 images, is another example of a deep learning image dataset used for recognition and classification training (Lin et al., 2015). A dataset specifically for remote sensing purposes is EuroSAT, a patch-based land use and land cover database built with Sentinel-2 imagery. It consists of 27,000 labeled and georeferenced images covering ten land cover and land use classes and 13 spectral bands (Helber et al., 2019). BigEarthNet is another Sentinel-2-based image dataset providing the scientific community with a significant amount of satellite imagery. 590,326 multi-class image patches derived from the CORINE Land Cover database provide a solid foundation for deep learning model training (Sumbul et al., 2019). An additional dataset worth mentioning is the Aerial Image Dataset (AID) that contains 10,000 aerial images representing 30 classes and was constructed for image scene classification (Xia et al., 2017). In the context of this thesis, there is one more image dataset that needs to be added to the list: the satellite image dataset of truck activity (SIDTA) introduced by Hu et al. (2024). This dataset, designed especially for truck activity classification, was constructed by defining class labels, extracting POI data for these categories, and collecting satellite images for these places. With 1,000 images per activity class, this dataset was used to train an image classification model to distinguish between predefined activity classes. This approach seemed very promising and was incorporated into the workflow of this thesis.

Image Dataset	Content	Application
ImageNet	3,200,000 images / 12 classes	object recognition, image classification
MS COCO	328,000 images / 91 object types	object recognition, image classification
EuroSAT	27,000 images / 10 classes	land use and land cover classification
BigEarthNet	590,326 images / 43 classes	land cover classification
AID	10,000 images / 30 classes	aerial scene classification
SIDTA	3,000 images / 3 classes	truck activity classification

Table 1: Image datasets

2.2 Convolutional Neural Networks

This section is designed to introduce a technology that is at the center of the framework designed for this thesis: convolutional neural networks (CNN). They represent a

specialized type of neural network for processing data that has a known, grid-like topology (Goodfellow et al., 2016, p. 330) and have proven to be particularly efficient in computer vision and image processing. Areas of application include image classification and segmentation, object detection, and video processing, among other fields. Their functioning is inspired by the visual cortex of primates (Khan et al., 2020), and the underlying foundation for the modern CNN framework was introduced with a back-propagation network designed to recognize handwritten digits (LeCun et al., 1989). Modern deep CNNs are especially powerful at learning because of the use of multiple feature extraction layers that learn representations from data automatically. Improved hardware technology and the access to large amounts of data have accelerated research in the field. Without needing explicit instructions, machine learning algorithms are able to learn the underlying relationships in data. CNNs are among the best for understanding image content (Khan et al., 2020), and their performance among pattern recognition systems being very good (Bengio, 2009).

2.2.1 CNN Components

One main feature of a CNN is its capacity to exploit spatial or temporal correlation in data. Its topology is segmented into multiple learning stages, and the ability for hierarchical feature extraction makes it rather popular (Khan et al., 2020). Generally speaking, a CNN consists of alternating layers of convolution and pooling followed by one or more fully connected layers just before the output layer (Gu et al., 2017).

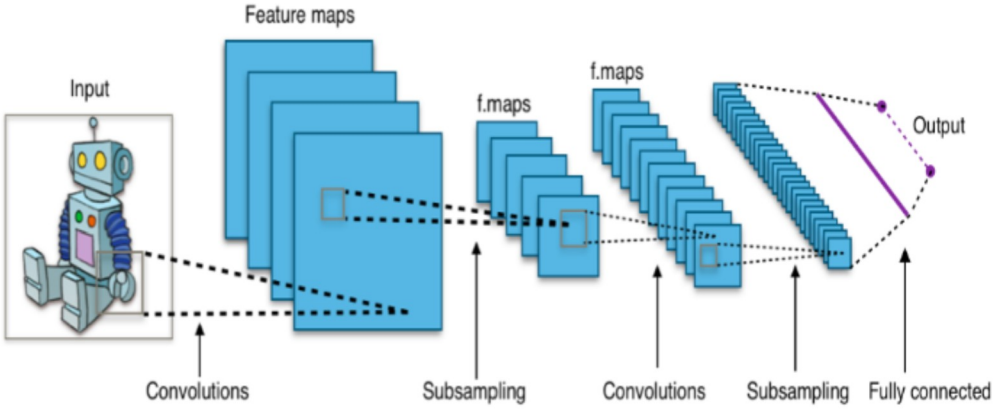

Figure 2: Typical CNN architecture (Mohamed, 2017)

Convolutional Layer

The central building block of CNNs are the convolutional layers, which are designed to learn patterns from the input data, build feature maps and pass them on to the next layers (Lee et al., 2015). What they do, is to apply convolution operations, which consist of sliding multiple small kernels or filters across small slices of the input image. These small slices correspond to the local receptive fields of the kernels-the regions of the input image that each kernel processes at a given position. These kernels have weight values, determining how they respond to different patterns assigned to them. By doing so, feature representations are detected, learned, and stacked together as a set of feature maps for

further operation (Gu et al., 2017). This process is regulated by two important parameters: stride, which is the amount by which the kernels shifts at each step, and padding that handles the borders of the feature maps in order to preserve information about the original input (Mohamed, 2017). The convolved results are then passed to an activation function like ReLU, which adds non-linearity, keeps positive values and returns a transformed output, which helps handle complex patterns and overcome the vanishing gradient problem (Khan et al., 2020).

Pooling Layer

Pooling layers are usually situated between convolutional layers and handle the task of reducing the resolution of the created feature maps and thereby improving network efficiency. Each down-sampled feature map is connected to the corresponding one from the preceding layer (Gu et al., 2017). The operation taking place is called pooling and happens locally, summing up similar information in the neighborhood of receptive fields, so that a combination of features, invariant to translational shifts and small distortions can be extracted. This process helps in regulating the complexity of the network and increasing generalization (Khan et al., 2020). By stacking several convolutional and pooling layers, the gradual extraction of higher level feature representations is possible. Common pooling operations include max pooling, where the maximum value, and average pooling, where the average value in each region of the feature map is chosen (Gu et al., 2017).

Fully Connected Layer

Fully connected layers are situated towards the end of the neural network. Depending on the architecture either one or more fully connected layers are part of the CNN. Every neuron is connected to all activations and neurons from the previous layer in order to generate global information and to perform high-level reasoning (Gu et al., 2017). By analyzing the output of all previous layers, non-linear combinations of the selected features can be computed for classification in the next step (Khan et al., 2020).

Output Layer

The output layer is the last layer within a CNN, and the one that uses all the gained information from the earlier stages to produce predictions about the detected classes (Gu et al., 2017). It uses a Softmax operation as post-processing, which computes normalized probability distribution across the target classes for the output (Goodfellow et al., 2016, p. 379).

2.2.2 CNN Training

Since the CNN implemented into the truck activity classification data pipeline was trained for this thesis, based on a hand-tailored image dataset, this next section will address image classification model training. In order to teach a model to perform the task at hand, training is necessary. The goal is that the network recognizes certain patterns within the data and makes accurate predictions. CNNs are a feed-forward multilayered hierarchical network good at learning internal representations from raw pixels, which makes them

attractive for tasks like image classification (Khan et al., 2020). Weights control what patterns the algorithm is looking for, since they contain the information learned by the model from the training data. The process behind this is gradual adjustment of these weights based on feedback data that is computed during training, which leads to gradient-based optimization of the model performance. It starts with random initialization, where the weight matrices are filled with randomly small values and then adjusted gradually by iterating through a training loop that works as follows.

- A batch of input data is passed through the model in order to make predictions, extracting features as explained above. This process is called the forward pass.
- Next the performance is evaluated by quantifying the error rate between predicted outputs and true labels by applying a loss function.
- Through a so-called backpropagation algorithm the gradient of the loss is calculated and finally the parameters for the weights are updated, thereby optimizing the model.
- This whole process is run in a loop until the loss of the model is as low as possible, and is called mini-batch stochastic gradient descent (Chollet, 2021).

In order for this learning loop to run as smoothly as possible it is helpful to apply some preprocessing to the training data. Usually the first step of data preprocessing is called batch normalization (BN), which is designed to normalize the inputs of each layer in order to have a stable distribution during training. BN has many advantages, such as reducing the dependence of gradients, stabilizing training and allowing higher learning rates without the risk of divergence (Gu et al., 2017). Since CNNs depend on large quantities of training data, a popular way to handle the scarcity of data compared to the number of parameters of the model is data augmentation, a way to transform the available data into new data without changing its nature. By sampling, mirroring, rotating, and shifting the data (Gu et al., 2017), artificial new data is generated, multiplying the amount of images by the number of augmentation methods applied (Mohamed, 2017). Another method that is helpful during model training is dropout, where the model randomly drops specific neurons, thereby improving generalization (Khan et al., 2020). This is very helpful in preventing overfitting, which is when a model learns the training data too closely. It is a powerful regularization technique and can force the network to be accurate even without certain information being available (Gu et al., 2017).

2.2.3 CNN Examples

Many different CNN algorithms were introduced over the years, advancing the field of machine learning and improving the performance of such networks. The evolution towards deeper and more efficient versions will be shown by a selection of a few milestones within the state-of-the-art research.

LeNet

The first CNN that showed state of the art performance was LeNet-5 (LeCun et al., 1998). It is a 7-layer-deep neural network designed for digit recognition with three convolutional

layers followed by two pooling or subsampling layers and one fully connected layer before the output layer (Fig. 3). It was designed to classify digits without being affected by small distortions, rotation, and variation of position and scale. LeNet was the first CNN, which reduced the number of parameters compared to other networks at the time, and was able to learn automatically from raw pixels (Khan et al., 2020).

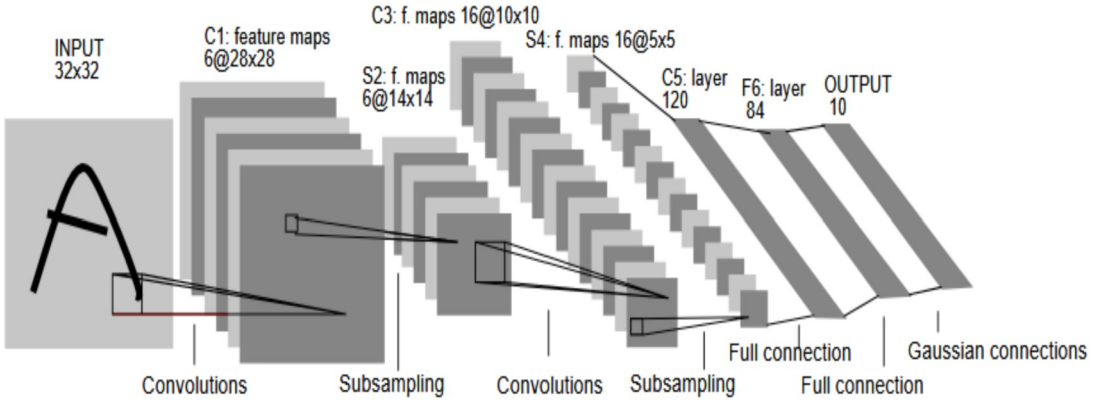

Figure 3: LeNet-5 architecture (Le Cun et al., 1998)

AlexNet

With LeNet being considered the first groundbreaking CNN for digit recognition, AlexNet (Krizhevsky et al., 2012) was the first deep CNN architecture advancing the technology of image classification and recognition. By applying parameter optimization strategies the authors were able to improve the learning capacity of the model (Khan et al., 2020). The network has 60 million parameters and 650k neurons and consists of five convolutional layers, some followed by max-pooling layers and three fully connected layers with a final 1000-way Softmax operator for classification (Krizhevsky et al., 2012). As a result of the efficient learning approach of AlexNet, it had a significant impact on the field of CNNs and has started a new era of research in the architectural advancements of these networks (Khan et al., 2020).

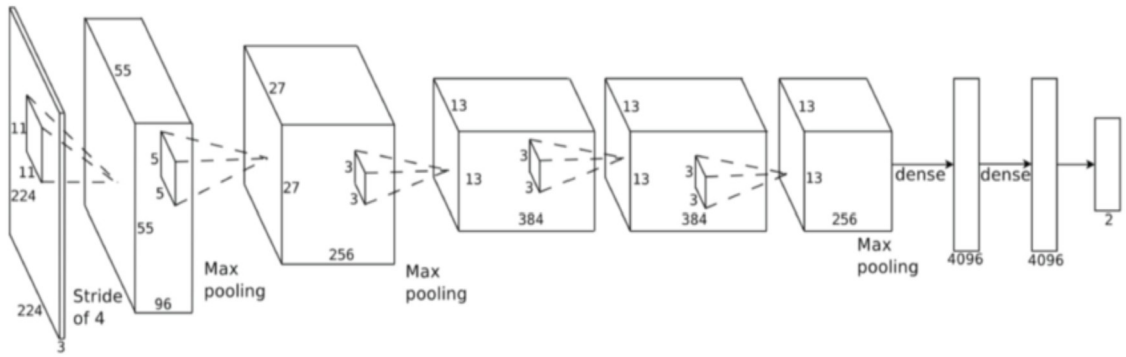

Figure 4: AlexNet architecture (Pedraza et al., 2017)

ResNet

Another big advancement in CNN architecture was introduced by adding the concept of residual learning (He et al., 2015). With deeper networks, training actually becomes more

difficult because information is lost as feature maps are passed from layer to layer (Nielsen, 2018). By reducing this vanishing gradient problem of deeper networks, ResNets apply a residual process and thereby have better classification performance. The architecture stacks convolutional layers into blocks for learning and extracting features while only some perform down-sampling (Adegun et al., 2023). Shortcut-connections were integrated into the algorithm, allowing information to skip one or more layers, adding neither extra parameters nor computational complexity (He et al., 2015). Being 20 times deeper than AlexNet (Khan et al., 2020), ResNet was able to outperform other networks and won several first places on competitions in image classification and other tasks (He et al., 2015). This model architecture has proven its ability to learn truck activities (Hu et al., 2024), which is why this image classification model was used for this thesis.

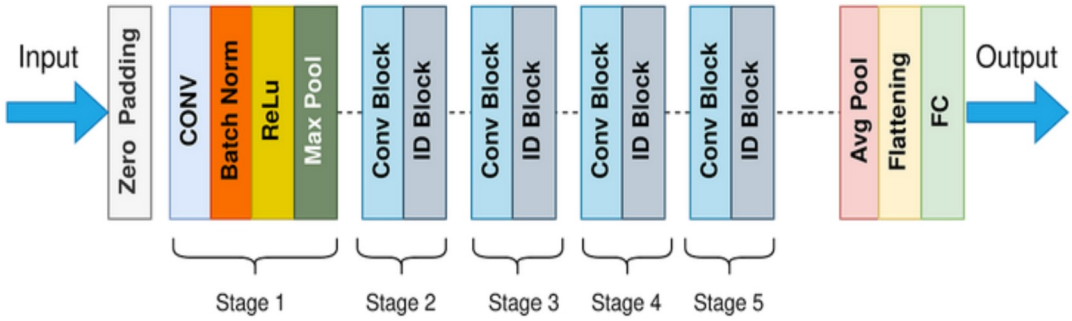

Figure 5: ResNet50 architecture (https://images.app.goo.gl/exhtJNgffsfQF5Ws5)

2.3 Truck Activity Classification

As described in Chapter 1, a crucial part of the framework designed for this study was the analysis of truck trajectory data in order to extract knowledge about stop locations. Most studies working with GPS truck data apply a two-step process for truck activity identification. Stop detection is conducted in a first step, and stop classification in a second step, which is why this section of the chapter is structured accordingly.

2.3.1 Stop Identification

The first necessary task to gain information from truck trajectory data is to identify smaller segments within the continuous stream of GPS signals. Stops are assumed to be obvious signs of these segments (Gong et al., 2018). The literature agrees that a stop event is represented by a group of chronologically consecutive signals within the data, fulfilling further spatiotemporal locality constraints (Sarti et al., 2017). Approaches for stop detection vary considerably. The overall goal of this step is to detect locations and time corresponding to stationary events of vehicles, inherent to the data.

Threshold-Based Methods

A broadly used method to detect stop events is to use certain predefined thresholds, such as vehicle speed, dwell-time, distance or acceleration. Speed thresholds are rather popular, since the idea that a truck is stopped at speed zero is quite compelling (Choudhry & Qian, 2023).

Speed Threshold

This approach consists of defining a ***speed limit***, below which the truck is considered to be stationary. Speed values of **1**, **5**, **8** and **14** kilometers per hour (km/h) have been applied (Hu et al., 2024; Yang et al., 2014; Aziz et al., 2016; Choudhry & Qian, 2023). Yang et al. (2014) were able to show that a threshold as high as 14 kilometers per hour only leads to a stop duration deviation within 30 seconds in regards to a zero km/h limit. Nevertheless, speed thresholds could lack generality throughout various methods, and especially in varying geographic contexts, such as rural and urban, since they are defined empirically (Hu et al., 2024). This easy-to-apply threshold method has its challenges, since a single value may not work in different settings. Speed is mostly inferred which makes it hard to accurately quantify it and also the quality of this approach may depend on factors unknown, making it a subjective task to set the threshold (Choudhry & Qian, 2023).

Dwell-Time Threshold

Another threshold-based technique is to apply a dwell-time duration limit. By analyzing the temporal signatures in combination with the geographic distance, sets of points that belong together as a stop event can be identified (Aziz et al., 2016). For a given set of points, the distance between each point and, at the same time, the elapsed time between each ping is calculated. With the first observed point the cumulative time is set to zero and for each following point that lies under a predefined distance threshold, the time is accumulated. This is done until the distance of the next point exceeds the distance limit. This means that all points within the expected threshold should be grouped together as a stop. This method is therefore able to include points to the stop even if the vehicle makes minor movements within the range of a property. This approach was introduced by Gingerich et al. (2016) with a 250m distance for the threshold and is seen as an adequate limit (Patel et al., 2022). Various studies either mention or directly apply this method for stop detection (Yang et al., 2014; Gingerich et al., 2016; Choudhry & Qian, 2023; Gong et al., 2014; Wu et al., 2021; Siripirote et al., 2020; Hu et al., 2024). Choudhry and Qian (2023) introduced an interesting framework, implementing a device matching algorithm to reconstruct long-term truck trajectories despite ID resets. They then developed a data processing pipeline for stop detection, using speed and dwell time thresholds. These stops were then grouped together with a clustering algorithm, followed by the identification of hubs and tours as last step of the analysis.

Density-Based Methods

Density-based stop detection is another well-established method for GPS data analysis, finding application in numerous studies. Clustering algorithms, such as DBSCAN and modified versions of it, are implemented to detect stationary events within trajectories, but also to group already detected stops into clusters (Aziz et al., 2016; Luo et al., 2017; Gong et al., 2015; Gong et al., 2018; Patel et al., 2022; Luong et al., 2015). DBSCAN is a density-based algorithm applied to detect high density areas in space (Luong et al., 2015). Points are grouped together based on two parameters: a distance threshold (Eps) and a minimum number of points needed to be considered a cluster (minPts). Locations with low density are labeled as noise (Patel et al., 2022). Instead of using the classical DBSCAN

algorithm, many scientists modified the code to fit the needs of their analysis (Aziz et al., 2016; Gong et al., 2015, Gong et al., 2018; Patel et al., 2022; Luo et al., 2017; Choudhry & Qian, 2023). One interesting example is the modification implemented by Patel et al. (2022), where the authors ensured that outliers were preserved after detection, so that locations with sparse density of stop points were not labeled as noise and excluded. This was done by grouping neighboring stops into clusters based on Eps only, not taking the minPts threshold into account, so that all stop points were kept for classification at a later stage. This approach was considered a potential improvement of the proposed framework for this thesis. Gong et al. (2015) introduced C-DBSCAN (Constraint DBSCAN), where a time sequence constraint and a direction change constraint were implemented yielding an improved algorithm achieving an accuracy of 90% in detecting stops. In another study Gong et al. (2018) implemented an entropy constraint into the earlier version of their algorithm, improving it by another 1.5% in accuracy, calling it DBSCAN-TE. Density-based stop detection has proven to be rather efficient but the method has its challenges and constraints.

Other Methods

The literature introduced other approaches to identify stop locations within GPS trajectories, some being somewhat intertwined with the categorization of stops, but they still propose techniques to locate stationary events. Taghavi et al. (2019) developed an easy-to-reapply framework using a Hidden Markov Model (HMM) to identify truck trip segments and extract activity and non-activity stops from large-scale truck GPS data while accounting for the spatiotemporal properties of GPS points. Milaghardan et al. (2018) proposed an approach for trajectory stop detection based on the Dempster-Shafer (D-S) theory of evidence, which was designed to detect stop points and associated degrees of uncertainty. Another approach was introduced by Thierry et al. (2013), where instead of grouping temporally contiguous and spatially adjacent points on a point-by-point basis, an algorithm calculating the global kernel density surface was developed to extract local density peaks as stop candidates. Even though these frameworks were able to detect stationary events within GPS data, threshold- and cluster-based methods remain the most popular (Tab. 2).

Reference	Threshold	DBSCAN	Other
Aziz et al. (2016)	✓	✓	
Choudhry et al. (2023)	✓		
Gingerich et al. (2016)	✓		
Gong et al. (2015)	✓	✓	
Gong et al. (2018)	✓	✓	
Hu et al. (2024)	✓		
Luo et al. (2017)	✓	✓	
Luong et al. (2015)		✓	
Milaghardan et al. (2018)			D-S
Nogueira et al. (2018)	✓		
Patel et al. (2022)	✓	✓	
Sarti et al. (2017)	✓		
Siripote et al. (2020)	✓		
Taghavi et al. (2019)			HMM
Thierry et al. (2013)			KDE
Wang et al. (2017)	✓		
Wu et al. (2021)	✓		
Yang et al. (2014)	✓		
Yang et al. (2021)	✓		

Table 2: Stop detection methods in literature

2.3.2 Stop Classification

Following this introduction to various stop detection approaches, this next section will focus on the second major step in GPS truck data analysis relevant for this thesis: stop classification. The goal of this step is to infer the functionality of identified stationary events. Before detailing the methodologies used for this task, it is useful to review the diverse definitions of stop categories found in the literature. Yang et al. (2014) differentiate between delivery and non-delivery stops, while Sarti et al. (2017) classify stops as either work-related or non-work-related. Other studies introduced more detailed distinctions. Gingerich et al. (2016) and Patel et al. (2022) separate stops into primary (goods transfer) and secondary (driver/vehicle needs) locations, whereas Gong et al. (2015, 2018) categorize stops into activity (work, rest, shopping) and non-activity (traffic congestion, red light) stop events. Some authors define stops in a more direct manner, defining stops as Loading/Unloading, Rest, Driving (Hu et al., 2024), or Fuel, Meal, Toll, Rest and Mixed stops (Aziz et al., 2016).

Feature Engineering

In order to perform stop classification, labels need to be designed, and features that determine which type of stop occurred need to be derived. Some of these attributes were already engineered during stop detection. Duration distribution is one of those aspects important and helpful for stop classification, depicting how long a vehicle was stationary in a certain location. Aziz et al. (2016) used the duration in combination with arrival time distribution to estimate the functionality of truck stops, assuming that these features were different for each type of stop. Another feature used for classification is carrier diversity,

which was used to derive an entropy index for each stop location, where a high number of different carriers corresponded to a high entropy score and vice versa. This approach was quite promising with a 95.8% accuracy in identifying secondary stops and showed that entropy can distinguish between primary and secondary stops, offering a novel approach for freight transport analysis (Gingerich et al., 2016). In a different approach, a specialization index (SI) was calculated to find out if a stop is rather specialized, as in specific to one kind of carrier, which would put it in the primary stop category, or diversified, which would lead to the assumption that it represents a secondary location (Patel et al., 2022). Including POI data and information into the classification process is also an approach that has been promising so far and applied by some researchers over the years. Sarti et al. (2017) considered the presence of POIs close to the detected stop locations, calculating features based on the proximity (200m) of these points of interest. Another study relying on features derived from POIs selected stop duration, average speed, the average distance between candidate stops and eleven types of POIs, the number of eleven types of POIs, the total number of POIs, and the number of road intersections as input for further classification (Wu et al., 2021). Hu et al. (2024) also used POIs in their framework, extracting satellite imagery for numerous locations based on the predefined labels of their study. Other features were computed by Yang et al. (2014), such as the distance from a stop to the city center and the distance to the closest bottleneck, whereas another approach is to derive information from road networks and land use data (Yang et al., 2021; Wu et al., 2021).

Classification Algorithms

After the relevant features are derived through various approaches as described above, the actual classification process has been attempted in many ways. One of these techniques is to implement rule-based decision making on the data, such as the threshold approaches mentioned in the previous section about stop detection. Apart from that, there is a tendency in the literature to adapt supervised machine learning algorithms for categorization of stop events. One of these is the Support Vector Machine (SVM), which is a learning algorithm that must be trained on data before performing the actual classification. It has proven to be very efficient with 96% accuracy in distinguishing between activity and non-activity stops (Gong et al., 2015), which was lifted up by 1.5% in regards to their earlier analysis a few years later (Gong et al., 2018). Other researchers used the same method, such as Wu et al. (2021), who, after identifying candidate stops with a time-distance threshold, implemented a Mobility Context Cube (MCC) as a 3D spatiotemporal model containing environmental elements for stop feature extraction, which were then used for classification with a SVM classifier. Another SVM approach outperformed threshold-based methods by learning feature interactions and proved to be resilient to urban GPS noise, contributing a robust, data-driven method for urban fright analytics and improving GPS data utility in logistics planning (Yang et al., 2014). Random Forest classification is another algorithm used by some researchers, such as Patel et al. (2022) using the DBSCAN algorithm on more than 48 million GPS pings, representing around 43k Canadian registered trucks, for location identification and then extracting features per stop for Random Forest classification. This method was able to yield an

overall accuracy of 83%. A total of 100 features, such as stop-wise features, point of interest features, cluster features and sequential features, were extracted from detected stops in a different study, to train a Random Forest model for stop classification, which then achieved a high accuracy of over 93%, thereby advancing fleet management methods (Sarti et al., 2017). Taghavi et al. (2019) came up with an easy-to-reapply framework using a Hidden Markov Model (HMM) to identify truck trip segments and extract activity and non-activity stops from large-scale truck GPS data while accounting for the spatiotemporal properties of GPS points. After model training with 50 randomly selected trajectories (over 65k data points), the HMM, a probabilistic model considering both observed sequences (GPS points) and hidden states (stop types), classified each GPS point into four different, predefined, states: activity stop, non-activity stop, traffic congestion and move (Taghavi et al., 2019). Other frameworks applied different methods, such as the Dempster-Shafer theory of evidence, which is designed to detect stop points and associated degrees of uncertainty. Spatial and temporal thresholds provided evidence for Belief or Disbelief of the hypotheses (stop or move) for point classification (Milaghardan et al., 2018). Finally a new kind of approach using image classification was introduced recently, integrating satellite imagery into the analysis instead of relying on rule-based methods. In a first step the authors created a satellite image dataset of truck activity (SIDTA) by collecting images of three types of truck activity points: loading/unloading, resting and driving (1000 images per category). This database was used for model training using three different models for comparison. ResNet50 was chosen for its efficiency and used for the actual classification of the GPS data. Post-processing was then performed by applying a static drift threshold to identify misclassifications such as trucks passing through service areas. The image classification model was able to achieve 96.95% accuracy in real-world GPS point classification outperforming traditional speed-time threshold methods and thereby offering a robust alternative to rule-based methods (Hu et al., 2024). Table 3 highlights some of the classification methods used in the literature.

Paper	Threshold	Cluster	SVM	Other
Aziz et al. (2016)		✓		
Choudhry et al. (2023)	✓	✓		
Gingerich et al. (2016)	✓			Entropy
Gong et al. (2015)			✓	
Gong et al. (2018)			✓	
Hu et al. (2024)				ICM
Patel et al. (2022)		✓		RF
Sarti et al. (2017)			✓	RF
Siripote et al. (2020)				Statistical
Taghavi et al. (2019)				HMM
Wu et al. (2021)			✓	
Yang et al. (2014)			✓	

Table 3: Stop classification methods in literature

3 Methods

After the introduction of the scientific background, Chapter 3 is designed to present the methodology used for this study. The structure of the workflow was inspired by the framework presented by Hu et al. (2024), since the goal of this thesis was to extend the approach to larger, multi-vehicle datasets, and to integrate preprocessing methods for stop detection. The methodology consists of four sequential steps (Fig. 6).

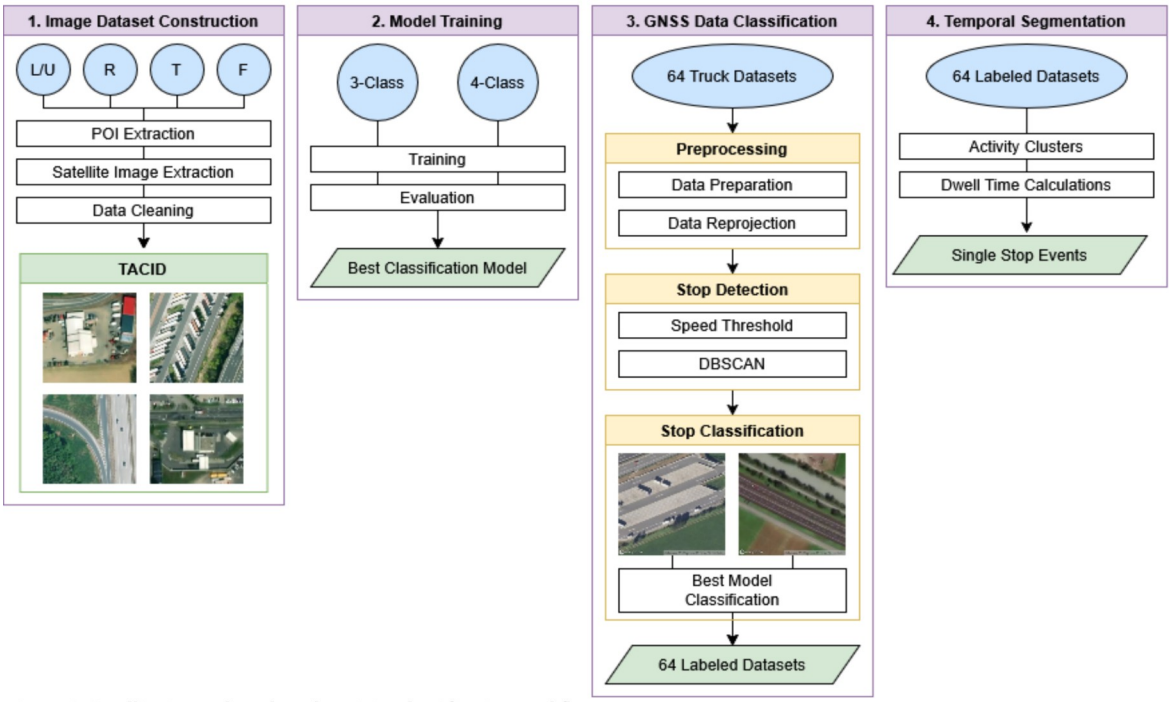

Figure 6: Satellite image-based truck activity classification workflow

These steps are interdependent, with the output of one serving as the input for the next. This chapter aims to describe the procedures selected for each part of the study, drawing on the literature reviewed in Chapter 2. It begins by explaining the techniques used for image data extraction in order to build the training image dataset TACID. After that, model fine-tuning for class recognition will be explained, followed by a methodological overview of the GNSS data stop detection and classification pipeline. Finally, it introduces the temporal segmentation approach used in the study. It will also highlight the different types of data necessary for each step of the processing pipeline. Before delving into the details of the methods, the very first section of this chapter will introduce the research area, followed by an overview of the software tools used for this thesis.

3.1 Research Area

The research area (RA) for this thesis is defined by the extent of all 64 GNSS truck trajectory files building the foundation for the analysis. The Austrian transport company that provided the data for this analysis operates across different European countries such as Austria, Germany, Italy, Slovenia, and Croatia. Many of the trucks have similar routes, making the assumption from Section 1.2 that stop locations could be reoccurring over time

very probable. Even though the area of operation of the truck fleet is limited to very specific areas of Europe, the decision was made to expand the research area for POI extraction in order to cover more space and geographical settings such as urban and rural areas for model training. It can be assumed that truck related POIs such as resting areas, fuel stations and logistics amenities have strong similarities all over Europe, which makes the decision of expanding the research area more understandable. Figure 7 depicts the extent of all different data sources used for this thesis.

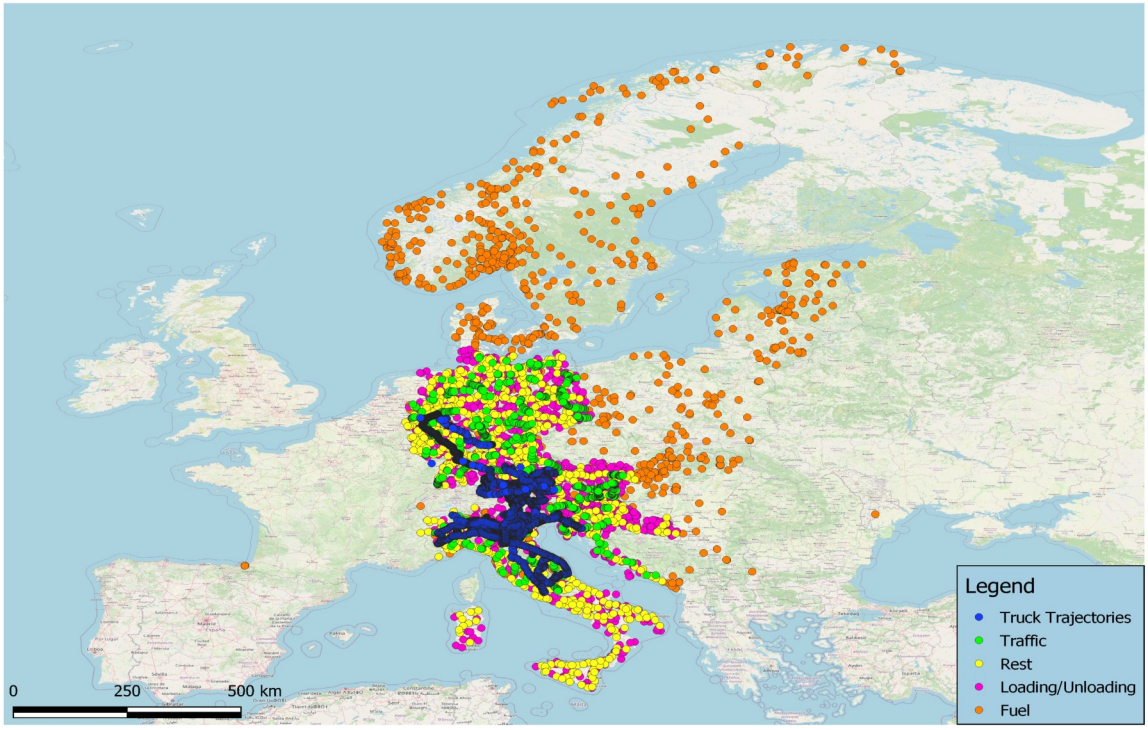

Figure 7: Research area

3.2 Software and Tools

In order to implement the methodological solutions for the proposed approach of this study it was necessary to choose the appropriate software and tools. For reasons of uniformity within the separate workflows of this thesis, it was decided to perform the different processing steps with a powerful tool able to handle multiple tasks such as data extraction, machine learning, data analysis, and visualization. The Python programming language (version 3.13.1) is such a tool, which is why it is widely used in data science, machine learning, and other applications. JupyterLab (version 4.3.4) was chosen as the appropriate interactive development environment (IDE), since it allows users to configure and arrange workflows in data science and machine learning. It makes it easy to develop step-by-step workflows that manipulate data and also visualize the results accordingly, keeping a clear structure and making specific approaches understandable. Another tool used for this thesis was the free and open-source software QGIS (version 3.28.13), used for interactive visual inspection of the processed data.

Several Python packages were employed for different tasks:

- **Pandas** (version 2.2.3): Data manipulation and analysis
- **Geopandas** (version 1.0.1): Data reprojection
- **Numpy** (version 2.0.2): Numerical computing and array operations
- **PyTorch** (version 2.7.0): Deep learning framework
- **Matplotlib** (version 3.9.2): Data visualization
- **Contextily** (version 1.6.2): Data visualization
- **Pyproj** (version 3.6.1): Data reprojection
- **Scikit-Learn** (version 1.6.1): DBSCAN clustering
- **OSMnx** (version 2.0.4): Data extraction
- **Shapely** (version 2.0.6): Manipulation of geometric objects

3.3 TACID Construction

The data pipeline for this thesis makes use of an ICM for stop classification, which had to be trained based on a truck activity classification image dataset (TACID) that was built specifically for this analysis in the same manner as SIDTA (Hu et al., 2024). Answering RQ 3 was directly dependent on the outcome of this part of the study, because this step determined the data quality for model training. The central importance of dataset generation for machine learning applications was mentioned in Section 2.1. This section will introduce how exactly this process was conducted, explaining what was done step by step. As opposed to classical approaches to dataset construction in computer vision, where classes are defined, candidate images collected, and then manually annotated (Russakovsky et al., 2015), this thesis applies a different framework, where classes are defined in a first step, POI locations are extracted for these different categories, satellite imagery representing their coordinates is acquired, cleaned, and saved into a database for model training.

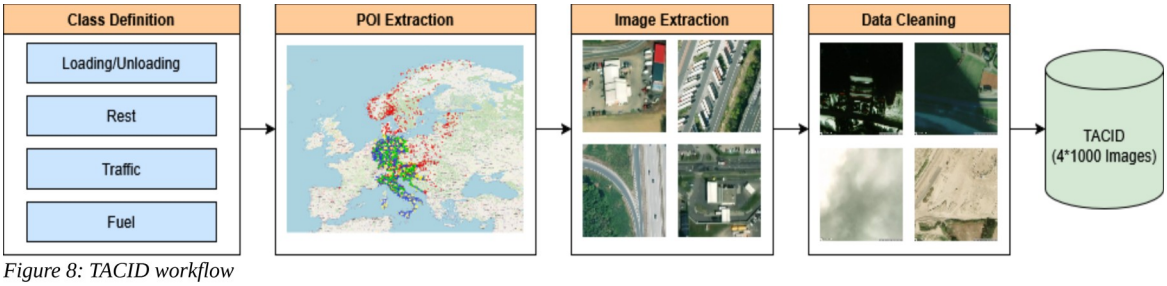

Figure 8: TACID workflow

3.3.1 Class Definition

The truck activity classes for this thesis are the following:

- **Loading/Unloading**: Truck stops for work-related reasons, such as the pick-up or delivery of goods.
- **Rest**: Truck stops for any non-work-related activity, such as taking a break, or parking the truck over night.

- **Traffic**: Stationary events because of traffic-related reasons, such as traffic congestion, red lights, roundabouts, and highway exits.
- **Fuel:** Stops to refill the truck at a fuel station.
- ***Driving***: All moments when the truck was not stationary.

Driving was kept as a class, but was excluded from the image classification process, since the focus of this thesis was put on stationary events, which is why this section will focus on the other classes. Instead, a traffic class was defined so that the third research question could be addressed. The other two classes were adopted from the baseline study: loading/unloading and rest**.** Two separate ResNet50 models were trained in order to address the second part of RQ 3. Model 1 on a three class image dataset and Model 2 was trained on four classes, adding fuel to the list, so that insights about another form of frequent truck stop can be gained.

	Loading/ Unloading	Rest	Traffic	Fuel	Driving
Three-Class	✓	✓	✓	×	×
Four-Class	✓	✓	✓	✓	×

Table 4: Model class definitions

3.3.2 POI Extraction

With the labels for the two classification models defined, the process of acquiring POI data representing these classes was performed. One of the shortcomings of the baseline study building the theoretical frame of reference of this thesis was that only data very specific to the one GPS route was acquired, yielding a dataset with very distinct class features. Such an approach might have been adequate for such a case, but for this thesis, which is covering a much more varied area, it was decided to acquire more diverse information. This was achieved using Open Street Map (OSM) POI data extracted with the OSMnx Python package. It allows to download, analyze and visualize geospatial features, such as amenities, buildings, and networks like streets and railways (Boeing, 2025). For cases where this approach did not yield fitting data, other data sources, such as freely accessible datasets were downloaded and modified to fit the RA of this thesis. Each label was treated separately to keep the data structure as clear as possible for later training purposes.

Loading/Unloading Class

In order to find locations for the loading/unloading class, web-crawling for freight-related locations was performed using a Python script specifically designed for the task. The tags for this search were all related to logistics and freight transport in the same way as introduced by Hu et al. (2024). A total of 1300 locations were extracted by crawling results one country at a time because of computational cost. The complete process of extraction and sampling happened in a reproducible manner consisting of three steps. In a first approach, all locations related to the defined tags were collected as point coordinates, yielding a total of 776,128 points of interest. To sample those locations in a spatially sensible way, a spatial grid was created and joined with the locations, keeping only two

POIs per grid cell to reduce the number of points. In a last step, a random sample of a predefined number of loading points was extracted for further use. Table 5 shows the amount of total extracted locations, the spatially stratified sample locations, and the POIs sampled for image extraction per country in order to make the selection process more understandable.

Country	Extracted Locations	Stratified sample	Image extraction
Austria	31,209	5,192	200
Germany	314,981	12,981	600
Slovenia	4,781	1,418	150
Croatia	8,163	1,693	150
Italy	416,994	4,951	200
Total	776,128	26,235	1,300

Table 5: Loading class POI data

Rest Class

Data acquisition for the rest class proved more challenging, since it was assumed that trucks operating within urban and rural areas do not necessarily always rest at highway resting areas, but at rather differing locations covering dissimilar surroundings. Link and Plötz (2024) published a paper and comprehensive dataset of real-world truck parking locations across Europe, fitting these conditions. It was systematically generated using OSM data, focusing on parking areas, rest areas, and fuel stations. The locations were filtered for truck accessibility, evaluated and then validated with special geocoding software, thereby creating a valuable resource for traffic science (Link & Plötz, 2024). This European Truck Parking Locations (ETPL) dataset was then filtered to the countries of the research area, and sampled down to 1300 locations by means of spatial stratification. This process allowed to include a diverse set of resting locations into model training.

	Total Locations	RA locations	Sample for extraction
ETPL dataset	13,323	3,640	1,300

Table 6: Rest class POI data

Traffic Class

Instead of analyzing all points from the 64 truck datasets, the focus was put on stationary moments only, which is why the traffic stop class was introduced to the analysis. In order to be able to train the classification model to recognize this type of activity, a dataset representing locations prone to traffic congestion, such as highway and road intersections, highway entries and exits, roundabouts, and traffic lights needed to be generated. Since the extraction of features such as traffic signals and intersections yielded huge amounts of information and was a time consuming process, it was decided to generate this part of the dataset manually instead. To do so, a shapefile was put together using the open source software QGIS with the ESRI Satellite basemap for orientation. This process was accomplished by selecting 1,050 frequent congestion areas in rural, but also urban

settings, thereby ensuring that a varied set of possible traffic stop locations was included into the training data.

Fuel Class

It proved challenging to extract POI data representing the newly added fuel class, since the goal was to find stations that explicitly have access for trucks. Using the tag “amenity:fuel”, even in combination with truck specific tags like “truck=yes” for data extraction with OSMnx, yielded all kinds of fuel stations, which made the results impractical for model training. So instead, a free dataset with truck fuel stations was found on the homepage of Circle K Europe. By applying filters for truck diesel and truck lane on their site finder page, it was possible to acquire a dataset with 916 stations in all kinds of settings throughout Europe. In order to increase the input data for model training it was decided to design a Python script using the OSMnx library to extract additional fuel stations on highways as point features for Germany and Italy only, since these countries were underrepresented in the dataset. The extracted locations were sampled down in the same stratified way as the loading/unloading stop locations. This process yielded a total of 1,366 fuel stations across Europe.

Dataset	Number of Fuel Stations
Circle K Europe	916
Germany	250
Italy	200
Total	1,366

Table 7: Fuel class POI data

3.3.3 Satellite Image Extraction

After acquiring the POI data, the next step was to extract the satellite imagery for these locations. The goal was to create a dataset with 1,000 images per category to enable proper model training. Table 8 displays the amount of extracted POIs and images per class before visual inspection. The platform chosen for image extraction was Mapbox, an online data provider, that allows users to extract satellite imagery through an application programming interface (API). Their static images API serves standalone, static map images generated from Mapbox studio styles and allowed to extract satellite imagery for designated coordinates. In order to get the imagery needed, the files containing the POI locations for the classes were modified, so that they only contained a list of coordinates (latitude, longitude) ready for extraction. A total of 4,920 images with 300*****300 pixels at zoom level 17 were collected and saved into a training dataset directory. Data quality was verified by visual inspection of all images, so that images showing cloud cover, barren landscapes or construction sites could be excluded from the image dataset, ensuring that only valid data was used for model training. The directory structure was set up in a way most favorable for implementation into the model training pipeline. For every class the images were divided into three subcategories: Training (70%), Validation (15%) and Testing (15%). The dataset was finally split into two separate sets, one with three classes

and the other with four, since one of the intentions of RQ 3 was to find out whether it is possible to teach the classification model to distinguish between rest and fuel stops.

Class	Number of POIs	Satellite images extracted	TACID
Loading/Unloading	1,300	1,280	1,000
Rest	1,300	1,286	1,000
Traffic	1,051	1,036	1,000
Fuel	1,366	1,318	1,000
Total	5,017	4,920	4,000

Table 8: POI and satellite image data for TACID

3.4 Model Training

With the dataset ready for model training, the next step was to decide upon the model training parameters and to start implementing the training pipeline. It was decided to leverage a pretrained model, since it is a common and very effective approach to deep learning on small datasets (Chollet, 2021, p. 224), and to do so with the PyTorch framework. Several aspects need consideration when training an image classification algorithm (section 2.2.2). This section will explain the workflow that was put together for this thesis step by step. The image classification model applied in this study is ResNet50, the same model implemented by Hu et al. (2024), and was trained for two different class combinations. Once with three classes and once with four. The training was performed on a pretrained version of the model using the initial ImageNet weights as starting parameters for the model. This means that the kernels start out by looking for patterns in the input data just in the same manner as they would while classifying images from the ImageNet dataset. In order to teach the model to classify the dataset, the approach of fine-tuning was chosen, which consists of adding new parts to an existing model, freezing the backbone of the model and then unfreezing layers to train them (Chollet, 2021, p. 234). As a new part of the ResNet model for this thesis, the fully connected layer at the end of the model architecture was replaced with a custom classification head, including dropout and a fully connected layer that produced logits for the output classes.

3.4.1 Model Parameters

Images were resized to 224*224 pixels, which is the standard size for ResNet models and normalized using the ImageNet mean and standard deviation values. Since deep learning models do not process an entire dataset at once, the data needed to be sampled into small batches for processing (Chollet, 2021, p. 35). Number and names of classes were defined and the number of epochs for the training loop was passed to the model, which define the amount of iterations over the whole training data. Another parameter that needed to be set in the beginning was the learning rate, which defines the magnitude of the move of the model parameters after an epoch. Table 9 highlights the hyperparameters that were chosen for both training models.

Parameter	Three-Class Model	Four-Class Model
Batch Size	16	16
Epochs	20	20
Classes	3	4
Learning Rate	0.001	0.001
Optimizer	Adam	Adam
Loss Function	Cross Entropy Loss	Cross Entropy Loss
Learning Rate Scheduler	Step LR	Step LR
Dropout	0.4	0.5

Table 9: Model training parameters

3.4.2 Data Augmentation

In order to handle overfitting on the training data, the training pipeline started off by applying data augmentation with a set of random transformations, including random resized cropping, horizontal flipping, rotation, color jitter, and affine transformations that aim to generate more training data from the existing training samples so that the model gets exposed to more aspects of the data for it to generalize better (Chollet, 2021, p. 221). Validation and test images were only resized and center cropped.

3.4.3 Optimization and Regularization

A StepLR learning rate scheduler was implemented to lower the learning rate after a defined number of epochs in order to make the weights change less quickly (Nielsen, 2018). The Adam optimizer, which is an adaptive learning rate optimization algorithm, was integrated into the model to add momentum (Goodfellow et al., 2016). By applying early stopping it was ensured that training stopped when validation performance degraded for a few epochs in a row. Weight decay and dropout further helped reduce overfitting and improve generalization. As means of regularization the cross entropy loss function was implemented, which is a popular function for multiclass classification problems (Nielsen, 2018).

3.4.4 Training Strategy

Two stages were implemented for progressive fine-tuning of the ResNet50 model:

- ***Stage 1: Classifier head training***: Only the parameters of the new fully connected layer were updated, while the pretrained backbone remained frozen. This helped to stabilize the classifier before fine-tuning the other layers.
- ***Stage 2: Full fine-tuning***: After Stage 1, the model weights with the best validation performance were reloaded, and the backbone of the model unfrozen. The whole network was then fine-tuned with a smaller learning rate for the backbone and a higher one for the classifier head, allowing the pretrained features to adapt to the dataset.

During training, batches of 16 images were passed through the network in a forward pass extracting patterns and producing class logits. The cross-entropy loss between predictions

and true labels was calculated and minimized through backpropagation. Gradients were used to update the model parameters with the Adam optimizer. The goal of this iterative process was to improve the classification accuracy for the predefined classes.

3.4.5 Monitoring

In order to keep track of training and validation loss and accuracy per epoch, a function was designed to track the according values along with learning rate changes introduced by the StepLR scheduler. Early stopping made sure that the training loop was ended if the validation loss stagnated for a certain number of epochs in a row, thereby preventing overfitting. This helped to implement the best model performance weights from Stage 1 into the second training stage, making sure fine-tuning was based on the best results.

3.4.6 Evaluation

In a final step after training the model was evaluated on the test data. Standard metrics like accuracy, precision, recall, and the F1-score were calculated for each class and a confusion matrix was generated depicting the model performance. The accuracy gives an overall impression on how the model performed, while precision (per class), recall, and F1-score show imbalances in model performance across classes. The confusion matrix is a useful visualization that depicts which classes are frequently misclassified.

3.5 GNSS Data Processing Pipeline

After highlighting the methodological approach to dataset construction and model training, this section will focus on the truck trajectory GNSS data processing pipeline. The steps depicted in section 3.3 and 3.4 were necessary to fully perform the truck activity classification as explained in the introduction. This part of the chapter will introduce the GNSS data and preparation, before moving on to stop detection and classification techniques. At the end of this section the temporal analysis of the stop clusters will be highlighted. Figure 9 shows a flowchart depicting the activity classification performed for this thesis.

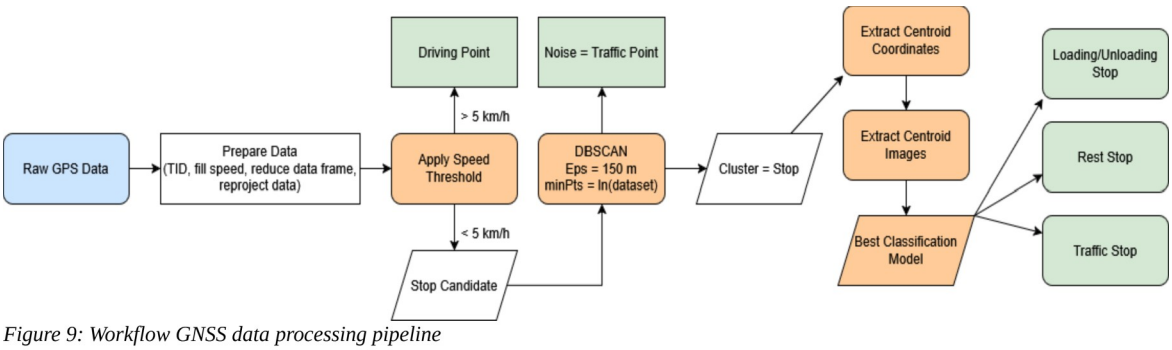

Figure 9: Workflow GNSS data processing pipeline

3.5.1 GNSS Truck Trajectory Data

The very foundation for this study is the truck trajectory data acquired directly from a logistics company located in Tirol, Austria, operating mostly throughout Austria, Germany, Italy and Switzerland. Access to their company's Telematics System was

granted for this thesis, allowing to extract time-series data for all fleet vehicles. The data contains different kinds of information, such as Global Navigation Satellite System (GNSS), Fleet Management System (FMS) and Digital Tachograph (DTCO) entries. Since there were some inconsistencies in the two latter parts of the data structure, notably a lot of missing values, the focus of this thesis was on the GNSS records, which exist in all files. Each file's GPS entry contains information about timestamp, latitude, longitude, and speed. The transmission frequency of the recordings varies from ten seconds to five minutes, and more, while most datasets have long intervals, up to several hours, in between transmissions, leading to gaps within the trajectories. The extraction of this data was performed using a Python script designed to log into and retrieve a list of devices along with their details from the fleet database, yielding 64 separate files. For each device with a recent GNSS timestamp, it fetched the time-series data covering a specified time period from the system and stored the retrieved data for each vehicle to a designated directory.

3.5.2 Preprocessing

In order to get the data ready for spatial analysis it was necessary to perform data cleaning, and to reproject the coordinates from a geographic into a projected reference system. The goal of this section is to describe the process and the methods applied to do so.

Preparation and Data Cleaning

A crucial step during data preparation was to check for systematic errors, such as very high speed values (>120km/h), and missing values in general. Raw GPS data generally needs to be examined to ensure data accuracy for the data processing pipeline (Gong et al., 2014). As mentioned above and depicted in Table 10, the focus of this analysis was put on the GNSS data (green) collected from the fleet database. FMS (blue) and DTCO (orange) entries were excluded from the whole process for lack of relevant information, and the GNSS data attributes were reduced to the necessary features with importance for this framework: timestamp, latitude, longitude, and speed.

timestamp	latitude	longitude	altitude	speed	track	poi	vehSpeed
2025-09-03 14:01:10	46.490455	11.4196366666667	306		126.0	15	0
2025-09-03 14:01:20	46.490455	11.4196366666667	306		126.0	15	0
2025-09-03 14:01:30	46.490455	11.4196366666667	306		126.0	15	0
2025-09-03 14:01:40	46.490455	11.4196366666667	306		126.0	15	0

vehDistance	engineSpeed	fuelUsed	fuelLevel	engineHours	engineTemp	cruiseControl	accPedal
926056.0	500	193548.0	56.0	15923.0	79.0	196.0	
926056.0	500	193548.0	56.0	15923.0	79.0	196.0	
926056.0	499	193548.0	56.0	15923.0	79.0	196.0	
926056.0	499	193548.0	56.0	15923.0	79.0	196.0	

axleWeight2	tachoState	driver1	workStates	driver1States	driver2States	addinfo
9857.0	192.0	[redacted]	10.0	16.0	192.0	336.0
9853.0	192.0	[redacted]	10.0	16.0	192.0	336.0
9851.0	192.0	[redacted]	10.0	16.0	192.0	336.0
9857.0	192.0	[redacted]	10.0	16.0	192.0	336.0

Table 10: Raw truck data

A thorough inspection of the data made it clear that missing values in the speed column as seen in Table 10 actually represented the speed value 0, which is why these NULL values were filled in with 0. With no missing values left within the column, an exclusion of false speed values (>120 km/h) was implemented to get rid of spatial outliers. The choice of 120 was made because of the assumption that trucks are not designed and allowed to drive at such a speed, even though the literature recommends the threshold to be set at 200 km/h (Gong et al., 2014). Since the data did not have any ID column, it was decided to add a truck specific identification number (TID) to the data frame in order to be able to convert the processed data back to the original based on ID and timestamp. Table 11 shows an example of preprocessed data without TID for privacy reasons.

timestamp	speed	latitude	longitude
2025-09-03 14:01:10	0.0	46.490455	11.419637
2025-09-03 14:01:20	0.0	46.490455	11.419637
2025-09-03 14:01:30	0.0	46.490455	11.419637

Table 11: Preprocessed truck data

Coordinate Reprojection

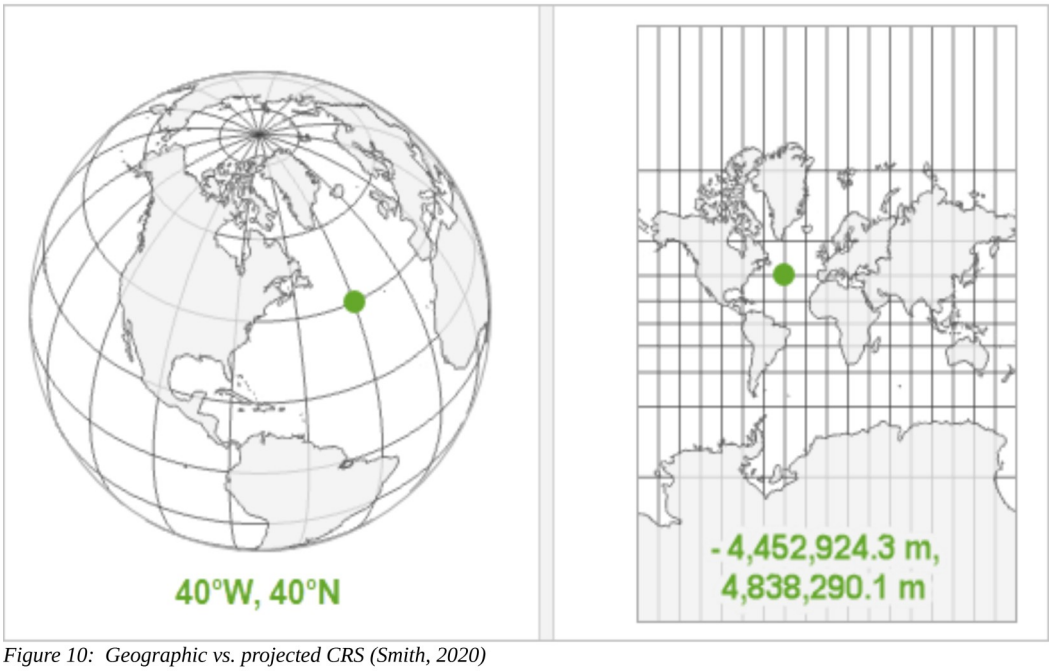

Figure 10: Geographic vs. projected CRS (Smith, 2020)

The GNSS data extracted from the fleet database had degrees of latitude and longitude based on the World Geodetic System 1984 (WGS 84) geographic coordinate system as coordinates, which is the standard reference system for GPS data. However, these angular units are not ideal for spatial computations such as distance, area, or directional analysis, since they are not using uniform measurement units. These calculations require linear units, such as meters, which are provided by various spatial projections (Smith, 2020). To make sure that all spatial analysis for this thesis was reliable and accurate, the GNSS data was reprojected from WGS 84 to the European Terrestrial Reference System 1989 (ETRS

89) projected coordinate system. ETRS 89 is specifically designed for continental Europe, thereby covering the whole research area and minimizing any computational distortions in relation to area or distance throughout the analysis. Both sets of coordinates were kept in the data frame as attributes for computation (X, Y) and for visualization (Lat, Lon) as depicted in Table 12. It becomes apparent that the projected coordinates are more consistent simply by looking at the precision of the values, which is rather varied for the geographical WGS 84 pairs.

Timestamp	X	Y	Latitude	Longitude
03/09/2025 14:02:00	4430101,011290586	2598505,7580350502	46,490455	11,419635
03/09/2025 14:57:37	4446605,4065184975	2625658,338075063	46,7318366666667	11,6418633333333
03/09/2025 15:16:42	4446598,397369126	2625670,0326614673	46,7319433333333	11,641775
08/09/2025 02:45:40	4458397,755143074	2700006,9211049927	47,39848	11,8190116666667
08/09/2025 21:32:42	4458347,296450876	2700021,055672601	47,3986183333333	11,8183483333333

Table 12: ETRS 89 (X, Y) vs. WGS 84 (Lat, Lon) coordinates

3.5.3 Stop Detection

As explained earlier and especially in relation to the first research question, this study focused on stationary moments only, so that only a fraction instead of all GPS signals had to be analyzed. As mentioned in Chapter 2, the most common method to detect stop events is based on simple threshold techniques applied to different aspects of truck operation such as vehicle speed (Choudhry & Qian, 2023). To get a better impression of this process, Table 13 shows an example of some of the GPS signals for one of the fleet vehicles. The timestamp represents date and exact time when the GPS signal was recorded, speed stands for the kilometers per hour velocity of the vehicle, TID is the identification code for the vehicle (blacked out for privacy reasons), and X, Y are the ETRS 89 coordinates in meters.

timestamp	speed	TID	X	Y
2025-07-30 13:36:08	15.0	[redacted]	4447425.037391934	2692031.8714540517
2025-07-30 13:36:18	11.0	[redacted]	4447412.105479806	2692000.4640216357
2025-07-30 13:36:28	0.0	[redacted]	4447425.777673576	2691992.2528123287
2025-07-30 13:36:38	0.0	[redacted]	4447426.529849395	2691992.45499938
2025-07-30 13:36:48	0.0	[redacted]	4447427.656088894	2691992.8508403986
2025-07-30 13:36:58	2.0	[redacted]	4447439.665543324	2691985.7134106597
2025-07-30 13:37:08	13.0	[redacted]	4447459.216528553	2691973.930733301
2025-07-30 13:37:18	15.0	[redacted]	4447498.062610432	2691950.545012672

Table 13: Preprocessed and reprojected truck data

Speed Threshold-Based Stop Candidate Detection

To exclude driving points, a speed threshold of 5 km/h was applied so that all points having a speed value over five kilometers per hour were filtered out and labeled as

'driving', leaving only potential stop candidates highlighted in yellow (Tab. 13). The reason for applying this sort of threshold was that it is frequently used and easy to apply on large amounts of data (Choudhry & Qian, 2023), and to help answer RQ 2. After selecting the points below the threshold, only potential stop candidate points were left for further analysis.

Density-Based Stop Location Identification

The next step for the analysis of the GNSS data was to apply clustering to the detected stop candidates so that single stopping points could be grouped together into clusters. This was necessary in order to answer RQ 2, since the underlying hypothesis was that this method would drastically reduce the amount of necessary satellite imagery for classification. Since the trajectories of different trucks frequently take place in the same geographical area, it was assumed that multiple stop locations would reoccur across time, space, and different vehicles. However, in order to address the overarching goal of data reduction, a clustering technique indifferent to temporal aspects of the input data had to be chosen to cluster all stops occurring at the same location into one single cluster, instead of one cluster per temporal stop. A promising algorithm for this approach was DBSCAN, which is able to detect clusters of different shape, labels areas with low density as noise, and is ideal when working with large datasets (Ester et al., 1996). By grouping stop candidate points into clusters and taking the centroid coordinates for these clusters, reoccurring stop locations could be reduced to one pair of X and Y values, thereby classifying multiple stops with the use of only one image, instead of one image per single stop. The DBSCAN algorithm relies on a set of key definitions necessary to understand what it does. All following descriptions are directly based on Ester et al. (1996):

Definition 1: (Eps-neighborhood of a point) The *Eps-neighborhood* of a point p, denoted by N_{eps} (p), is defined by

N_{eps} (p) = {q ∈ *D* | dist(p, q) ≤ Eps}.

Definition 2: (directly density-reachable)A point p is *directly density-reachable* from a point q wrt. Eps, MinPts if

1) p ∈ N_{eps}(q) and

2) |N_{eps}(q)| ≥ MinPts (core point condition).

Definition 3: (density-reachable) A point p is *density-reachable* from a point q wrt. Eps and MinPts if there is a chain of points

p_1, … p_n, p_1 = q, p_n = p such that p_i+1 is directly density-reachable from p_i.

Definition 4: (density-connected) A point p is *density-connected* to a point q wrt. Eps and MinPts if there is a point o such that both, p and q are density-reachable from o wrt. Eps and MinPts

Definition 5: (cluster) Let *D* be a database of points. *cluster C* wrt. Eps and MinPts is a non-empty subset of *D* satisfying the following conditions:

1) ∀ p, q: if p ∈ C and q is density-reachable from p wrt. Eps and MinPts, then q ∈ C. (Maximality)

2) ∀ p, q ∈ C: p is density-connected to q wrt. Eps and MinPts.

(Connectivity)

Definition 6: (noise) Let $C_1,...,C_k$ be the clusters of the database D wrt. Parameters Eps_i and $MinPts_i$, i = 1,..... k.

Then we define the *noise* as the set of points in the database D not belonging to any cluster C_1, i.e. noise = {p ∈ D | ∀ i: p ∉ C_i}.

Based on these underlying definitions, the DBSCAN algorithm requires two parameters: Eps (neighborhood radius) and MinPts (the minimum points within this radius to form a cluster). Points that have at least MinPts neighbors within Eps are considered core points, while points within the neighborhood of a core point are border points. Points not fulfilling any of these two thresholds are labeled as noise. Another advantage of DBSCAN is the fact that the actual number of clusters within the data is not necessary as it is with k-means clustering. The robustness to noise within the dataset is also advantageous.

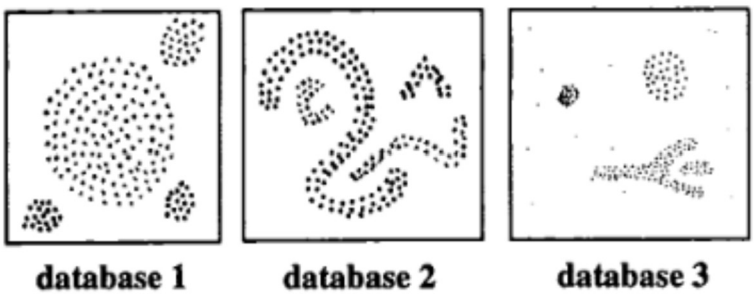

Figure 11: DBSCAN cluster examples (Ester et al. 1996)

3.5.4 Stop Classification

After identifying stop locations and clusters, the pipeline continued by extracting the stop location coordinates for satellite image acquisition. These images served as input data for the fine-tuned ICM, which classified these locations according to the activity classes defined for this thesis. Noise points were excluded from this process, since it was assumed that these low-density areas represented traffic stops.

Coordinate and Image Extraction

The DBSCAN algorithm highlighted above provided a set of stop locations with the according coordinates, but instead of handling each stop separately, the mean longitude and latitude of these clusters were chosen as representative location of the truck stop (Aziz et al., 2016). The data pipeline made sure to store all corresponding connections between stop locations and clusters, in order to trace back towards the original data frames of the fleet vehicles for proper classification of all datasets. Satellite imagery was then extracted for all representative stop locations so that the fine-tuned ResNet50 model could classify them across the predefined activity classes for the final characterization of the data. The

images were acquired in the same manner as during the construction of the TACID dataset, by using the API access to the Mapbox static images service.

Cluster Image Classification

The next step within the workflow, now that images were extracted, was to use the trained model for the final satellite image classification. The only thing left to do in this step was to run the collected imagery through the classification model in order to get the class predictions. These class labels were added to the data as a new feature so that every single cluster point was categorized. By combining these data frames with the previously excluded driving and noise points, all points were now labeled.

3.5.5 Temporal Postprocessing

The framework for this thesis contained one last processing step: the temporal analysis of the discovered stop events (Fig. 12). In order to derive visiting patterns and stop duration for the detected locations, a dwell-time calculation needed to be performed as demonstrated by Hu et al. (2024). To do so, elapsed time calculations were conducted on the labeled datasets, thereby adding a new column for the transmission frequency between GPS points, but also the elapsed time from first to last signal. It was possible to derive temporal visiting patterns with this information, by applying a cumulative process to the points within each cluster. For each point *p* in a cluster, the temporal distance *t* was checked against a dwell-time threshold. If the time elapsed between two points was below the threshold, the points were seen as belonging to the same stop and their time accumulated. If the time elapsed between two points was greater than the threshold, a new stop was detected. By iterating through each single dataset interesting patterns and limitations were discovered, which will be highlighted in Chapter 4.

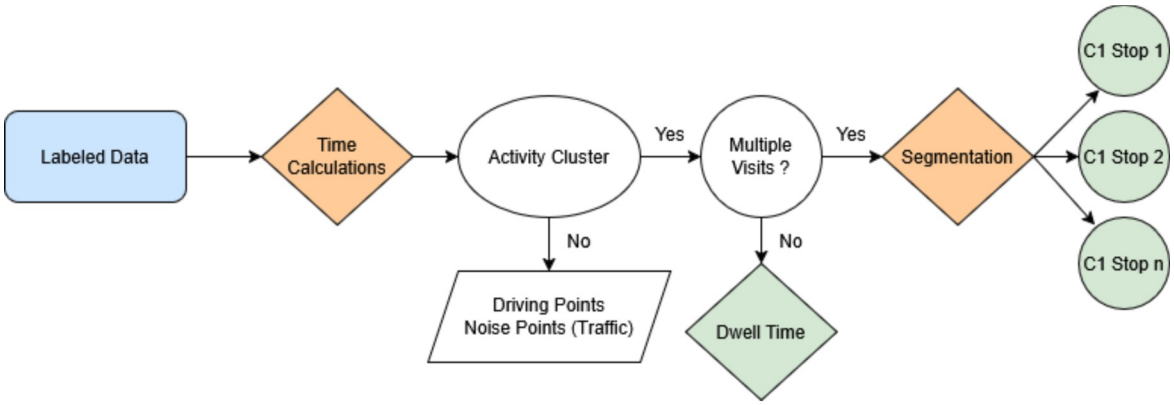

Figure 12: Workflow temporal segmentation

4 Results

The previous three chapters were designed to introduce background and motivation, to highlight the relevant state of the art literature, and to describe the methodological framework developed for this thesis. They were necessary to understand the theoretical implications inherent to the proposed approach, and to make the goal and necessity of such a study comprehensible. This chapter, on the other hand, will reveal the different

results that were produced during each separate step of the data processing pipeline. It will follow the same structure as Chapter 3, describing and visualizing the detected outcomes for each subsection. The first section will focus on TACID, depicting selected image examples for each truck activity class, and showing a few instances of unusable satellite imagery that were removed from the training data. Thereafter model training outcome will be introduced for both the three-class and the four-class models by presenting relevant performance evaluation metrics and displaying misclassified image examples for a better understanding of the challenges encountered. After that, the chapter will focus on the GNSS data workflow leading to the activity classification, which was the central goal of this thesis, and the last section will highlight the outcome of the temporal segmentation of the detected stop locations.

4.1 Dataset Construction

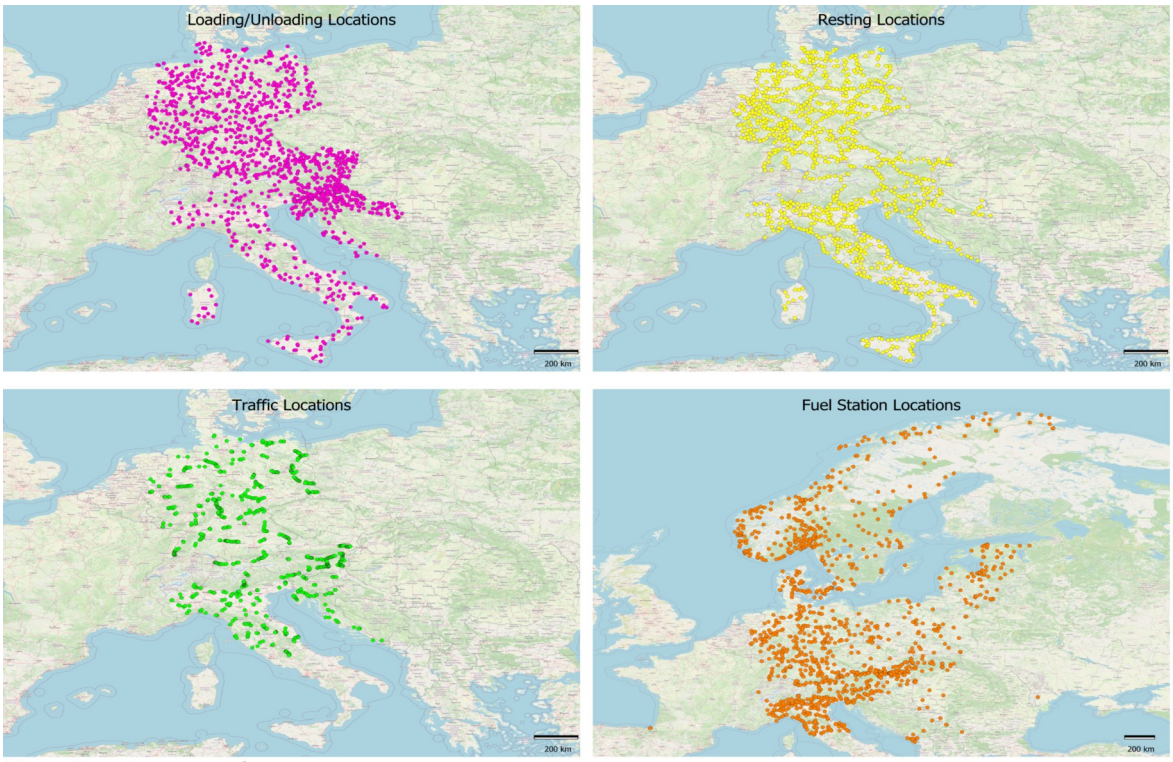

Figure 13: POI extent per class

Chapter 3 introduced the steps taken in order to construct the training dataset for this thesis. After defining class labels for classification, POI locations were extracted from OSM, and downloaded from other freely accessible sources, so that satellite imagery could be crawled through the Mapbox API, stored, and cleaned in order to create TACID. The final training dataset consisted of 4000 satellite images for the four activity classes. Each class was processed separately so that the data structure of the training dataset consisted of a clear folder hierarchy for model training purposes. The following section will depict some examples of satellite imagery corresponding to each class label. Figure 13 depicts

the spatial extent of the collected POI locations for the different activity classes that were necessary for image extraction. Loading/Unloading and resting places have very similar extents, covering the five countries that were included in the truck trajectories from the GPS data, while traffic locations were selected manually, covering scenes representing the class across large parts of the research area. The fuel class included fuel stations from northern and eastern Europe, making sure to cover a broad set of service areas accessible for trucks.

4.1.1 Loading/Unloading Class

The first class handled was the work-related ***Loading/Unloading*** class. As described in section 3.3.2 the tags defined for location extraction were varied but specific to logistics and transport, and the spatial distribution of the collected locations over the research area was visualized (Fig.13). 1,300 images were extracted from Mapbox, inspected to ensure appropriate data quality, and 1,000 scenes were kept for the dataset. Figure 14 displays a selection of the collected scenes, revealing what this activity class can look like. Some of the main features are big buildings, areas where vehicles can move along, parking spaces with parked cars and trucks, and materials in piles or close to buildings. Image resolution is not exactly the same for each scene, and some images contain water bodies and trees.

Figure 14: Loading class scenes

4.1.2 Rest Class

As mentioned in section 3.3.2, the process for rest location extraction was not conducted by crawling OSM, but by downloading the freely accessible ETPL dataset created by Link and Plötz (2024). These truck stop locations covering Europe were filtered to the five countries of the RA reducing the size of the dataset, which was then spatially sampled down to 1300 locations for satellite image acquisition. Using their coordinates for image extraction, the rest class for TACID was created, covering different types of places for truck rest (Fig. 15). The scenes visualized for this class mostly show large parking areas close to roads, but also smaller areas where vehicles can stop right next to roads. Some of these pictures show parked trucks and cars, while others have a few small buildings on the premises.

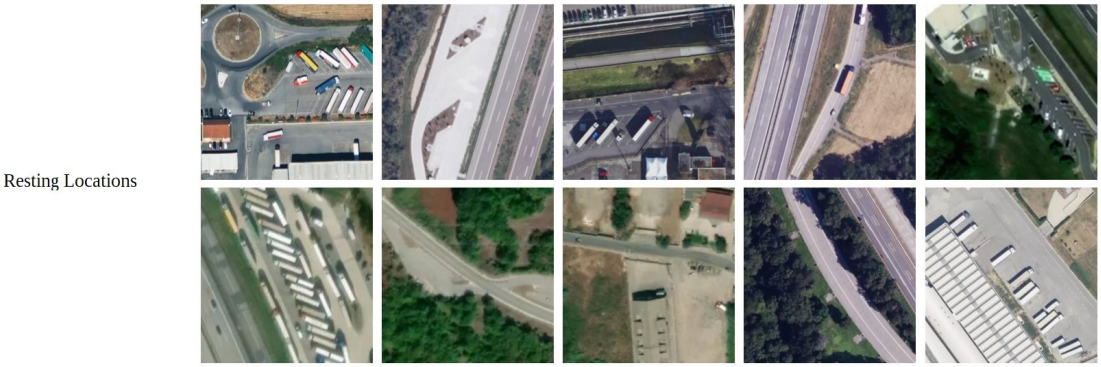

Figure 15: Rest class scenes

4.1.3 Traffic Class

The creation of the traffic class dataset was performed manually using QGIS as described in Section 3.3.2. Points were selected across the RA but not necessarily spread out evenly, since it can be assumed that roads and intersections viewed from above look very similar in different geographical settings throughout the research area. The coordinates of those points were used for extraction with the Mapbox API. Figure 16 introduces a few scenes showing the different settings that were chosen. All images contain roads, some being highways with fields and trees next to them, others showing road intersections inside populated areas. Road exits and entries are also included, as are some roundabouts.

Figure 16: Traffic class scenes

4.1.4 Fuel Station Class

The process of data acquisition for the fuel class, which was necessary to answer parts of the third research question, was conducted by downloading a Circle K dataset and extracting highway fuel stations from OSM, as described in Section 3.3.2. This class covers more area than the other three classes (Fig. 13), since the data for truck fuel stations from Circle K covers parts of northern and eastern Europe. Figure 17 shows example images for the fuel class, depicting the prevalent features. All images are centered on buildings with flat roofs which are rather small. These are the fuel stations, but other features of these images are roads, parking areas, parked cars and trucks, and also trees, fields, and other buildings.

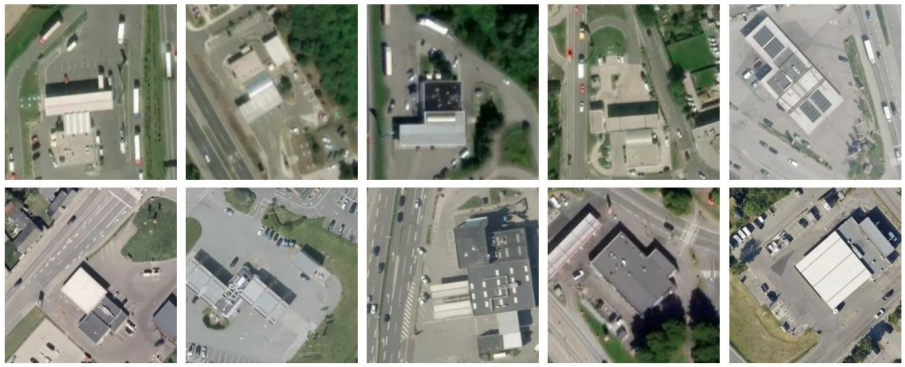

Figure 17: Fuel class scenes

4.1.5 Data Cleaning

With the process of data acquisition completed, the next step was to ensure data quality by means of data cleaning in order to complete TACID and use it for model training. Each collection of satellite imagery was inspected visually by going through the data one image at a time. During this process, faulty scenes were identified and removed, ensuring each class was represented by 1,000 images for machine learning (ML). Data quality challenges encountered during the inspection included cloud cover, poor image resolution, poor radiometric quality, reflection, and scenery showing no relevant features for ML, such as construction sites. As described in section 3.3.3, the amount of collected satellite images per class was higher than 1,000 images per label, so that such errors could be detected and deleted, leaving enough data for proper training. Figure 18 depicts a few examples of deleted scenes per class.

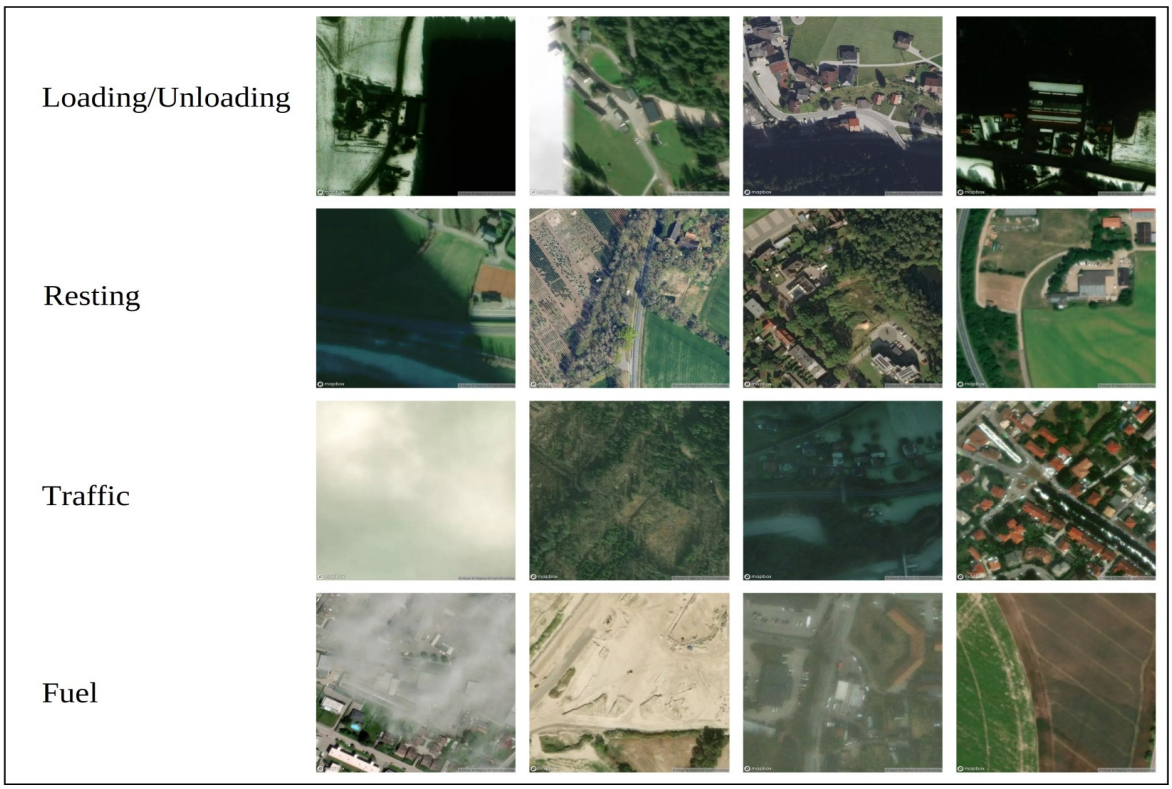

Figure 18: Deleted images from TACID

4.2 Model Training

After completing the creation of the TACID dataset, the next step within the thesis framework was to start model training. This section evaluates the performance of the two ResNet50 models that were trained on the training data: the three-class model (Loading/Unloading, Rest, Traffic) and the four-class model (adding Fuel). The comparison of these two models was necessary to address RQ 3 and aimed to assess if the four-class model was able to distinguish between rest and fuel labels, even though these classes tend to be geographically adjacent, especially in the setting of highway service stations.

4.2.2 Three-Class Model Results

Model training was started on the three-class model in order to assess RQ 3. The focus on traffic stops instead of driving points was assumed to optimize the overall workflow of truck activity classification by reducing the amount of input data for the analysis. Therefore, it was necessary to train the ICM to distinguish between Loading/Unloading, Rest, and Traffic stops. Several iterations over the training data were performed, slightly adjusting the hyperparameters after each training loop. After five separate runs, the best model achieved 82% overall accuracy. Further adjustments did not improve the results, which is why this outcome was chosen as the best classification model. The learning curves for the three-class model show that the learning process runs smoothly for the first 12 epochs, with training and validation loss being very close to one another. After that a

gap between training and validation loss becomes apparent and the accuracy also drifts apart after the 12th training run as depicted in Figure 19. Early stopping terminated the training loop after epoch 16, since the model was not improving its performance any more. The difference between training and validation loss was 0.4 at this stage, while training and validation accuracy were off by around 0.1 at the end.

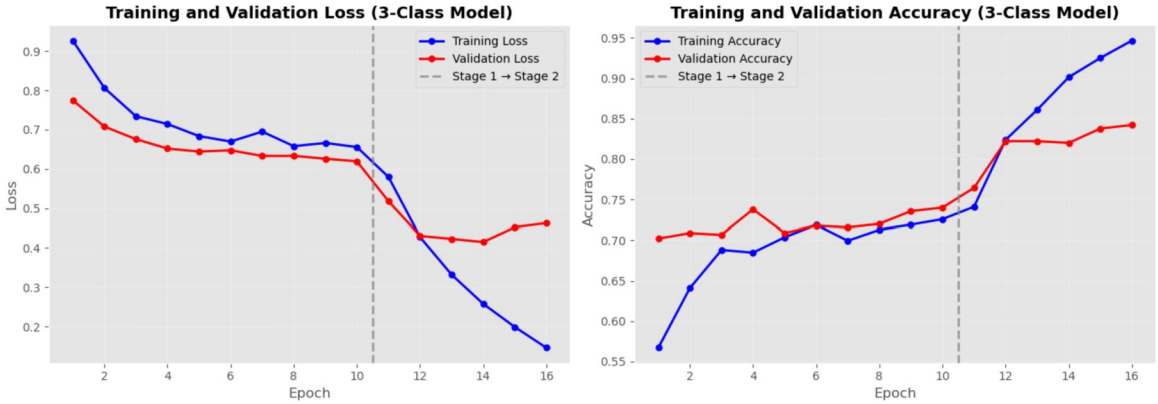

Figure 19: Learning curves three-class model

Table 14 visualizes a detailed classification report, showing that the loading class achieved the highest F1-score (0.86) followed by the traffic class (0.84), while the rest class (0.75) was lower. The precision values show that all three classes were within a close range (0.81 – 0.83) with the rest class achieving the highest and traffic the lowest score. Recall values depict that the loading class had the best results (0.90) and the rest class the lowest score (0.69), while traffic performed at 0.81. The macro average values represent the average value across all classes and clearly match the overall accuracy of 82%. It also becomes apparent how many class instances supported these results. The overall accuracy and the macro average was based on the whole test dataset, while class-wise outcome was dictated by the class-specific test sets.

	Precision	Recall	F1-Score	Support
loading	0.82	0.90	0.86	150
rest	0.83	0.69	0.75	150
traffic	0.81	0.81	0.84	150
accuracy			0.82	450
macro average	0.82	0.82	0.82	450

Table 14: Classification report three-class model

Only 103 out of 150 rest class images were classified as such, while the loading class had 135, and the traffic class 131 matches, as visualized in Figure 20. 25 images were classified as traffic stops and 22 were predicted as loading, while they actually belonged to the rest class. The model also predicted 12 images as rest, when they actually were from the traffic class, and 9 as rest even though they belonged to the loading class. The metrics for both the loading and the traffic class generally match the recall and precision values from the classification report (Tab. 14).

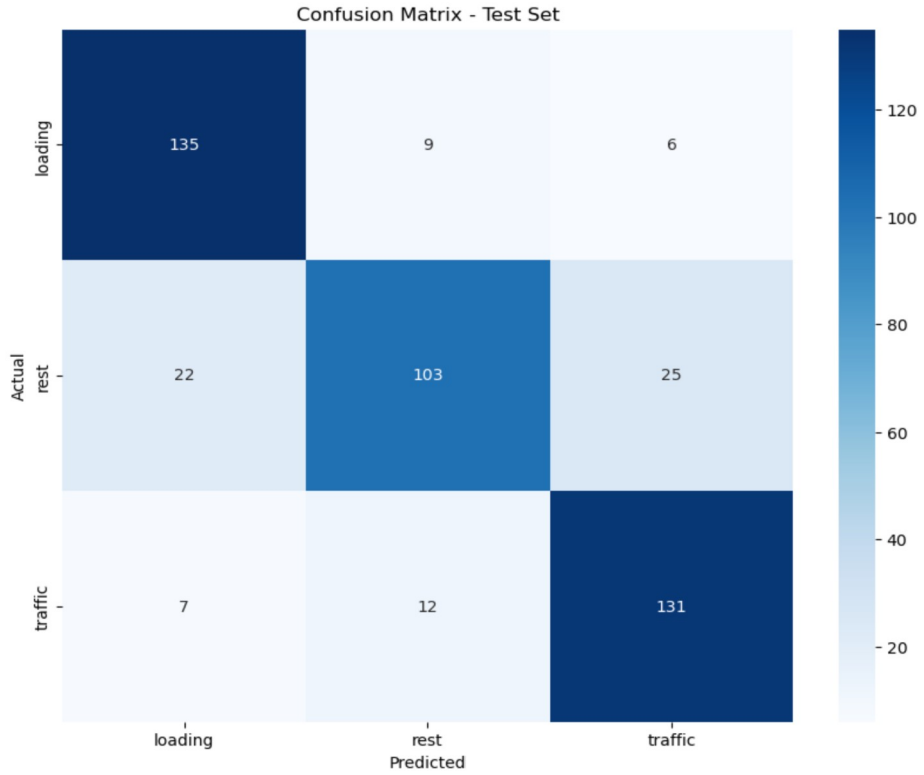

Figure 20 Confusion matrix three-class model

In order to better understand these patterns, some examples of misclassified images were displayed for inspection. Figure 21 shows scenes belonging to the rest class being labeled as loading instances and some images that were predicted as rest scenes, but actually belonged to the loading class. Both image collections show scenes with buildings in the image center but also flat roofs with parking spaces next to them, trees, parked vehicles, roads, and fields. The same goes for the misclassification between the traffic and the rest class. Most images show roads with multiple lanes and nature around them, some vehicles, and some smaller buildings. The rest image on the left (top) displays roads with buildings all around. Few images contain shaded areas, but most scenes depict locations at a resolution that allows recognition of distinct features within the images.

Figure 21: Misclassified images three-class model

4.2.3 Four-Class Model Results

This section is necessary to address the third research question and highlights the results for the four-class model. The goal was to distinguish between rest stops and stops for

refueling purposes purely based on satellite imagery instead of rule-based implementation, in order to reduce the dependency of the model on predefined parameters such as dwell time. As mentioned above, the model parameters for the four-class model were very similar to the three-class parameters, only adjusting the dropout value to 0.5. After several iterations and adjustments the final overall test accuracy for the model was **69.17%**. The training and validation loss curves cross each other between epochs four and six, run close to one another for a while, before drifting apart after epoch 12. At epoch 16, the distance between both curves is around 0.5. The accuracy curves crossed three times at epoch two, four, and five, then stayed close for six epochs before crossing again at epoch 12 and then drifting apart until early stopping ended the training run of the four-class model. The final gap was around 0.2, which was almost double compared to the three-class model (Fig. 22).

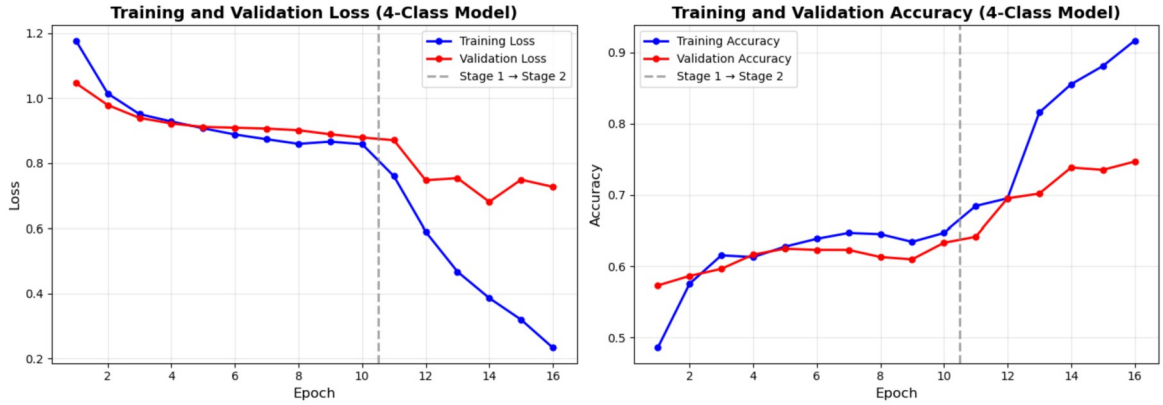

Figure 22: Learning curves four-class model

The detailed classification report (Tab. 15) depicts the relevant metrics for all four classes. The F1-scores show that the newly introduced fuel class reached 0.51, the rest class 0.56, loading a value of 0.82, and the traffic class reached the highest value at 0.83. The four-class model's precision values show that the lowest results were achieved by the rest class (0.51), followed by fuel (0.69), traffic (0.77) and loading (0.81). The recall values for each class depict that the fuel class had, by far, the lowest outcome (0.41), rest the second lowest (0.63) followed by loading (0.83) in second, and traffic (0.90) in first place. The accuracy of the model was evaluated at 69%, which is consistent with the macro average scores (0.68 – 0.70). The supporting class images from the test dataset were 150 instances for each class, and the overall accuracy was based on all 600 images.

	Precision	Recall	F1-Score	Support
fuel	0.69	0.41	0.51	150
loading	0.81	0.83	0.82	150
rest	0.51	0.63	0.56	150
traffic	0.77	0.90	0.83	150
accuracy			0.69	600
macro average	0.70	0.69	0.68	600

Table 15: Classification report four-class model

The four-class confusion matrix (Fig. 23) shows that 61 fuel, 125 loading, 94 rest, and 135 traffic images were labeled correctly. 19 predicted fuel instances actually belonged to the rest class, six to the loading class, and two to the traffic class. From the loading class predictions, twelve were misclassified as fuel, 13 as rest, and four as traffic instances. 70 predicted rest images belonged to the fuel class, eleven to loading, and nine to the traffic class. The prediction of the traffic class was misclassified seven times as fuel, eight times as loading, and 24 times as rest.

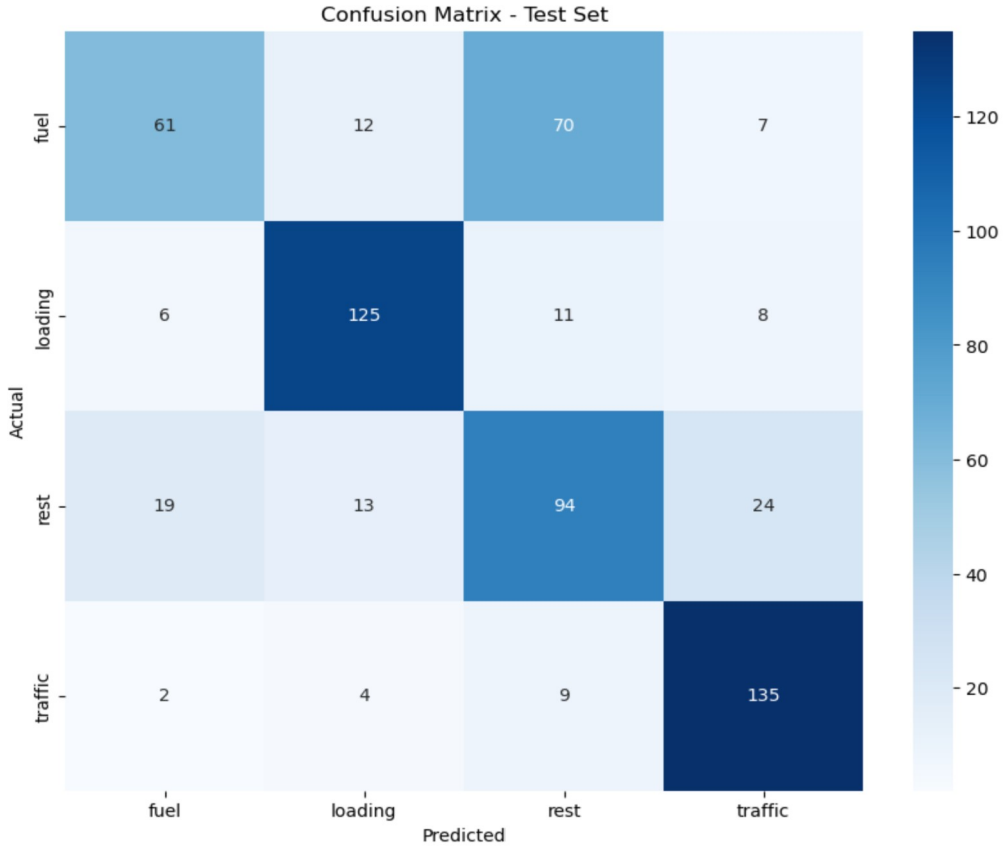

Figure 23: Confusion matrix four-class model

In order to better understand these findings, some examples of misclassification were included and visualized in Figure 24. Images that were predicted as rest instances but actually belonged to the fuel class clearly show similar features with flat-roof buildings in the center, parked trucks, and large parking areas. The images that were predicted as fuel instances but actually belonged to the rest class show various buildings, large paved areas, and parked vehicles. Rest predictions belonging to the traffic class depict roads next to each other, fields and trees, some buildings, and some shaded areas. Finally, some loading scenes misclassified as rest instances are displayed. The images depict buildings, parking areas with parked vehicles, roads, and fields and trees.

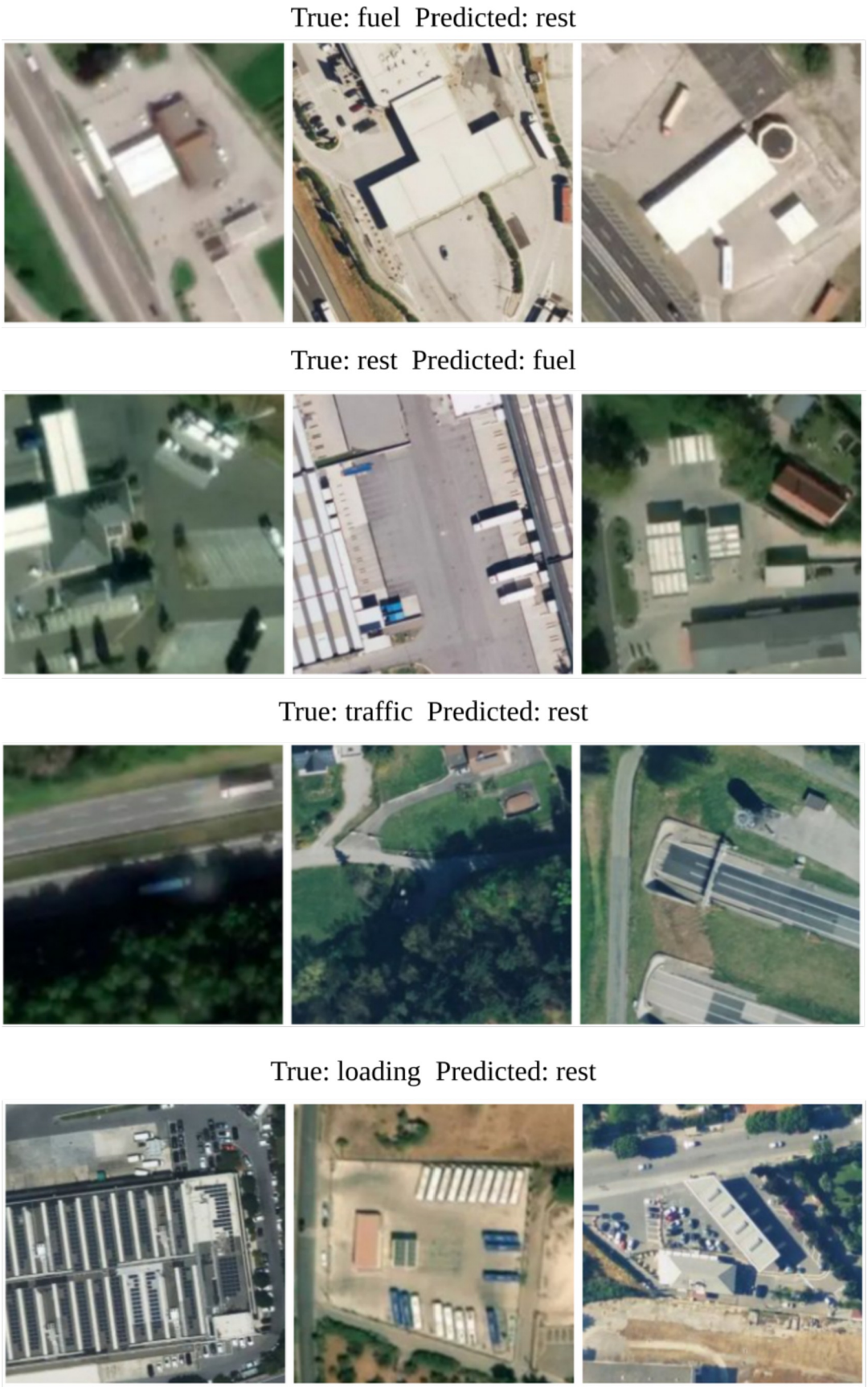

Figure 24: Misclassified images four-class model

4.3 GNSS Data Processing Pipeline Results

The previous sections depicted the creation of the TACID dataset and visualized the results of model training in depth. As mentioned in Chapter 3, each step depended upon the results from the previous one, which is why the focus of this final results section will be on the analysis of the truck trajectory GNSS data. The dataset consisted of 64 separate files, each representing the truck movements for the time span from September 3rd 2025 until September 10th 2025. The truck trajectory processing pipeline was designed to analyze each file separately, since the data was acquired this way and for reasons of computer memory and performance. Figure 25 depicts all files combined on the left and one single file on the right to get a better impression of the extent of the data.

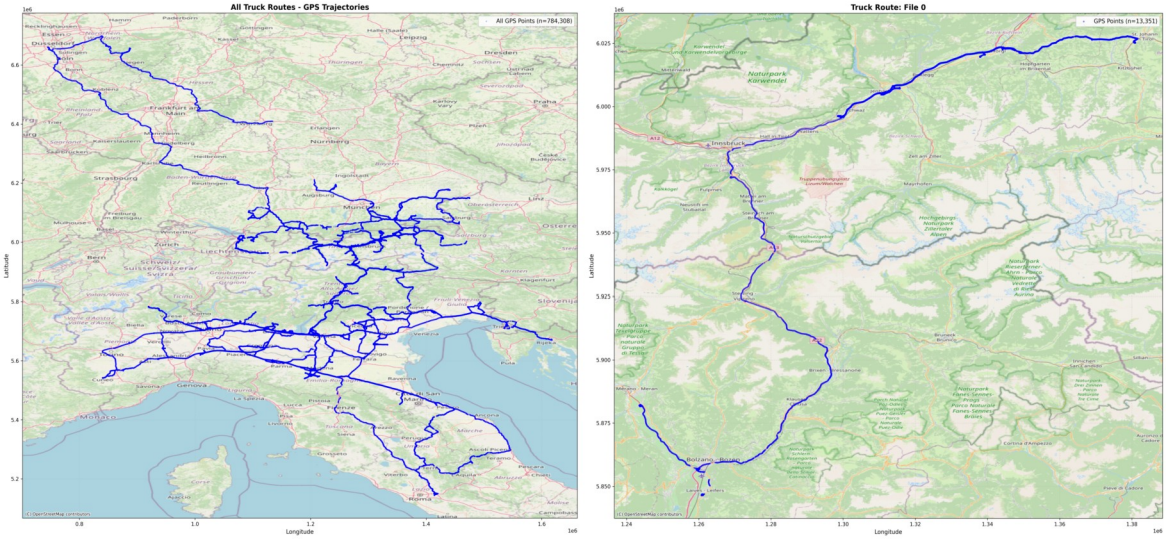

Figure 25: All trajectories vs. single file

4.3.1 Preprocessing

As mentioned in section 3.5.2, various preprocessing steps had to be applied to the data in order to be able to perform deeper analysis. By keeping only the most relevant attributes from the GNSS data, the extent of the datasets was reduced to timestamp, coordinates, and speed information. Table 16 shows two entries from such a reduced file, highlighting the fact that the speed column had missing values, which is why it needed closer consideration.

timestamp	latitude	longitude	speed
2025-09-03 14:01:24	47.272812	10.930452	89.0
2025-09-03 14:35:24	47.259760	11.376253	NaN

Table 16: Reduced data frame

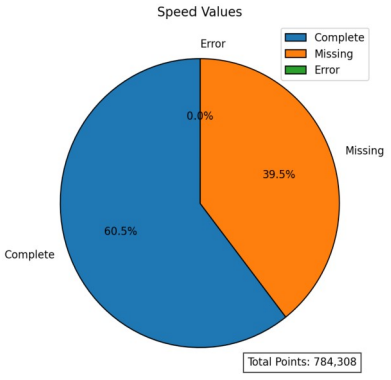

Figure 26: Speed values percentages

A total of 309,471 missing speed values were discovered and filled in with the corresponding 0 km/h value. This means that almost 40% of all points represent stationary events with zero speed. While looking for GNSS points with speed errors, the threshold of 120 km/h yielded only 17 out of 784,308 total points, which were labeled as ‘speed error’

and removed, leaving a total of 784,291 GPS points for spatial analysis and classification (Fig. 26). The last preprocessing steps and reprojection were conducted as explained in section 3.5.2 before continuing the analysis.

4.3.2 Stop Detection Results

The next step was to detect stop events within the truck trajectories. As mentioned above, the first part of this section will highlight the outcome for the application of the speed threshold before moving on to the implementation of the DBSCAN algorithm for stop location clustering.

Speed Threshold Filtering

In order to address RQ 2, the application of a speed threshold of 5 km/h was performed so that only stop candidates were left to analyze. This focus on stationary moments within the GNSS truck data was performed for data reduction. From the 784,291 GPS points across 64 files, the implementation of the speed threshold removed 456,521 signals from the data pipeline, yielding a total of 327,770 stop candidate points for cluster analysis (Fig. 27). Compression ratio and reduction percentage underline this important step and the significance of these results.

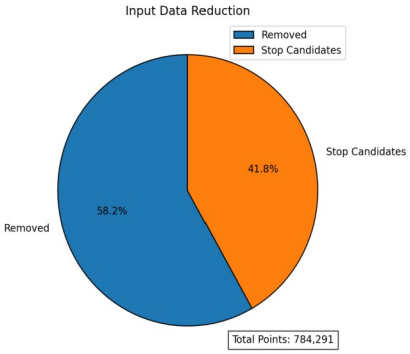

Figure 27: Data reduction speed threshold

- **Compression Ratio** = Original Size/Reduced Size

  → The filtered dataset was 2.39 times smaller than the original.

- **Reduction Percentage**=(1–Reduced Size/Original Size)*100

  → 58.2% of the data were removed by applying a speed threshold of 5 km/h

When looking at the file-wise reduction (Fig. 28) of input points, this data reduction becomes visually apparent. It is clear that the file sizes are rather varied, with roughly the first two thirds being closer to 20,000 point signals and the last third closer to 5,000 points. A few exceptions can be identified, such as file 40, which was by far the largest collection of GPS signals, reaching more then 35,000 pings. Files 9, 35, 42, and 59 were very small datasets not even representing 1,000 recorded signals. The larger datasets clearly contained more driving points than the smaller samples.

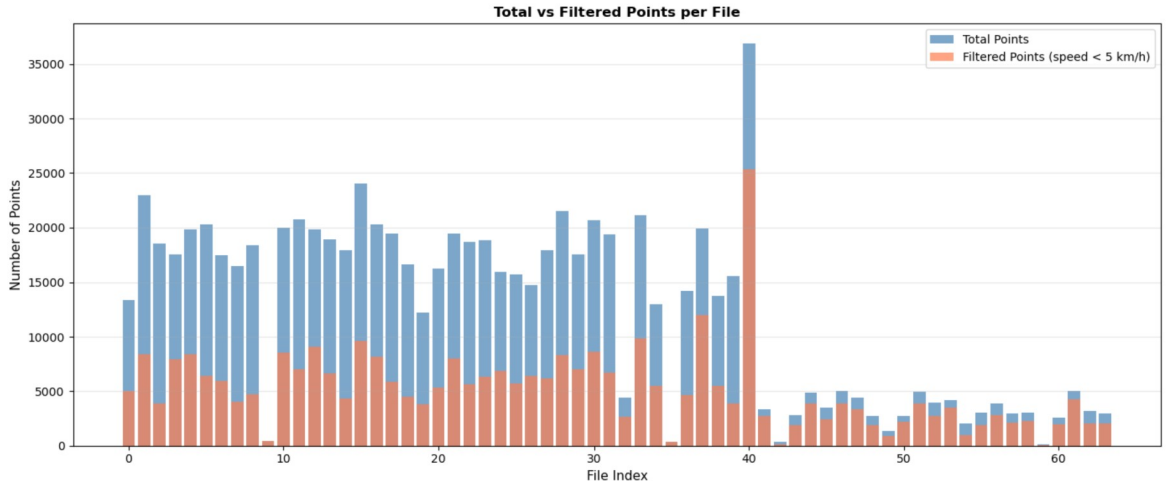

Figure 28: Total vs. filtered GPS points per file

To make this process a bit more visually understandable, Figure 29 highlights the distribution of the speed values over time for four separate files. The files were chosen because of their very different characteristics. Files 10 and 51 represent one regular file from the first two thirds and the last third of all files, respectively, while Files 9 and 40 represent the two types of outliers. The different scatter plots highlight the extent of the data reduction achieved by the application of the speed threshold, with the exception of file 9, which consists of almost purely points with a speed value below the threshold. File 10 does not really show strong patterns within the data, since the driving points are spread out over the whole plot without very apparent interruptions. Files 40 and 51, on the other hand, show clear gaps within the removed speed values over time, hinting at stationary moments of extended time and very specific time frames when the trucks were moving. These visualizations do not allow to deduce meaningful predictions about the activity patterns but they suggest the fact that deeper analysis could help gain more insight.

The extent of all stop candidate points (Fig. 30) shows clear patterns of operation already before clustering. The company operating out of Tirol, Austria, has some stop locations further away from their headquarters, where points become less densely distributed, but clearly has more relevant locations in the vicinity of their hub of operations south of Munich, Germany, where stops seem to be more frequent. It also becomes quite apparent that the Brenner Autobahn is an area of frequent congestion, since the whole road from Tirol down to Italy is filled with stop points.

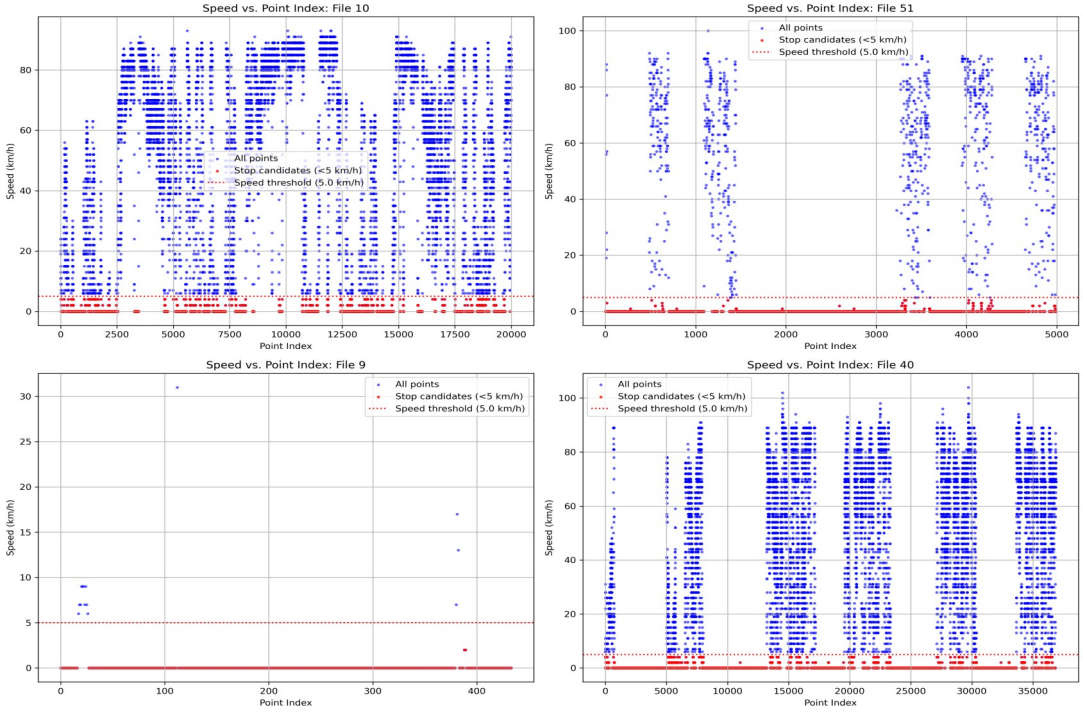

Figure 29: Speed over time: four selected files

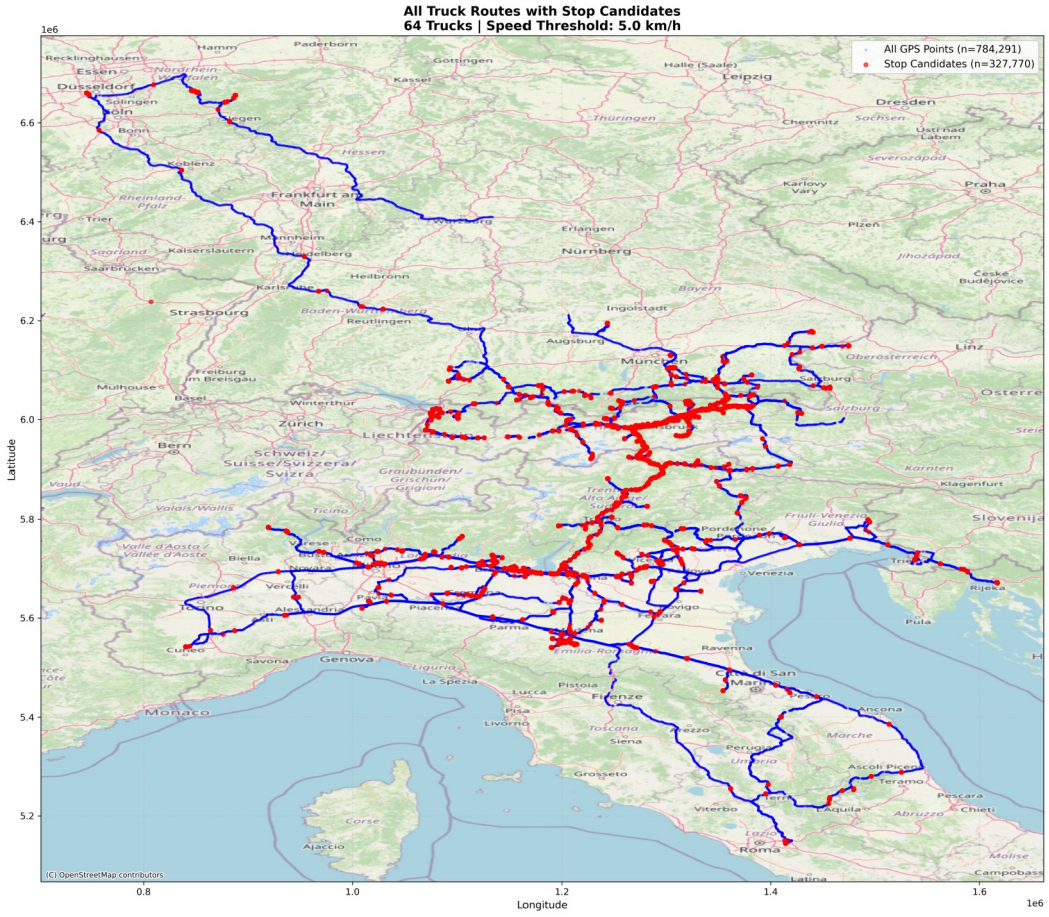

Figure 30: All points vs. stop candidates

DBSCAN Results

After stop candidate identification through the application of the 5 km/h speed threshold, the next step within the framework was to identify which of these candidates belonged to the same locations. Without this information the final step of activity classification would not be feasible in the manner proposed by this thesis. With input data reduction in mind, the idea was to find important locations within the detected stop candidates, which means clusters of stop candidates belonging to the same geographical area. The framework for cluster analysis used for this thesis was the well-established DBSCAN algorithm. The spatial threshold parameter (**Eps**) was tested for four different values: **50**, **100**, **150**, and **200 meters**. Looking at the results for these different settings, it becomes clear that by performing this step, many apparent traffic stops get clustered into groups, since some sections of the analyzed area contained a lot of such traffic stops that were close to one another. The approach seemed promising, since it was able to reduce the amount of observed points drastically. Table 18 gives an overview of the extent of data reduction depending on the spatial parameter. With a spatial threshold of 50 meters, 1,893 clusters and 8,230 noise points were detected, while an Eps value of 100 yielded 1,677 clusters and 6,239 noise points. With a 150 meters radius, the 327,770 stop candidates got reduced to 1,582 coordinate pairs for cluster centroids and 5,286 noise points, and a 200 meters radius produced 1,513 clusters and 4,625 noise points.

Eps in Meters	Detected Clusters	Noise Points	Noise Rate
50	1893	8230	2.51%
100	1677	6239	1.90%
150	1582	5286	1.61%
200	1513	4625	1.41%

Table 17: Cluster output

In order to find out which spatial radius (Eps) was best suited to detect meaningful clusters, a few selected scenes were displayed in Figures 31 and 32. One example depicting the choice of the most suited Eps value is a single stop event from File 11 that took place from September 5th at 09:11 am until 10:05 am. Minor movements within the compounds of the stop location took place during this stop event, but the signals still belonged to the same stop. Figure 31 shows the original file (top left) and the cluster results for three different Eps values, again showing the impact of the application of a speed threshold on the amount of GPS signals that needed to be analyzed. An Eps value of 50 meters yielded three separate clusters and a few noise points, 100 meters produced two distinct stops without noise, and the spatial radius of 150 meters correctly grouped all observed points into one single cluster. The 200 meter Eps was excluded from this visualization, since the results were the same as with 150 meters.

Another example suggesting that 150 was a good choice for the Eps setting was depicted in Figure 32. The scene depicts a single stop event that happened on September 4th from 07:53 am until 10:18 am, again with minor movements during the stay. The 50 meter Eps value yielded three separate clusters and some noise at the top right corner along the intersection, which represented two short traffic stops. A 100 meter value for the spatial radius grouped all stop candidates within the compounds into one single cluster,

identifying the two noise points correctly, as does the 150 meter analysis. The 200 meter value for Eps, on the other hand, integrated the two noise points into the stop event. These findings support the decision to use the 150 meter radius for further analysis to reduce the amount of input data for the classification model.

Another decision made at this point was to exclude noise points from the classification process, since these low density points represented short traffic stops due to congestion and short stops at intersections and traffic lights. Longer traffic stops produced more GPS signals in the same area, making sure that the DBSCAN algorithm identified them as actual stop events, which were later classified using the classification model. Figure 33 depicts a few scenes of such noise points and also clustered traffic stop events in order to make this decision more transparent and understandable. With the raw GNSS trajectory data prepared, reprojected, filtered for stop candidates, and these locations clustered, the next step was to extract the centroid coordinates of these clusters in order to proceed to the task of satellite image acquisition before classification.

Figure 31: DBSCAN results 50, 100, 150 meter Eps

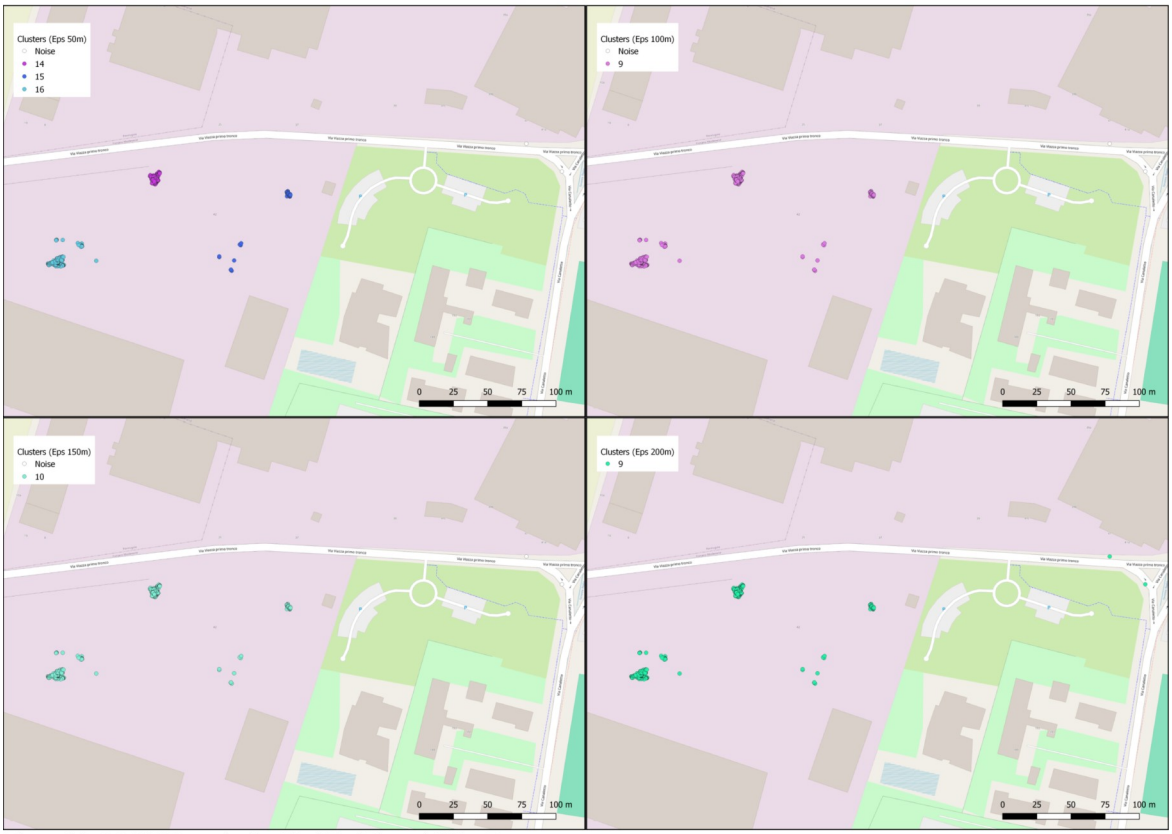

Figure 32: DBSCAN results for 50, 100, 150, and 200m Eps

Figure 33: Noise points and traffic clusters

Coordinate and Image Extraction Results

With the corresponding coordinates for each cluster centroid computed, the processing step of satellite image extraction was straightforward to perform. As mentioned earlier, the image acquisition was achieved in the same manner as for the creation of the TACID dataset. The data was extracted using the Mapbox API and saved into a predefined directory, which would serve as input for the pretrained classification model.

Figure 34 depicts a selection of acquired cluster centroid satellite images, showing that the applied methodology from the previous processing steps of this thesis seemed to produce valid coordinates. The images were very similar to the ones presented for the creation of the TACID dataset, showing highway parking areas, roads with trucks and cars on them, buildings, and materials stacked in an outdoor area next to buildings and paved areas.

Figure 34: Cluster centroid images

A visual inspection of the collected image data helped detecting some poor-quality images, which was to be expected. Figure 35 introduces some of these scenes, depicting that some images showed aerial photographs containing blurry scenes, others showing forest with no distinct features, or poor lighting conditions. The visual inspection resulted in the identification of 17 such poor-quality images within the 1,582 downloaded images, which is a very low rate of 1.1%.

Figure 35: Poor quality centroid images

Before moving on to the classification of these cluster centroid images, Figure 36 depicts a scene of stop candidate points for a chosen location and the according cluster centroid positions. This visualization again highlights the significant data reduction achieved through stop candidate clustering. Stop candidates from all files were displayed together in blue, and the clusters represent the stops by different trucks. Some cluster centroids are very close to one another because multiple trucks stopped at the same general location.

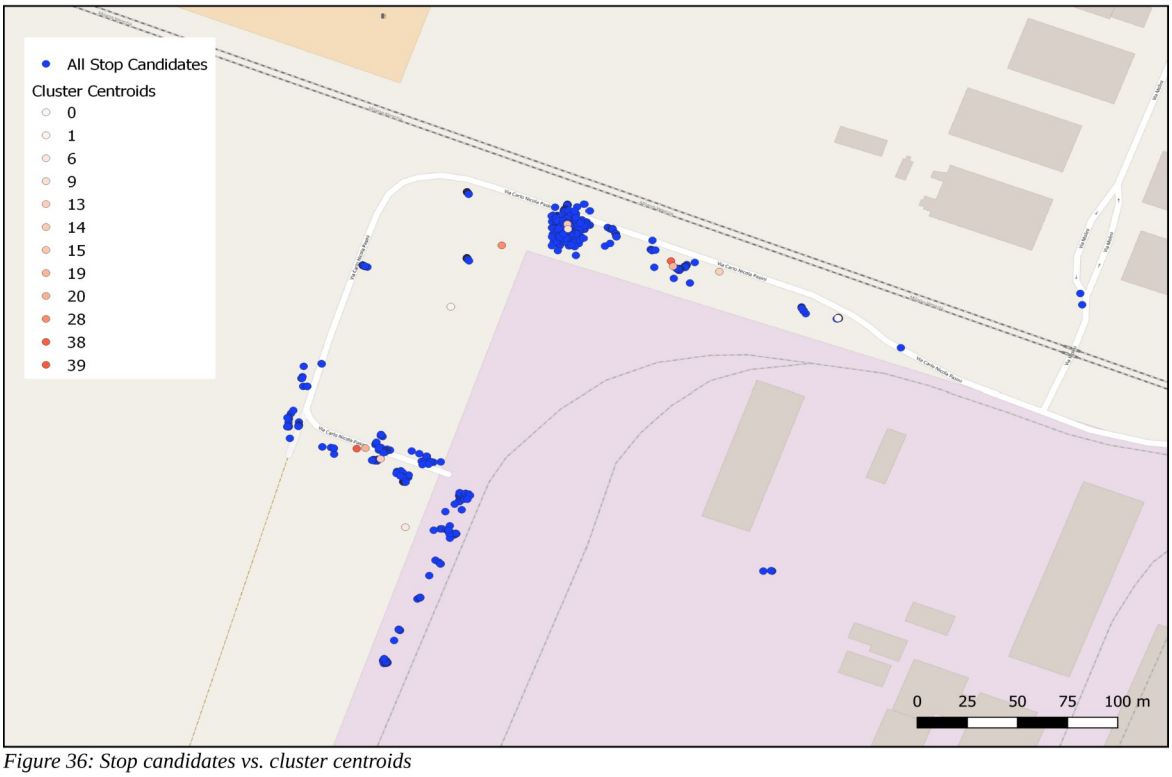

Figure 36: Stop candidates vs. cluster centroids

4.3.3 Stop Classification Results

The next step of the study was to use the detected cluster centroid images for classification using the pretrained ResNet50 model. Since the four-class model showed poor performance in distinguishing between ***rest*** and ***fuel*** stops, the decision was made to classify the processed data with the three-class model only in order to predict the cluster center locations into the three predefined classes: ***loading/unloading***, ***rest***, and ***traffic***. This was done by loading the best version of the three-class model, which was saved during the training process. The model classified the 1,582 images using the previously learned weight settings and produced the output (Fig. 37) with an average confidence score of 0.89.

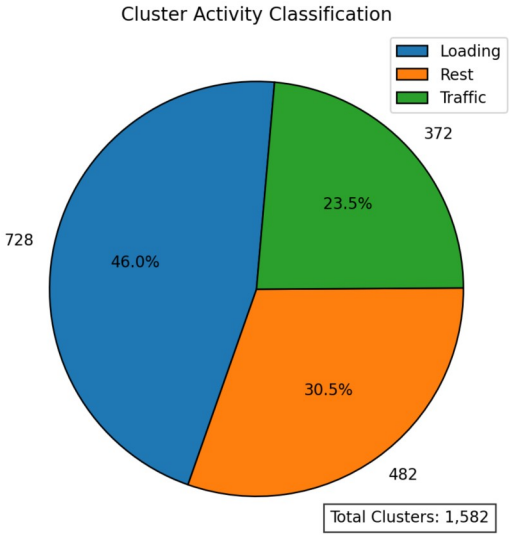

Figure 37: Activity distribution

The model detected 728 loading/unloading stops, followed by 482 rest stops and 372 traffic stops. Figure 38 depicts a few examples per class, adding the confidence score to the according image, which gives information about how confident the model was while making the decision to assign a certain label. The top three images were labeled as loading and it is apparent that all three scenes share similar features, notably big buildings, spacious parking areas, and parked vehicles. The predicted rest class images are more

varied. The one in the middle looks like a parking area next to a building, whereas the other two show buildings right next to roads and highways. The predicted traffic scenes are very similar to one another showing roads and highways in each image, except the one at the bottom right, which also visualizes a parking area and some buildings right next to a road.

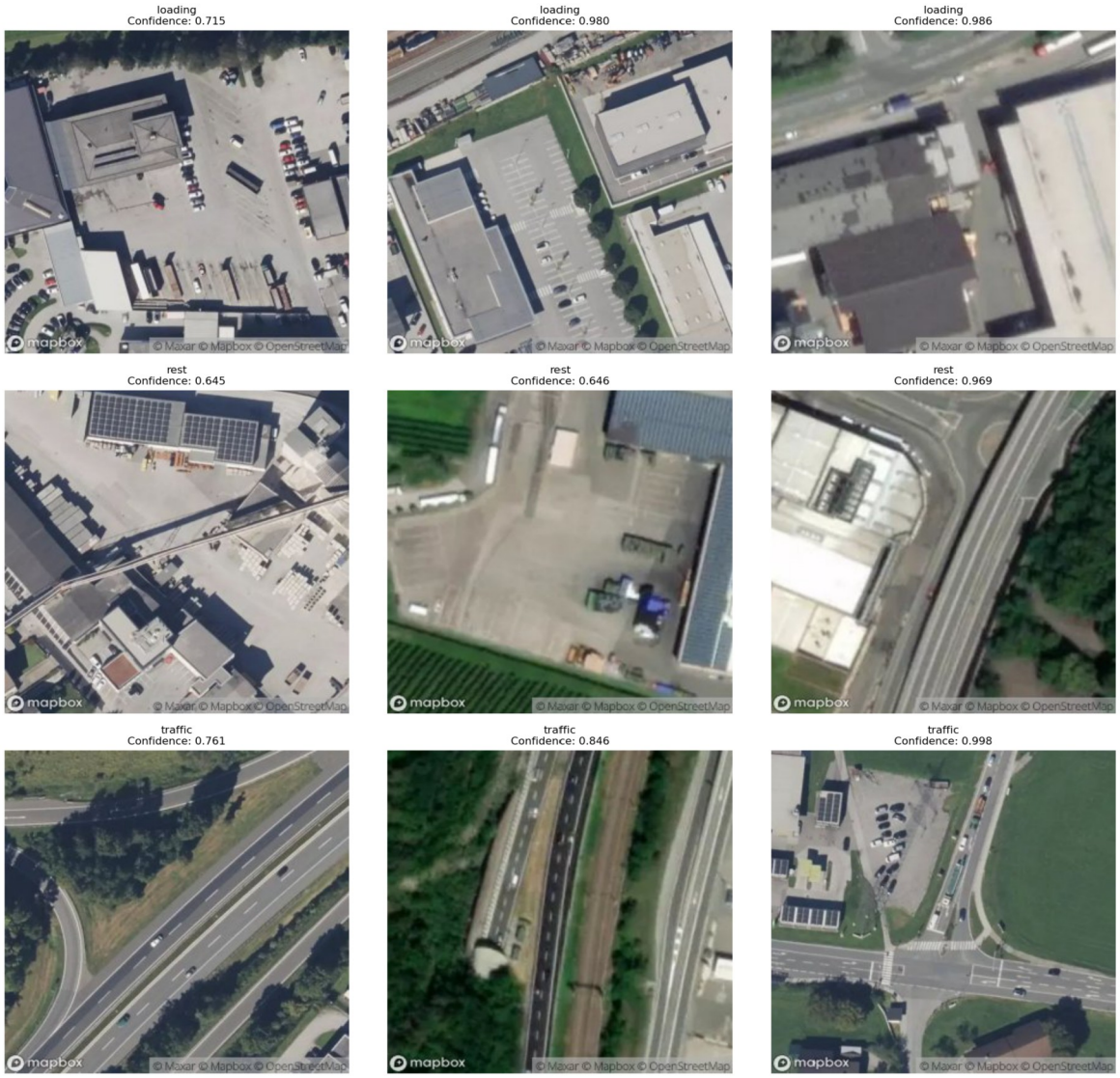

Figure 38: Predicted labels with confidence score

In order to perform an accuracy assessment, a subset of 150 images was labeled manually. This approach was necessary to obtain an indicator of how well the model performed on real world data instead of the training data. Unfortunately, the labeling could not be performed without bias, since no actual ground truth data was accessible for this thesis. This is why the images had to be labeled manually, without knowing the activity that was actually performed at these locations. 50 images per predicted class label were selected for the test sample, choosing images with confidence scores from 0.42 to 1.0 and covering 52

unique TIDs, in order to get a varied sample for manual labeling. The results from the comparison of predicted and manually labeled images were as visualized in Table 18.

Class	Manual label	Predicted label
loading/unloading	52	50
rest	46	50
traffic	51	50

Table 18: Manual vs predicted labels

The classification report (Tab. 19) shows the performance of the model, which reached 71.3% overall accuracy. The loading and traffic class yielded the highest F1-scores (0.74 and 0.81), whereas the F1-Score for the rest class was at 0.58. The precision values underline these results, with traffic reaching 0.82, rest 0.56, and loading 0.76. Recall depicts that out of all instances that should have been predicted, 72% of all loading images, 61% of all rest instances, and 80% of all traffic stops, were labeled correctly.

	Precision	Recall	F1-Score	Support
loading/unloading	0.76	0.72	0.74	53
rest	0.56	0.61	0.58	46
traffic	0.82	0.80	0.81	51
accuracy			0.71	150
macro average	0.71	0.71	0.72	150

Table 19: Detailed classification report

The confusion matrix (Fig. 39) depicts that out of all predicted rest labels 28 were true, whereas eight belonged to the traffic class and 14 to loading. 41 traffic stops were identified correctly, while 8 predicted traffic instances belonged to the rest class and one to the loading class. The loading class was predicted correctly 38 times but mislabeled 10 times as rest and two times as traffic.

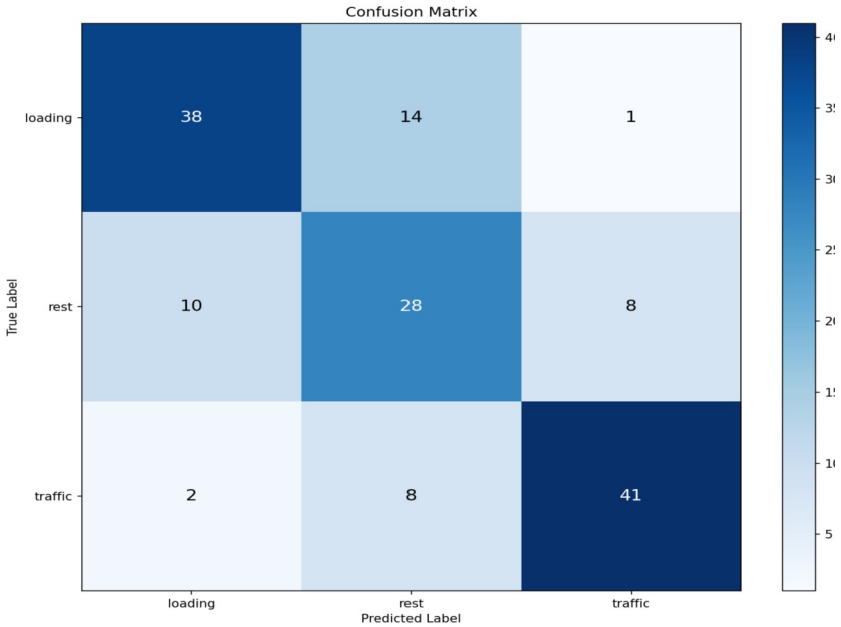

Figure 39: Confusion matrix for image classification

Figure 40 depicts some of the misclassified images for a better understanding. Two of the four images that were wrongly attributed to the rest class actually belonged to the loading class, depicting large buildings with parked vehicles and materials stacked on concrete areas. The other two showed suggested traffic stop scenes (top right and middle left) showing a road with fields and trees on the sides, and some parked vehicles and buildings. Two images were classified as traffic, with the one in the top row on the right having loading as manual label, showing roads and buildings next to areas covered by trees and roads. The one on the bottom right depicts a road with a highway rest station next to it, with parked trucks and cars in the scene and was actually labeled as rest. The last three images were labeled as loading by the classification model, while both images in the last row, which depict the same scene, were manually labeled as rest. They show a large parking area next to a building and lots of parked cars and trucks. The image in the middle shows a road passing through a forest and had the manual label of a traffic stop.

Figure 40: Misclassified image examples

4.3.4 Transmission Frequency Analysis

Before looking into the truck-wise stopping patterns that were discovered through this satellite image-based classification methodology, the transmission frequency needed to be analyzed. RQ 4 addressed this data feature and the hypothesis that it would be a limiting factor for meaningful truck trajectory segmentation was established. This section will explore if the assumption can be confirmed or rejected. In order to get a first impression of the transmission patterns across the 64 fleet vehicles Figure 41 visualizes the mean and median frequencies and the standard deviation for each file in a bar plot. In regards to the mean and median values it becomes apparent that the first 40 files have rather high transmission frequencies compared to the last third of all files. Files 9 (~1,400), 35 (>1,600), and 42 (>1,600) are clear exceptions reaching very low mean averages. The standard deviation shows a different pattern with the first two thirds being very close to the overall average standard deviation value of 643.5 seconds and the last 20 files being much less spread out except for a few files (48, 55, 60, 62). The file with the highest variability is file 42, reaching a standard deviation value (>12,000) three times higher than file 35, which is also much higher than the average.

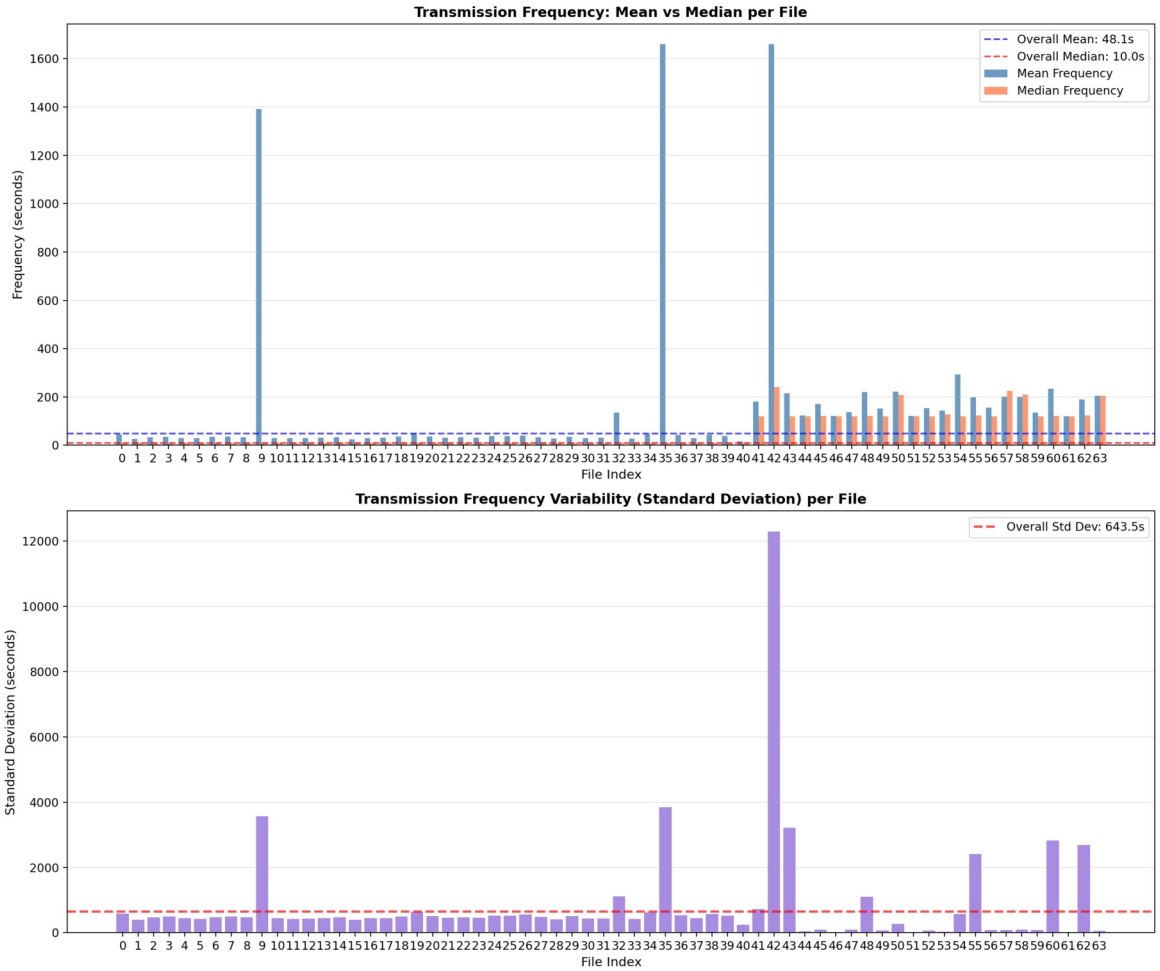

Figure 41: Transmission frequency: mean vs. median and standard deviation per file

To get a better impression of these findings it was decided to look at the variation of the transmission frequency for a few selected files to highlight the spread of the data. To do so four separate files displaying varied transmission patterns were selected for closer inspection. It was decided to select file 0 because it was very close to the average observed values for both mean and standard deviation, file 35 for its values way above both averages, file 40 with both values below the average, and file 57 because it displayed mean and median values above the average but its standard deviation was much lower than the average value. Figure 42 depicts four separate box plots for these files making it clear that, except for file 57, all observed datasets contained extreme outliers within their transmission frequencies. The box and whiskers do not even get displayed correctly in these plots, again underlining the challenge presented by the high variability of the signal transmission rates. With gaps reaching close to 11,000 seconds (~3 hours) it becomes very apparent that a meaningful temporal analysis was not easily achieved.

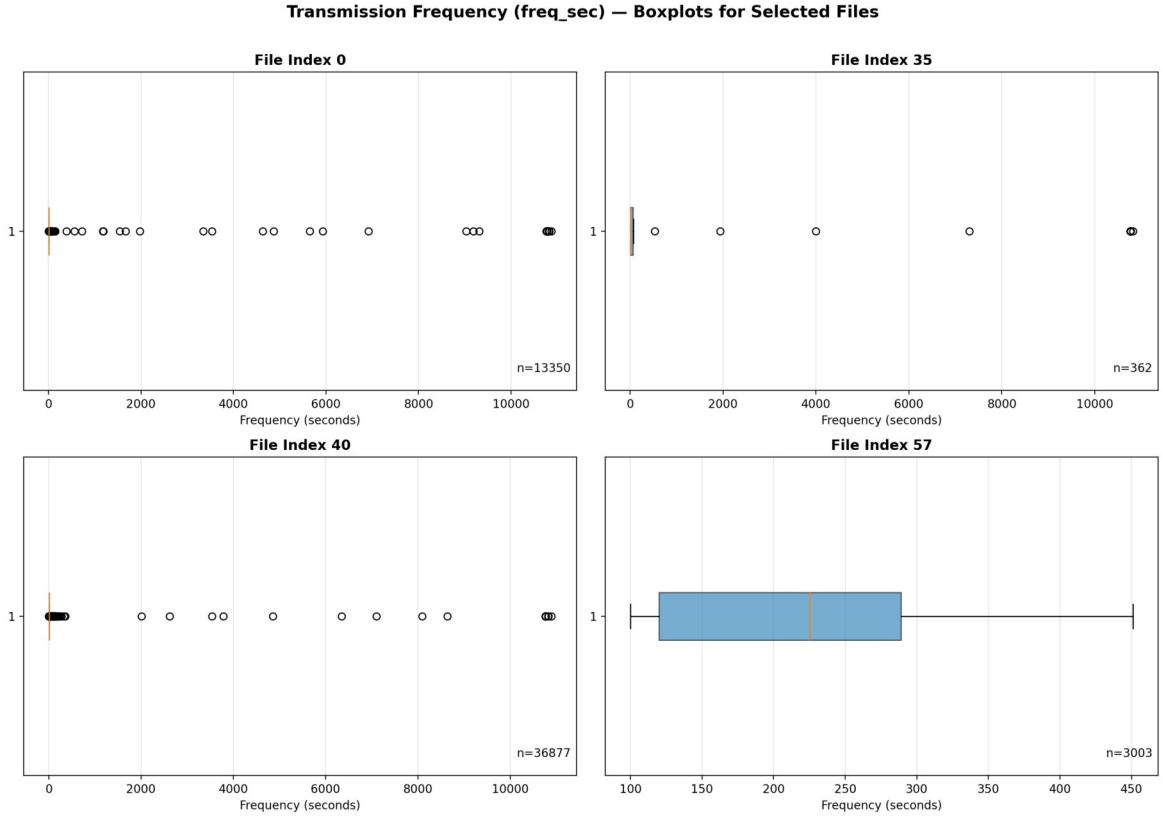

Figure 42: Transmission frequency box plots for four selected files

Basic statistical analysis (Tab. 20) for the whole dataset highlighted these trends further, depicting the average values computed from the aggregated data for all 64 trucks. The challenge of dealing with frequency outliers and long temporal gaps gets highlighted by these statistics showing that the strong spread of the data comes from a small percentage of all recorded transmission gaps. Since 75% of all points show frequencies up to ten seconds only, it becomes very clear that large temporal gaps (2,535 total) distort the values significantly, which makes meaningful analysis rather complicated. The largest gap even reaches a duration of almost 49 hours during the observed seven days of truck trajectories.

Metric	Value
Mean	48.1 seconds
Standard Deviation	643.5 seconds (~11 minutes)
Minimum	1 second
25% quantile	10 seconds
50% quantile	10 seconds
75% quantile	10 seconds
Maximum	176,692 seconds (~49 hours)
Number of gaps > std	2,535

Table 20: Basic Statistics for aggregated transmission frequency

As the irregularities within the transmission frequency distribution have been depicted, the next section will focus on the temporal segmentation of the detected stop location clusters in order to gain knowledge about the visiting patterns of certain places, but also the time spent there. This section addressed the topic necessary to answer RQ 4, but the next part of this thesis will help determine whether the hypothesis can be confirmed or must be rejected.

4.3.5 Truck-Wise Stop Patterns

The last step of the processing pipeline was to analyze the labeled results from section 4.3.3 to look for temporal visiting patterns within the detected clusters. Stop locations were detected by applying a speed threshold and then clustering the stop candidate points, but this approach did not yet clarify if trucks visited the same locations multiple times during the observed time span and how much time they spent at those places. The hypothesis was that, without ping transmissions at regular intervals, a meaningful segmentation of the truck trajectories would prove to be challenging. Figure 37 from section 4.3.3 depicted the percentages for each detected activity (ICM classes only) for all stop clusters, but this section will look at all the data in order to detect truck-wise patterns. Figure 43 visualizes the classification results for each separate file with driving points added, showing the percentages for each task. Except for files 9 and 35, every truck depicts a mix of different activities with loading (orange) clearly accounting for the highest overall percentage (53.45%) and traffic stops the least (5.52%), while resting accounted for roughly 20% and driving for another 20%.

Knowing how many stop location clusters were discovered by truck, the next step was to find out how much time was spent at these locations, and if the places were visited multiple times or not. This part of the thesis will depict these results based on a few

selected files in order to illustrate the discovered patterns. Figure 44 clearly indicates that most clusters represent locations that were visited for a short period of time, up to two hours with some exceptions. File 35 had only one cluster with 167 hours spent there, which corresponds to the whole time span of the observed data (seven days). All three other files contained one cluster where large amounts of time were spent and also various other stop locations with short visits. What needed to be clarified was if these long time clusters contained multiple visits and how long these visits were.

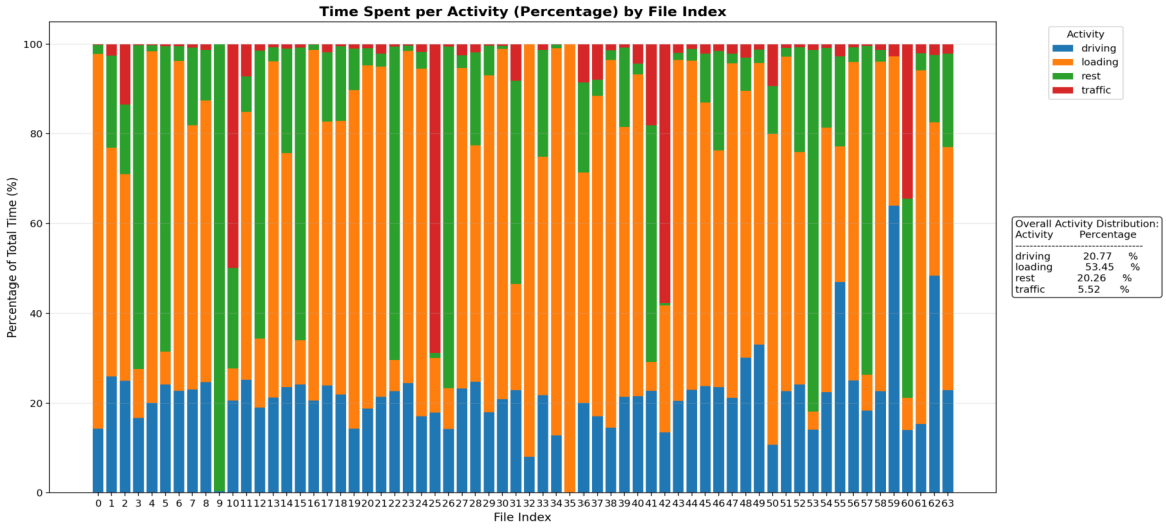

Figure 43: Time per activity per file

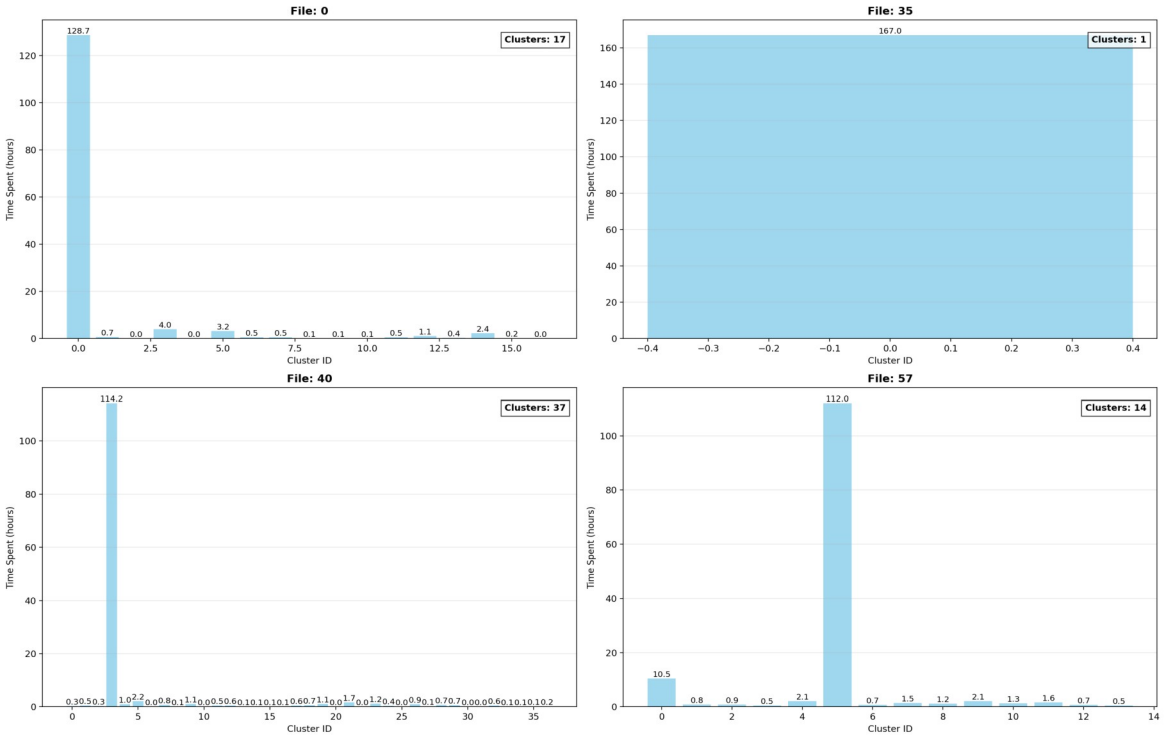

Figure 44: Number of detected clusters and time spent for selected files

The temporal analysis was performed for each file by looking at clusters separately and adding temporally-adjacent points into the same visit, but starting a new event when a gap

larger than a predefined threshold (600 seconds) was detected. The assumption was that regular transmission frequency would produce meaningful segmentation whereas large gaps within the recordings would distort the analysis results. In order to make the results understandable the same four files were used for visualization and deeper inspection of the segmentation results. Figure 45 depicts the same information as shown above (Fig. 44) but with the number of visits on the y axis of the bar plot so that the relevant information can be deduced easily. For file 0, it is obvious that all clusters, except for a few, were visited more than once, with cluster 0 containing over 50 separate stops. The cluster in file 35 was visited almost 60 times even though it was the only detected stop location, whereas file 40, the largest dataset, had a few locations that were visited several times with cluster 3 having over 30 distinct stop events. The last visualized file (57) depicts four out of 14 clusters that contain multiple visits, but none reaching more than 8 separate stop events. Looking at the large transmission gaps within these four files (Fig. 46) the necessity of a deeper inspection of these results becomes apparent, which is why one selected multi-visit cluster for each file will be analyzed for cross-examination.

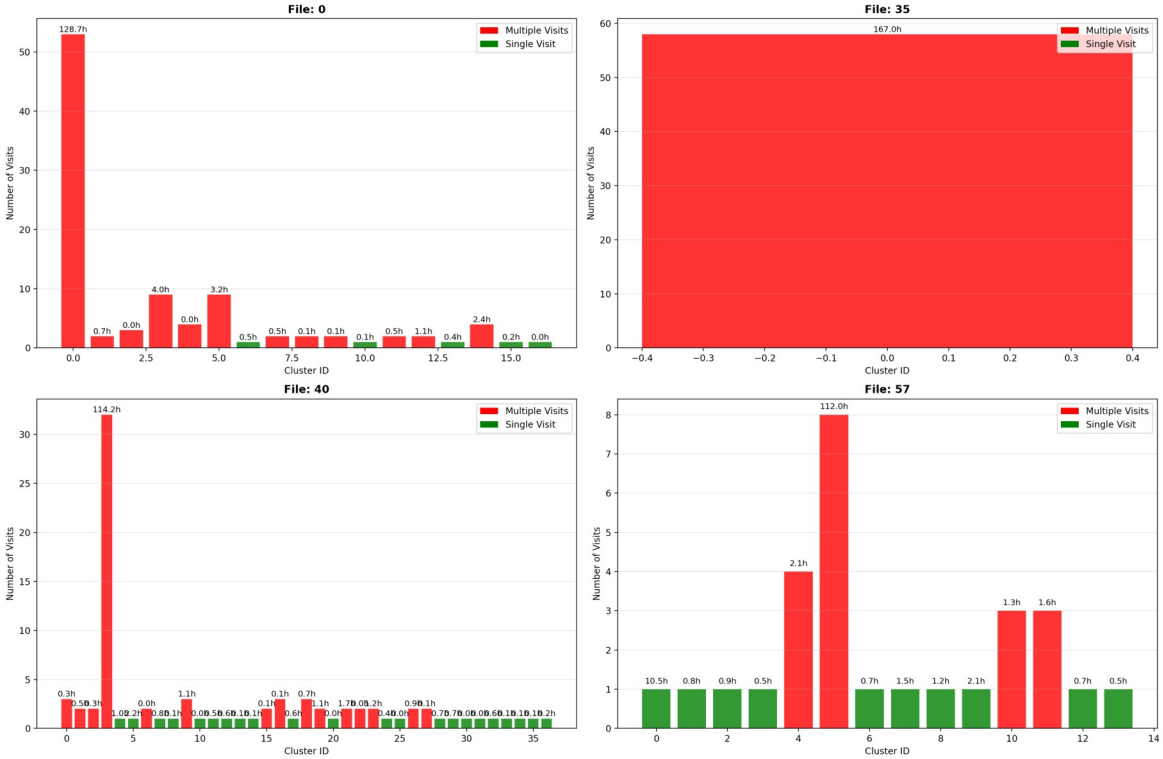

Figure 45: Number of clusters and time spent for selected files with numbers of visits

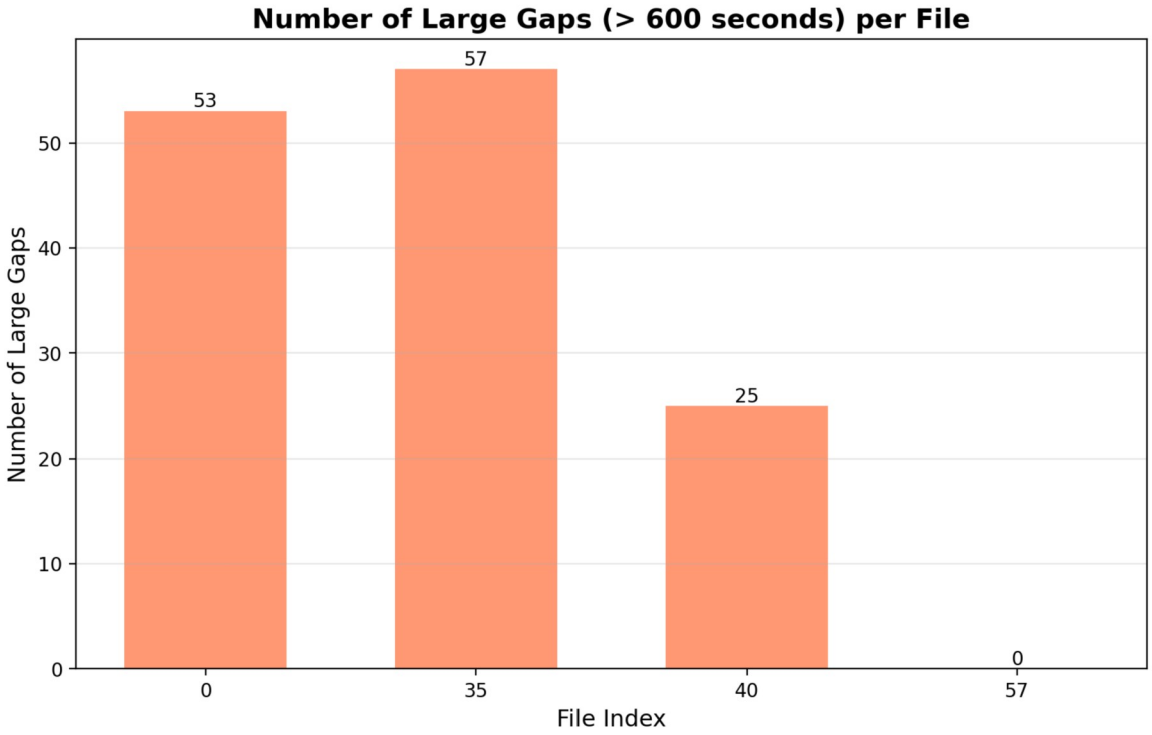

Figure 46: Number of large temporal gaps within selected files

Cluster-Wise Inspection

The first inspected Cluster was Cluster 0 from File 0 with 53 predicted separate visits. For easy visual inspection the temporal visiting patterns were visualized in a table depicting the number of each visit with start/end time and duration. The predicted output was then combined with the actual visiting patterns detected by visual inspection of the data in QGIS below the visits belonging to the actual stop. By segmenting the cluster manually it became clear that instead of 53 stops, the location was only visited seven separate times. Large temporal gaps made a meaningful analysis impossible, even though the time gap threshold was set to 600 seconds (ten minutes).

Visit	Start	End	Duration
1	2025-09-03 15:38:36	2025-09-03 15:48:39	0.2
…	…	…	…
7	2025-09-04 08:47:20	2025-09-04 09:07:38	2.2
Stop 1	2025-09-03 15:38:36	2025-09-04 09:07:38	17.5
8	2025-09-04 10:26:02	2025-09-04 10:35:22	0.2
9	2025-09-04 13:05:57	2025-09-04 13:27:28	2.9
Stop 2	2025-09-04 10:26:02	2025-09-04 13:27:28	3
10	2025-09-04 14:34:28	2025-09-04 14:43:58	0.2
…	…	…	…
17	2025-09-05 10:22:08	2025-09-05 10:29:01	1.7
Stop 3	2025-09-04 14:34:28	2025-09-05 10:29:01	20

18	2025-09-05 12:53:56	2025-09-05 13:03:16	0.2
…	…	…	…
40	2025-09-08 02:30:01	2025-09-08 04:04:24	4.6
Stop 4	2025-09-05 12:53:56	2025-09-08 04:04:24	6
41	2025-09-08 05:07:15	2025-09-08 05:32:20	0.4
Stop 5	2025-09-08 05:07:15	2025-09-08 05:32:20	0.4
42	2025-09-08 14:29:06	2025-09-08 16:25:33	1.9
…	…	…	…
48	2025-09-09 03:06:50	2025-09-09 03:19:47	1.9
Stop 6	2025-09-08 14:29:06	2025-09-09 03:19:47	9.83
49	2025-09-09 14:40:04	2025-09-09 14:50:34	0.2
…	…	…	…
53	2025-09-10 02:26:50	2025-09-10 02:31:22	2.6
Stop 7	2025-09-09 14:40:04	2025-09-10 02:31:22	11.85

Table 21: File 0 cluster 0: predicted vs. actual stops

Cluster 0 from File 35 showed similar challenges when it came to segmentation into separate stops. The pipeline predicted 57 distinct visits, even though the visual inspection showed that the truck was stationary for the whole observed time span leading to only one single actual stop.

Visit	Start	End	Duration
1	2025-09-03 08:55:42	2025-09-03 08:56:42	0.0
2	2025-09-03 11:56:12	2025-09-03 11:57:12	3.0
3	2025-09-03 14:56:42	2025-09-03 14:57:42	3.0
4	2025-09-03 17:57:12	2025-09-03 17:58:12	3.0
…	…	…	…
54	2025-09-09 23:53:39	2025-09-09 23:54:39	3.0
55	2025-09-10 02:54:09	2025-09-10 02:55:09	3.0
56	2025-09-10 05:54:38	2025-09-10 05:55:38	3.0
57	2025-09-10 07:46:46	2025-09-10 07:14:29	2.0
Stop 1	2025-09-03 08:55:42	2025-09-10 07:56:49	167

Table 22: File 35 cluster 0: predicted vs. actual stops

The next inspected file depicted some very interesting patterns in regards to the hypothesis. Cluster 3 form File 40 had 32 predicted separate stops versus seven actual visits. The first three and the fifth stops were predicted correctly with transmission frequency during these events being high and regular (10 seconds). Then long temporal gaps start to distort the predictions leading to 29 more stops being detected, even though there are only four stops within the data.

Visit	Start	End	Duration
1	2025-09-03 16:31:16	2025-09-04 04:26:52	11.9
Stop 1	2025-09-03 16:31:16	2025-09-04 04:26:52	11.9
2	2025-09-04 08:54:34	2025-09-04 09:15:34	0.3
Stop 2	2025-09-04 08:54:34	2025-09-04 09:15:34	0.3
3	2025-09-04 13:41:36	2025-09-05 04:25:36	14.7
Stop 3	2025-09-04 13:41:36	2025-09-05 04:25:36	14.7
4	2025-09-05 16:00:56	2025-09-05 22:08:50	6.1
…	…	…	…
25	2025-09-08 02:54:04	2025-09-08 03:07:45	2.6
Stop 4	2025-09-05 16:00:56	2025-09-08 03:07:45	58.56
26	2025-09-08 13:49:03	2025-09-08 13:57:33	0.12
Stop 5	2025-09-08 13:49:03	2025-09-08 13:57:33	0.12
27	2025-09-08 14:08:03	2025-09-09 00:25:38	3.0
39	2025-09-09 04:29:58	2025-09-09 04:43:57	1.3
Stop 6	2025-09-08 14:08:03	2025-09-09 04:43:57	4.3
30	2025-09-09 14:15:27	2025-09-09 23:21:43	9.1
…	…	…	…
32	2025-09-10 04:08:58	2025-09-10 04:14:21	1.9
Stop 7	2025-09-09 14:15:27	2025-09-10 04:14:21	13.97

Table 23: File 40 cluster 3: predicted vs. actual stops

Cluster 5 from File 57 was predicted with eight separate stops, which was confirmed by the visual inspection. This file did not contain any large temporal gaps (Fig. 46), leading to a reliable separation into distinct stops. This was possible even though the transmission frequency rates within this file varied from 100 to 451 seconds.

Visit	Start	End	Duration
1	2025-09-04 15:29:46	2025-09-08 04:29:18	84.8
Stop 1	2025-09-04 15:29:46	2025-09-08 04:29:18	84.8
2	2025-09-08 16:13:19	2025-09-09 05:02:13	12.7
Stop 2	2025-09-08 16:13:19	2025-09-09 05:02:13	12.7
3	2025-09-09 08:56:13	2025-09-09 09:12:09	0.3
Stop 3	2025-09-09 08:56:13	2025-09-09 09:12:09	0.3
4	2025-09-09 12:02:01	2025-09-09 12:16:13	0.2
Stop 4	2025-09-09 12:02:01	2025-09-09 12:16:13	0.2
5	2025-09-09 14:02:13	2025-09-10 03:02:16	12.9
Stop 5	2025-09-09 14:02:13	2025-09-10 03:02:16	12.9
6	2025-09-10 07:38:11	2025-09-10 07:54:17	0.3
Stop 6	2025-09-10 07:38:11	2025-09-10 07:54:17	0.3
7	2025-09-10 11:00:13	2025-09-10 11:18:17	0.3
Stop 7	2025-09-10 11:00:13	2025-09-10 11:18:17	0.3

8	2025-09-10 12:50:16	2025-09-10 13:24:17	0.5
Stop 8	2025-09-10 12:50:16	2025-09-10 13:24:17	0.5

Table 24: File 57 cluster 5: predicted vs. actual stops

5 Discussion

The aim of this thesis was to find out if the SIATAR framework presented by Hu et al. (2024) could be adapted in a way to effectively handle large, multi-vehicle datasets covering diverse geographical contexts. This novel approach introduced a deep learning algorithm for image recognition to the field of truck GPS data analysis and classification, producing very promising results. The reason for this attempt was the fact that the satellite image-based approach was only performed on a small homogeneous truck trajectory covering one single route, which was identified as a research gap that needed investigation. To address this research gap, the hypothesis was established that the framework needed to be combined with additional processing steps in order to derive meaningful spatial and temporal patterns from the analyzed data. Based on this assumption, a set of research questions and hypotheses was established to help guide the implementation of the adapted framework. Combining the method with well-established preprocessing techniques with a focus on data reduction was considered as a central necessity for the success of such an attempt. Another goal was to test whether an image classification model would be capable of distinguishing between different types of rest stops, which was tested by training two separate networks, and temporal segmentation of the trajectories was attempted, even though GPS signal transmission frequency was considered to be a limiting factor.

The adapted framework developed for this thesis suggests that the combination of the satellite image-based classification method with other techniques was able to produce some interesting findings in regards to the research questions. Before addressing each question separately, this section was designed to highlight the main findings of the research.

- Extending the **training image dataset** to more heterogeneous geographical contexts introduced certain **similarities between activity classes**, which had an impact on the classification model performance.
- The **three-class** model achieved an overall accuracy of **82%**, while the f**our-class** model only classified the test data with **69%** accuracy.
- Applying a **speed threshold** of 5 km/h **reduced** the amount of analyzed GPS signals **by 58.2%**, and **DBSCAN** clustering reduced the amount even further, yielding 1,582 stop locations for image classification, which corresponds to **0.2%** of the input data.
- The analysis identified **transmission frequency rates** to be a strongly **limiting factor** for temporal route **segmentation**.

With these major findings in mind, the goal of this chapter is to interpret these results and put them in context with the research questions and hypotheses in order to get reliable

answers and deduce the implications and limitations encountered. The focus on feasibility and applicability of such a framework meant that the other research questions had to be answered first, in order to be able to answer the main question. For this reason, research questions two, three, and four were designed in a more specific manner, and their answers were necessary before making a final statement with regard to the main focus of this study.

5.1 Input Data Reduction

The second research question was designed to address the extent of data necessary for the classification model. It was assumed that detecting stop locations within the truck trajectories would help in reducing the amount of analyzed GPS points significantly. By applying a speed threshold of five kilometers per hour to identify stop candidate points and using the DBSCAN algorithm to cluster these coordinates into stop locations, the framework designed for this thesis was able to confirm this hypothesis. The simple speed filter achieved a data reduction rate of 58.2% in a first step, removing 456,521 of the 784,291 original GPS points from the processing pipeline, which demonstrated the strong impact of this approach (Section 4.3.2, Fig. 27). These results align with the literature and were to be expected, even though this technique alone is not ideal for the identification of stop events. It tends to include very short stops on highways into the stop candidates and also excludes minor movements within the premises of a stop location (Choudhry & Qian, 2023). Nevertheless, the outcome was favorable for the analysis at hand, since it reduced the amount of points for the application of the clustering algorithm. Spatial clustering using the DBSCAN algorithm in order to group these identified stop candidates into meaningful stop location turned out to be another very powerful tool for data reduction. Chapter 4 showed that, using a spatial radius (Eps) of 150 meters and a minimum points (minPts) value corresponding to the natural logarithm (ln) of the corresponding datasets (Birant & Kut, 2007) reduced the amount of analyzed locations further down to 1,582 locations (Tab. 18), which corresponds to 0.20% of the original data frame. By displaying a few examples of such point agglomerations (Fig. 31, Fig. 32), the reliability of clustering for actual stop location detection was confirmed, as was to be expected considering the state of the art literature (Patel et al., 2022; Gong et al., 2015). The issue of very short stops detected by the speed threshold was dealt with during the clustering approach, since low-density areas (noise) were labeled as traffic and excluded from the classification pipeline. Figure 33 displayed some excluded noise points, but also examples of traffic stop clusters detected by the algorithm. This highlighted the strength of a speed threshold even more, since it can be assumed that without filtering for stop candidates, more traffic stop clusters would have been detected, thereby leading to a lower data reduction rate. The framework demonstrated that by applying this two-stage stop identification, only 0.2% of the original GPS data had to be passed to the classification model to determine which activities took place during observed time frame. This 99.8% data reduction was very significant, thereby resulting in a confirmation of the established hypothesis and a positive answer to the second research question.

Stage	Number of points	% of original dataset
Raw data	784,308	100%
After speed threshold	327,770	41.8%
After DBSCAN	1,582	0.20%

Table 25: Data reduction percentages

5.2 Similar Accuracy for different Classes

The third research question established to see if the framework was applicable to a different context was designed to find out if the introduction of different classes would have an impact on the overall accuracy of the classification process. Since the proposed framework focused on stationary events, it was interesting to find out if the model would be able to recognize a different set of classes but also if the distinction between different types of rest stops (rest and fuel) would be possible and if the model would be able to learn to classify four classes. The hypothesis established in this context was that classes should be identified with reasonable accuracy, if they displayed distinct features, but the distinction between rest and fuel stops would prove to be challenging, because strong similarities in the displayed scenes would reduce the overall accuracy. Considering this set of questions, the results presented in Chapter 4 were clear.

Adapting the framework to distinguish between loading, resting, and traffic stops produced rather satisfying results, achieving an overall accuracy of 82% on the test dataset, as demonstrated in Section 4.2.2. This decrease in performance compared to the values achieved by Hu et al. (2024) can be explained by the extension of the training dataset to more diverse geographical settings. Rather than focusing on highway rest stations only, the rest class training data was acquired with the help of the ETPL dataset introduced in Section 3.3.2, which was created in a scientifically reliable way, representing a broad variety of truck resting locations across diverse geographic contexts across Europe (Link & Plötz, 2024). Such varied input data for model training makes the outcome presented in section 4.2 about model training more understandable. The confusion matrix for the three-class model (Fig. 20, Chapter 4) depicted that the main struggle of the model lay within the rest class, which was misclassified multiple times because of strong similarities between classes. Section 4.1 about the creation of TACID highlighted these similar class features, which were to be expected. The recall value of 0.69 for the rest class underlines this fact, showing that the model was only able to classify about 70% of all rest instances correctly, while loading (0.90) and traffic (0.81) were classified with much higher accuracy (Tab. 14). The fact that trucks regularly take breaks in areas not specifically displaying distinct spatial features, as do highway rest stations, makes these findings understandable. Considering this, the overall accuracy of 82% of the three-class model on the test data proved to be a rather satisfying result, regardless of the 14% drop in model performance compared to Hu et al. (2024).

The introduction of the fuel class to the four-class classification model produced results that were to be expected. The overall accuracy dropped by 13% from 82% (three-class) to 69% on the test dataset. The struggles already encountered within the three-class model

were multiplied during the attempt to introduce the fuel class to the model. Looking at the F1-scores from section 4.2.3 (Tab. 15), it becomes clear that the four-class model struggled with the distinction between the rest and the fuel class. Only 40% of all true fuel instances were identified correctly as such (Recall 0.41), while 70 predicted rest class members actually belonged to the fuel class (Fig. 23), hinting at very strong similarities between these two classes. These results were to be expected, since inter-class similarities make it hard for classification models to learn class specific features (Xia et al., 2017). The confusion matrix for the four-class model highlights the fact that misclassifications happened between all classes rather frequently, showing that this model was not suited to classify the GPS data. This can partially be explained by the fact that highway rest stations sometimes have a fuel station included, but more generally by similar features in general, such as flat-roof buildings, parking areas, and parked vehicles in the analyzed scenes. The overall outcome was very likely influenced by the small dataset size and diversity, which is a common problem in model training (Cheng et al., 2017).

Model	Overall accuracy
Three-Class	82%
Four-Class	69%

Table 26: Classification model performance: three-class vs. four-class

5.3 Temporal Segmentation

The last research question that was established, focused on the temporal visiting patterns within the detected stop locations. Since DBSCAN groups points only according to spatial features, it had to be determined if locations were visited multiple times by the application of a dwell time calculation. The hypothesis was that transmission frequency rates would prove to be a limiting factor for such an attempt, stating that regular signals would facilitate the division into separate stops. Variations and large temporal gaps within the transmissions were assumed to be a challenge that would lead to difficulties.

The results from Chapter 4 confirm this hypothesis and depicted the strong influence of signal rates on the deduction of meaningful temporal patterns. The analysis of transmission frequency rates across all analyzed files in Section 4.3.4 showed that rates were very inconsistent (Fig. 41) and large temporal gaps were found in most files, which made a meaningful temporal analysis very challenging. This directly aligns with the state of the art literature, where Choudhry and Qian (2023) mentioned these challenges and deduced that they make the application of segmentation algorithms complicated, except for the datasets they were created for. In this thesis, 2,535 temporal gaps (Tab. 21) larger than the standard deviation across all files (643.5 seconds) were discovered. By going through the data it was possible to detect some examples highlighting these findings as presented in section 4.3.6. It was concluded that, in order to perform a meaningful temporal analysis, a dataset with regular transmission frequency without large gaps is necessary. It was proven that these large gaps lead to distorted results yielding a very high number of individual visits (Fig. 45), whereas a regular frequency as encountered in Table 24 lead to a correct segmentation of the detected stop events. Table 23 underlines these

findings, since it contained periods with regular frequencies (Stops 1, 2, 3, and 5) that were segmented correctly, but also periods of irregular transmission (Stops 4, 6, and 7) where it proved to be impossible to derive the visiting patterns correctly. In conclusion, the research question regarding temporal segmentation could be answered positively. Temporal visiting patterns could only be derived when transmission frequencies were regular and consistent. Irregular signals and large temporal gaps have proven to be a strongly limiting factor, that prevented reliable temporal pattern detection in real-world GPS datasets, which confirmed the fourth hypothesis.

5.4 Applicability of the framework

The central research question of this thesis was if such a satellite image-based classification methodology for truck activity classification was applicable to large, multi-vehicle datasets, covering varied geographical settings (rural and urban). The hypothesis established in this context stated that, in order to apply such a framework to larger amounts of data, it would require more diverse training data and additional processing steps to handle the data volume. The results presented in Chapter 4 and the answers to the previous research questions depicted in this chapter indicate that such an adaptation is possible, but with limitations.

The framework demonstrated several successful adaptations for large-scale application. Training the classification model on a different set of classes was achieved through the newly created TACID dataset, designed in a similar fashion as introduced by Hu et al. (2024). However, the extension of the rest class to a more heterogeneous set of images introduced certain inter-class similarities (Section 4.1). Combining the SIATAR method with established stop detection techniques proved to be a promising approach, successfully performing truck activity classification for 64 trucks over a one-week period. The data reduction approach achieved significant efficiency, reducing the classification workload by more than 99% (from over 700,000 GPS points to 1,582 stop location clusters), while maintaining the ability to detect actual stop locations as demonstrated in Chapter 4. Additionally, the labeling of 5,286 noise points as traffic stops and their subsequent exclusion from the classification pipeline (Fig. 33) further contributed to computational efficiency. However, several limitations constrain the applicability of the framework. The three-class model achieved only 71% accuracy on manually labeled cluster scenes, which is significantly lower than the 97% demonstrated by Hu et al. (2024). The estimated performance range of 71-82% accuracy can be considered acceptable given the more diverse training data and inter-class similarities. This aligns with Zhu et al. (2019), who stated that overall accuracies between 50% and 60% for difficult-to-distinguish classes can be considered promising, but still represents a strong decline in model performance. Meaningful validation was not possible due to the lack of ground truth data for the analyzed dataset, and temporal segmentation proved to be very challenging because of irregular transmission frequency and large temporal gaps within the data.

In conclusion, the central research question can be answered positively. It was possible to adapt the framework to large, multi-vehicle datasets across varied geographical settings, and the hypothesis regarding the need for diverse training data and additional processing

steps can be confirmed. Scalability and data reduction were demonstrated by the framework.

5.6 Limitations

While the findings of this thesis confirm the established hypotheses and showed that this kind of approach can be promising, challenges and limitations were encountered that have direct implications for future research and practical implementation.

The main limitation for this study was the lack of ground truth data necessary to evaluate the actual performance of the workflow. Even though the three-class model was able to achieve 82% accuracy on the test dataset, the activity classification output from the processing pipeline could not be properly assessed. By comparing the results to a set of manually labeled images, it was possible to estimate the performance at 71% accuracy, but this value must be seen critically. The manual labels of these scenes were assigned without knowing about the actual activities performed at these locations, which only enabled to estimate and not determine the true model performance. This limitation has direct implications for industry implementation, since fleet operators would need to conduct their own validation before deploying such a system with enough confidence.

This lack of ground truth data also made it challenging to determine how accurate the application of a speed threshold and the spatial clustering were in detecting stop locations. The visual inspection described in Section 4.3.2 demonstrated that some reliable results were gained from the processing pipeline, but the sheer amount of analyzed data made it impossible to examine the outcome for every single file and route.

Another limitation was the irregular transmission frequency across datasets, but also within single datasets as described in Section 4.3.4. It became clear that, in order to perform meaningful analysis of GPS trajectories, a regular collection of signals is indispensable. Low data transmission rates and long temporal gaps make it hard to derive reliable trips and stop events, but also reduce the density of such datasets, which is essential when performing spatial clustering. As a consequence some short stop events may not be detected because of the low number of transmission signals at the corresponding locations. Such transmission related data problems also make the calculation of stop duration challenging. The practical implications of this finding for telematics system design are discussed in section 5.7.

Extending the satellite image-based classification method to a broader geographical context introduced strong inter-class similarities, which reduced model accuracy, as was to be expected. The fact that trucks can perform various activities at the same locations, such as resting on the parking space of an industrial compound, or being stuck in traffic right next to a highway resting station, just to name some examples, reduces the reliability of the proposed method. Especially when trying to distinguish between rest and fuel stops, the limitations of this approach became apparent. These accuracy constraints have important implications for the types of analyses this framework can reliably support, as discussed below.

5.7 Implications

The results of this thesis have implications for both the research community and the freight transport industry, highlighting the potential of satellite image-based activity classification, while also demonstrating methodological considerations that must be addressed.

Implications for Research

Several contributions to the field of mobility research can be derived from this study, such as the fact that the deep learning-based image classification can successfully be integrated with GPS preprocessing techniques to analyze large-scale, multi-vehicle datasets. The SIATAR framework introduced by Hu et al. (2024) can be adapted to handle heterogeneous geographical contexts covering both rural and urban settings. This is a valuable insight for researchers developing similar methodologies, since it demonstrated the trade-off between geographical diversity and classification accuracy. However, as highlighted in section 5.6, this adaptation comes with accuracy constraints. The drop from 97% to 71% to 82% suggests that researchers must carefully balance geographical diversity against classification performance depending on their research objectives.

Another methodological contribution was made by demonstrating the effectiveness of combining established preprocessing techniques with novel classification approaches. The integration of speed thresholds and DBSCAN clustering as a two-stage stop detection method proved to be very effective, achieving a 99.8% data reduction while maintaining the capacity to correctly identify actual stop locations. This has significant implications for researchers working with large GPS datasets, as it demonstrated that computational cost of an analysis can be reduced considerably while keeping analytical capability. These findings align with the results of previous studies on GPS trajectory research (Choudhry & Qian, 2023; Patel et al., 2022; Gong et al., 2015; Gong et al., 2018; Aziz et al., 2016).

The thorough investigation of transmission frequency impact on temporal pattern detection is another important contribution for research. While irregular transmission frequency has been acknowledged as a challenge in the literature (Choudhry & Qian, 2023; Sarti et al., 2017; Wu et al., 2021), this thesis provided explicit evidence of how transmission frequency irregularities limit the reliable segmentation of GPS trajectories. The finding that regular transmission intervals of three to five minutes would be sufficient for meaningful temporal analysis offers practical guidance for future data collection and research design. This suggests that moderate but consistent signal rates are preferable to high frequencies with irregularities, which has clear implications for researchers planning GPS data-based studies. The identification of 2,535 temporal gaps (section 5.3) demonstrates that transmission consistency should be strongly considered in study design.

The lack of ground truth data, which limited this thesis, also highlights an important methodological implication: future research applying similar frameworks should prioritize datasets with verified activity labels for reliable performance validation.

Implications for the Logistics Industry

Next to the research contributions, the results also have implications for the freight transport industry. By demonstrating the ability to analyze 64 truck datasets over a one-week period with minimal computational resources, this study was able to suggest that such methods could be implemented by fleet operators to keep track of the movements of their fleet without extensive infrastructure investments.

The classification of truck activities into loading, resting, and traffic categories provides relevant information that could support operational decision-making. Fleet management companies could use such information to optimize route planning, identify inefficiencies in loading operations, or better understand driver rest patterns for safety and well-being purposes. However, as discussed in Section 5.6, the 71% to 82% accuracy range represents an important limitation. While this may be sufficient for fleet analysis and pattern identification, it suggests that such a framework may not be ideal for the verification of individual driver activities or monitoring where higher accuracy would be essential.

The inter-class similarity challenges discussed in Section 5.2, particularly between rest and fuel stops, have practical implications for system implementation. Logistics companies and fleet operators should be aware that certain activities may require additional data sources for reliable classification. The framework is most reliable for distinguishing loading operations from other activities, demonstrating its applicability for delivery pattern analysis.

The findings regarding GPS transmission frequency have clear practical implications for fleet operators and telematics providers. The research demonstrates that while high frequency GPS signals may seem relevant, regular transmission intervals are more desirable for meaningful analysis. As described in Section 5.6, the greatest challenge in this context came from long temporal gaps within the data, instead of low transmission frequencies, suggesting that telematics systems should prioritize transmission regularity over frequency, which represents a relatively straightforward system design consideration.

Overall, the findings demonstrate that satellite image-based activity classification has potential for both researchers seeking scalable GPS analysis methods and fleet operators aiming to understand vehicle operations more efficiently.

6 Conclusion

This research aimed to adapt a satellite image-based truck activity classification method to analyze large, multi-vehicle datasets covering truck movements in both rural and urban contexts. A framework combining machine learning-based image recognition and established GPS data analysis techniques was developed to address this research gap and derive activity patterns from raw trajectory data. The approach was divided into four separate but interdependent steps. First, the training dataset TACID was created with 4000 satellite images representing four activity classes: Loading/Unloading, Rest, Traffic, and Fuel. The second step focused on image classification model training for two models: a three-class model (loading, rest, and traffic stops) and a four-class model (adding fuel). Then a stop detection and classification pipeline was designed in order to reduce the data

volume for the classification model, which then classified the detected places into the predefined classes. In a last step it was attempted to derive temporal visiting patterns from the classified data. Four research questions defined the scope of this research:

- **1.** Is a satellite image-based framework for truck activity classification applicable to large datasets covering multiple vehicles operating in varied geographical settings at a local scale, including urban and rural trajectories over a one-week-period?
- **2.** Does a two-stage preprocessing pipeline for stop identification (speed threshold filtering and spatial clustering) reduce the amount of satellite images required for classification significantly?
- **3.** Can an ICM distinguish between loading/unloading, rest, and traffic stops with satisfying accuracy, and can it further distinguish between rest and fuel stops with similar accuracy?
- **4.** Will GPS transmission frequency limit trajectory segmentation?

6.1 Key Findings

By implementing the proposed workflow, this thesis was able to produce several interesting outcomes.

- **Dataset Construction**:

  The creation of the custom Truck Activity Classification Image Dataset by collecting class specific POI locations and then extracting corresponding satellite imagery for these places was successful. The dataset was rigorously cleaned to exclude poor-quality images, ensuring diverse and reliable input for model training.

- **Model Training:**
  - The t**hree-class model** achieved an accuracy of **82%** on the test dataset, with the loading class performing best (F1-Score: 0.86) and the rest class worst (F1-Score: 0.75).
  - The f**our-class model** showed reduced performance with **69%** overall accuracy, struggling to distinguish between rest and fuel stops (F1-Scores: 0.56 and 0.51). This confirmed the hypothesis that inter-class similarities would have a negative impact on model performance.
- **GNSS Data Pipeline:**

  The application of a **5 km/h speed threshold** and the **DBSCAN clustering** of the detected stop candidates reduced the data volume by **99.8%**, exceeding the attempted 75% significantly.

  The Three-Class Model classified **1,582 stop cluster centroids** with an average confidence of 0.89. Manual validation on a subset of 150 images yielded a **71%** accuracy, with traffic stops achieving an F1-Score of 0.81.

- **Temporal Segmentation:**

  The high variability in GNSS signal transmission with gaps up to 49 hours confirmed Hypothesis 4, strongly limiting temporal segmentation. Large gaps distorted dwell-time calculations, making the derived stop duration unreliable.

6.2 Research Questions-Answers

The findings highlighted above helped answering the research questions, thereby contributing to the fields of mobility research and logistics management. It was possible to confirm all four hypotheses throughout this research, which lead to a positive answer to all four research questions as displayed below:

- **RQ 1**: Applicability to large datasets

  The framework proved to be applicable to large, multi-vehicle datasets across varied geographical settings. However, the reduced accuracy compared to the benchmark 96% highlighted the need for further research on the topic.

- **RQ 2**: Data Reduction

  The two-stage preprocessing pipeline reduced the input data by 99.8%. This validated the hypothesis that the focus on stationary events would lead to significant data reduction.

- **RQ 3**: Model Accuracy for Activity Classification
  - The three-class model achieved 82% accuracy, which was satisfying.
  - The four-class model achieved only 69% accuracy, particularly struggling with the distinction between the rest and fuel class, thereby confirming the hypothesis that class similarity would reduce model accuracy.

- **RQ 4**: Impact of Transmission Frequency

  Irregular transmission frequencies with large temporal gaps limited temporal segmentation, confirming the hypothesis. Without consistent GPS signals, deriving visiting patterns and stop duration was unreliable.

6.3 Future Research

These findings highlight the success and relevance of the proposed combination of workflows. Even though it was possible to prove the validity and scalability of such a satellite image-based approach for truck activity classification, several limitations and implication were encountered during this study, which lead to direct recommendations for future research.

The most obvious recommendation is to apply the presented framework to a ground truth dataset with class labels. This would allow researchers to reliably evaluate the performance of such an approach in the context of large, multi-vehicle truck trajectory datasets. If the dataset has constant signal transmission rates, researchers could apply a

spatiotemporal clustering algorithm and thereby make the temporal postprocessing step of the framework obsolete.

One more aspect of this thesis that needs further investigation is the creation of the TACID dataset. Researchers could try to expand this database and include more images per class to ensure a better feature extraction during model training. It could be beneficial to inspect the collected POI data more thoroughly, to make sure the satellite images are centered exactly on the expected features necessary for pattern learning.

Another recommendation for future work could be to experiment with deeper CNN models such as ResNet101 or others for better feature extraction. Even though it can be expected that model accuracy won't get significantly better because of inter-class similarities, this could still help to achieve better and more reliable results for activity classification. By extending the training loop and unfreezing the model layers less quickly, researchers could try to improve the outcome from model training.

By collaborating with telematics providers to standardize transmission frequency intervals, the strong limitation inherent to this data feature could be tackled in a beneficial way for future GPS data analysis. When working with datasets that are collected with a regular transmission frequency, temporal segmentation could reveal patterns crucial for research and especially practical implementation.

In summary, this thesis has demonstrated that combining GNSS trajectory data analysis with satellite image classification provides an innovative approach to truck activity pattern understanding. By integrating well-established clustering algorithms with machine learning models, it introduced a scalable framework that is capable of deducing insights from raw telematics data. While several challenges related to data quality, data regularity, and visual similarity between classes remain, the adapted methodology presents a meaningful step towards more automated, data-driven analysis. The study shows that combining spatial data science and remote sensing has strong potential for the research in the field of freight transport. With improvements in data quality and availability, such frameworks could contribute to a more comprehensive and efficient understanding of freight transport across road networks.

7 References

Adegun, A. A., Viriri, S., & Tapamo, J.-R. (2023). Review of deep learning methods for remote sensing satellite images classification: Experimental survey and comparative analysis. *Journal of Big Data*, *10*(1), 93. https://doi.org/10.1186/s40537-023-00772-x

Aziz, R., Kedia, M., Dan, S., Basu, S., Sarkar, S., Mitra, S., & Mitra, P. (2016). Identifying and Characterizing Truck Stops from GPS Data. In P. Perner (Ed.), *Advances in Data Mining. Applications and Theoretical Aspects* (Vol. 9728, pp. 168–182). Springer International Publishing. https://doi.org/10.1007/978-3-319-41561-1_13

Bengio, Y. (2009). Learning Deep Architectures for AI. *Foundations and Trends® in Machine Learning*, *2*(1), 1–127. https://doi.org/10.1561/2200000006

Birant, D., & Kut, A. (2007). ST-DBSCAN: An algorithm for clustering spatial–temporal data. *Data & Knowledge Engineering*, *60*(1), 208–221. https://doi.org/10.1016/j.datak.2006.01.013

Boeing, G. (2025). Modeling and Analyzing Urban Networks and Amenities With OSMnx. *Geographical Analysis*, *57*(4), 567–577. https://doi.org/10.1111/gean.70009

Cheng, G., Han, J., & Lu, X. (2017). Remote Sensing Image Scene Classification: Benchmark and State of the Art. *Proceedings of the IEEE*, *105*(10), 1865–1883. https://doi.org/10.1109/JPROC.2017.2675998

Chollet, F. (2021). *Deep Learning with Python* (2nd ed.). Manning Publications Co. LLC. https://sourestdeeds.github.io/pdf/Deep%20Learning%20with%20Python.pdf

Choudhry, A., & Qian, S. (2023). Inferring truck activities using privacy-preserving truck trajectories data. *Journal of Intelligent and Connected Vehicles*, *6*(1), 16–33. https://doi.org/10.26599/JICV.2023.9210002

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255. https://doi.org/doi:%252010.1109/CVPR.2009.5206848

Ester, M., Kriegel, H.-P., Xu, X., & Sander, J. (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 226–231. https://doi.org/10.5555/3001460.3001507

Gingerich, K., Maoh, H., & Anderson, W. (2016). Classifying the purpose of stopped truck events: An application of entropy to GPS data. *Transportation Research Part C: Emerging Technologies*, *64*, 17–27. https://doi.org/10.1016/j.trc.2016.01.002

Gong, L., Morikawa, T., Yamamoto, T., & Sato, H. (2014). Deriving Personal Trip Data from GPS Data: A Literature Review on the Existing Methodologies. *Procedia - Social and Behavioral Sciences*, *138*, 557–565. https://doi.org/10.1016/j.sbspro.2014.07.239

Gong, L., Sato, H., Yamamoto, T., Miwa, T., & Morikawa, T. (2015). Identification of activity stop locations in GPS trajectories by density-based clustering method combined with support vector machines. *Journal of Modern Transportation*, *23*(3), 202–213. https://doi.org/10.1007/s40534-015-0079-x

Gong, L., Yamamoto, T., & Morikawa, T. (2018). Identification of activity stop locations in GPS trajectories by DBSCAN-TE method combined with support vector machines. *Transportation Research Procedia*, *32*, 146–154. https://doi.org/10.1016/j.trpro.2018.10.028

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, L., Wang, G., Cai, J., & Chen, T. (2017). *Recent Advances in Convolutional Neural Networks*. arXiv. https://doi.org/10.48550/arXiv.1512.07108

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep Residual Learning for Image Recognition*. arXiv. https://doi.org/10.48550/arXiv.1512.03385

Helber, P., Bischke, B., Dengel, A., & Borth, D. (2019). EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *12*(7), 2217–2226. https://doi.org/10.1109/JSTARS.2019.2918242

Hu, J., Yang, S., Yang, Y., Xu, X., & Zhang, X. (2024). Identifying Truck Activity from Global Positioning System Data Based on Satellite Imagery. *Transportation Research Record: Journal of the Transportation Research Board*, *2679*(3), 680–696. https://doi.org/10.1177/03611981241283012

International Union of Railways. (2024). *2024 Report on Combined Transport in Europe*. https://uic.org/IMG/pdf/uic_uirr_report_2024-2.pdf

Khan, A., Sohail, A., Zahoora, U., & Qureshi, A. S. (2020). A Survey of the Recent Architectures of Deep Convolutional Neural Networks. *Artificial Intelligence Review*, *53*(8), 5455–5516. https://doi.org/10.1007/s10462-020-09825-6

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, *25*. https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., & Jackel, L. D. (1989). Handwritten Digit Recognition with a Back-Propagation Network. *Advances in Neural Information Processing Systems*, *2*. https://proceedings.neurips.cc/paper_files/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791

Lee, C.-Y., Gallagher, P. W., & Tu, Z. (2015). *Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree*. arXiv. https://doi.org/10.48550/arXiv.1509.08985

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., & Dollár, P. (2015). *Microsoft COCO: Common Objects in Context*. arXiv. https://doi.org/10.48550/arXiv.1405.0312

Link, S., & Plötz, P. (2024). Geospatial truck parking locations data for Europe. *Data in Brief*, *54*, 110277. https://doi.org/10.1016/j.dib.2024.110277

Luo, T., Zheng, X., Xu, G., Fu, K., & Ren, W. (2017). An Improved DBSCAN Algorithm to Detect Stops in Individual Trajectories. *ISPRS International Journal of Geo-Information*, *6*(3), 63. https://doi.org/10.3390/ijgi6030063

Luong, C., Do, S., Hoang, T., & Deokjai. (2015). A Method for Detecting Significant Places from GPS Trajectory Data. *Journal of Advances in Information Technology*, 44–48. https://doi.org/10.12720/jait.6.1.44-48

Milaghardan, A. H., Ali Abbaspour, R., & Claramunt, C. (2018). A Dempster-Shafer based approach to the detection of trajectory stop points. *Computers, Environment and Urban Systems*, *70*, 189–196. https://doi.org/10.1016/j.compenvurbsys.2018.03.007

Mohamed, I. S. (2017). *Detection and Tracking of Pallets using a Laser Rangefinder and Machine Learning Techniques*. https://doi.org/10.13140/RG.2.2.30795.69926

Nielsen, M. (2018). *Neural Networks and Deep Learning*. Determination Press. http://neuralnetworksanddeeplearning.com/

Nogueira, P. T., Celes, C., Martin, H., A. F. Loureiro, A., & M. C. Andrade, R. (2018). A Statistical Method for Detecting Move, Stop, and Noise: A Case Study with Bus Trajectories. *Journal of Information and Data Management*, *9*(3), 214. https://doi.org/10.5753/jidm.2018.2041

Patel, V., Maleki, M., Kargar, M., Chen, J., & Maoh, H. (2022). A cluster-driven classification approach to truck stop location identification using passive GPS data. *Journal of Geographical Systems*, *24*(4), 657–677. https://doi.org/10.1007/s10109-022-00380-y

Pedraza, A., Gallego, J., Lopez, S., Gonzalez, L., Laurinavicius, A., & Bueno, G. (2017). Glomerulus Classification with Convolutional Neural Networks. In M. Valdés Hernández & V. González-Castro (Eds.), *Medical Image Understanding and Analysis* (Vol. 723, pp. 839–849). Springer International Publishing. https://doi.org/10.1007/978-3-319-60964-5_73

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, *115*(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y

Sarti, L., Bravi, L., Sambo, F., Taccari, L., Simoncini, M., Salti, S., & Lori, A. (2017). Stop Purpose Classification from GPS Data of Commercial Vehicle Fleets. *2017*

IEEE International Conference on Data Mining Workshops (ICDMW), 280–287. https://doi.org/10.1109/ICDMW.2017.43

Siripirote, T., Sumalee, A., & Ho, H. W. (2020). Statistical estimation of freight activity analytics from Global Positioning System data of trucks. *Transportation Research Part E: Logistics and Transportation Review*, *140*, 101986. https://doi.org/10.1016/j.tre.2020.101986

Smith, H. (2020, February 27). *Geographic vs Projected Coordinate Systems*. https://www.esri.com/arcgis-blog/products/arcgis-pro/mapping/gcs_vs_pcs

Sumbul, G., Charfuelan, M., Demir, B., & Markl, V. (2019). Bigearthnet: A Large-Scale Benchmark Archive for Remote Sensing Image Understanding. *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*, 5901–5904. https://doi.org/10.1109/IGARSS.2019.8900532

Taghavi, M., Irannezhad, E., & Prato, C. G. (2019). Identifying Truck Stops from a Large Stream of GPS Data via a Hidden Markov Chain Model. *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2265–2271. https://doi.org/10.1109/ITSC.2019.8917156

Thierry, B., Chaix, B., & Kestens, Y. (2013). Detecting activity locations from raw GPS data: A novel kernel-based algorithm. *International Journal of Health Geographics*, *12*(1), 14. https://doi.org/10.1186/1476-072X-12-14

Wu, T., Shen, H., Qin, J., & Xiang, L. (2021). Extracting Stops from Spatio-Temporal Trajectories within Dynamic Contextual Features. *Sustainability*, *13*(2), 690. https://doi.org/10.3390/su13020690

Xia, G.-S., Hu, J., Hu, F., Shi, B., Bai, X., Zhong, Y., Zhang, L., & Lu, X. (2017). AID: A Benchmark Data Set for Performance Evaluation of Aerial Scene Classification. *IEEE Transactions on Geoscience and Remote Sensing*, *55*(7), 3965–3981. https://doi.org/10.1109/TGRS.2017.2685945

Yang, X., Sun, Z., Ban, X. J., & Holguín-Veras, J. (2014). Urban Freight Delivery Stop Identification with GPS Data. *Transportation Research Record: Journal of the Transportation Research Board*, *2411*(1), 55–61. https://doi.org/10.3141/2411-07

Yang, Y., Jia, B., Yan, X.-Y., Jiang, R., Ji, H., & Gao, Z. (2021). *Identifying intracity freight trip ends from heavy truck GPS trajectories*. arXiv. https://doi.org/10.48550/arXiv.2106.09881

Zhu, X. X., Hu, J., Qiu, C., Shi, Y., Kang, J., Mou, L., Bagheri, H., Häberle, M., Hua, Y., Huang, R., Hughes, L., Li, H., Sun, Y., Zhang, G., Han, S., Schmitt, M., & Wang, Y. (2019). *So2Sat LCZ42: A Benchmark Dataset for Global Local Climate Zones Classification*. arXiv. https://doi.org/10.48550/arXiv.1912.12171

List of AI-Tools

AI-Tool	Usage	Affected parts of the thesis	Comments
ChatGPT	Title suggestions	Title	The suggestions was slightly modified.
ChatGPT	Abstract draft	Abstract	By uploading the thesis as PDF the following prompt was used: “help me write the abstract for this thesis. not longer than half a page.” The suggestion was then modified.
ChatGPT	Modification of text passages	Chapters 1, 5, and 6	The AI was used to refine these chapters. This was done to reduce the extent and to handle informal phrasing. The output was then modified and adapted.
NotebookLLM	Literature analysis	Chapter 2	The AI was used to discuss specific methods and ideas from the literature to help with finalizing Chapter 2 and understanding certain concepts in machine learning.
Claude	Coding	Data extraction, data analysis, machine learning, visualizations	The implementation of all code for this thesis was achieved with the help of this AI. By telling it exactly what was needed for separate workflows, code snippets were generated, modified, and integrated into the pipeline.

Appendix A: TACID Code

Loading/Unloading Extraction

```
# Tags for data extraction and place definition
tags = {
    'building': ['warehouse', 'industrial', 'logistics'],
    'landuse': ['industrial', 'commercial'],
    'office': ['logistics', 'transport'],
    'industrial': ['logistics'],
    'access': ['delivery']
}

place = 'Italy'

# 1. Download POIs
pois = ox.features_from_place(place, tags=tags)

# 2. Convert to points if necessary
def ensure_points(gdf):
    gdf['geometry'] = gdf['geometry'].apply(
        lambda geom: geom.centroid if geom.type in ['Polygon', 'MultiPolygon']
else geom
    )
    return gdf[gdf['geometry'].type == 'Point']

pois_points = ensure_points(pois)
pois_points = pois_points[~pois_points.geometry.is_empty]

# 3. Keep relevant columns
columns_to_keep = ["geometry", "industrial", "landuse", "office", "type"]
pois_p = pois_points[columns_to_keep].copy()
pois_p = pois_p.reset_index(drop=True)

# 4. Create spatial grid (fishnet) and join POIs to grid cells
xmin, ymin, xmax, ymax = pois_p.total_bounds
n_cells = 100  # You can adjust this to control granularity
dx = (xmax - xmin) / n_cells
dy = (ymax - ymin) / n_cells

# Create grid cells
cells = []
for i in range(n_cells):
    for j in range(n_cells):
        cells.append(box(xmin + i*dx, ymin + j*dy,
                         xmin + (i+1)*dx, ymin + (j+1)*dy))

grid = gpd.GeoDataFrame({'geometry': cells}, crs=pois_p.crs)

# Spatial join: assign each POI to a grid cell
grid = grid.reset_index().rename(columns={'index': 'cell_id'})
pois_with_cell = gpd.sjoin(pois_p, grid, how='left', predicate='within')

# 5. Stratified sample: take up to k points per cell
```

```
k = 2  # Maximum points per grid cell
strat_sample = (
    pois_with_cell
    .groupby('cell_id')
    .apply(lambda grp: grp.sample(n=min(len(grp), k), random_state=0))
    .reset_index(drop=True)
)

max_total = 200
if len(strat_sample) > max_total:
    strat_sample = strat_sample.sample(n=max_total, random_state=1)

# 6. Export
output_path = r"C:\Users\..\training_POI_locations\italy_sample_new.csv"
strat_sample.to_csv(output_path, index=False, encoding="utf-8")
```

Fuel Extraction

```
# 1. Define tags for fuel stations
tags = {
    'highway': 'services'
}

place = "Germany"

# 2. Download fuel stations
fuel_pois = ox.features_from_place(place, tags=tags)

# 3. Convert polygons to centroids, keep only points
def ensure_points(gdf):
    gdf['geometry'] = gdf['geometry'].apply(
        lambda geom: geom.centroid if geom.type in ['Polygon', 'MultiPolygon']
else geom
    )
    return gdf[gdf['geometry'].type == 'Point']

fuel_points = ensure_points(fuel_pois)
fuel_points =
fuel_points[~fuel_points.geometry.is_empty].reset_index(drop=True)

# 4. Create spatial grid (fishnet) and join POIs to grid cells
xmin, ymin, xmax, ymax = fuel_points.total_bounds
n_cells = 100  # Adjust granularity of grid
dx = (xmax - xmin) / n_cells
dy = (ymax - ymin) / n_cells

# Create the grid
cells = []
for i in range(n_cells):
    for j in range(n_cells):
        cells.append(box(xmin + i*dx, ymin + j*dy, xmin + (i+1)*dx, ymin +
(j+1)*dy))

grid = gpd.GeoDataFrame({'geometry': cells}, crs=fuel_p.crs)
grid = grid.reset_index().rename(columns={'index': 'cell_id'})
```

```
# 5. Spatial join: assign each POI to a grid cell
fuel_with_cell = gpd.sjoin(fuel_points, grid, how='left', predicate='within')

# 6. Stratified sample: take up to k points per cell
k = 2  # Maximum points per cell
strat_sample = (
    fuel_with_cell
    .groupby('cell_id')
    .apply(lambda grp: grp.sample(n=min(len(grp), k), random_state=42))
    .reset_index(drop=True)
)

max_total = 250
if len(strat_sample) > max_total:
    strat_sample = strat_sample.sample(n=max_total, random_state=1)

# 8. Convert geometry to lon/lat for CSV
strat_sample["lon"] = strat_sample.geometry.x
strat_sample["lat"] = strat_sample.geometry.y
strat_sample = strat_sample.drop(columns="geometry")

# 9. Export as CSV
output_folder = r"C:\Users\..\training_POI_locations\Fuel_new"
os.makedirs(output_folder, exist_ok=True)
output_path = os.path.join(output_folder, "germany_fuel_sample_new_250.csv")

strat_sample.to_csv(output_path, index=False, encoding="utf-8")
```

Image Extraction for CSV Files

```
def get_map(access_token, lat, lon, zoom=17, width=300, height=300,
marker=False, map_style="satellite-v9"):
    """
    Fetch a satellite map image from Mapbox Static Images API.

    Args:
        access_token: Mapbox access token
        lat: Latitude
        lon: Longitude
        zoom: Zoom level (0-22)
        width: Image width in pixels (1-1280)
        height: Image height in pixels (1-1280)
        marker: Whether to add a marker at the center
        map_style: Mapbox style ID (satellite-v9, streets-v12, outdoors-v12,
etc.)
    """
    base_url = f"https://api.mapbox.com/styles/v1/mapbox/satellite-v9/static"

    # Construct the full URL
    # Format: /styles/v1/{username}/{style_id}/static/{overlay}{lon},{lat},
{zoom},{bearing},{pitch}/{width}x{height}{@2x}
    url = f"{base_url}/{overlay}{lon},{lat},{zoom},0,0/{width}x{height}"

    params = {
```

```
        "access_token": access_token
    }

    response = requests.get(url, params=params)

    if response.status_code != 200:
        raise RuntimeError(f"Failed to fetch map for ({lat}, {lon}):
{response.status_code} {response.text}")

    return Image.open(BytesIO(response.content))


def fetch_images_from_csv(csv_path, access_token, output_dir, zoom=17,
limit=None, sleep_time=1, map_style="satellite-v9"):
    """
    Read coordinates from CSV and download map images using Mapbox.

    Args:
        csv_path: Path to CSV file with 'lat' and 'lon' columns
        access_token: Mapbox access token
        output_dir: Directory to save images
        zoom: Zoom level for maps
        limit: Maximum number of images to fetch (None for all)
        sleep_time: Seconds to wait between requests
        map_style: Mapbox map style
    """
    df = pd.read_csv(csv_path)

    # Ensure output directory exists
    os.makedirs(output_dir, exist_ok=True)

    if limit:
        df = df.head(limit)

    for idx, row in df.iterrows():
        lat = row['lat']
        lon = row['lon']
        try:
            img = get_map(access_token, lat, lon, zoom=zoom,
map_style=map_style)
            filename = f"fuel_{idx + 1}.png"
            filepath = os.path.join(output_dir, filename)
            img.save(filepath)
            print(f"[{idx + 1}/{len(df)}] Saved: {filename}")
            time.sleep(sleep_time)  # avoid hitting rate limits
        except Exception as e:
            print(f"[{idx + 1}/{len(df)}] Failed for ({lat}, {lon}): {e}")


# === Run script ===
if __name__ == "__main__":
    # Replace with your Mapbox access token
    ACCESS_TOKEN = 'p..g'  # Your Mapbox access token
    CSV_PATH = r"C:\Users\..\fuel_europe_locations.csv" # adapt to file
    OUTPUT_DIR = r"C:\Users\..\dataset\fuel" # adapt to class

    fetch_images_from_csv(
```

```
        CSV_PATH,
        ACCESS_TOKEN,
        OUTPUT_DIR,
        zoom=17,
        limit=1500,
        sleep_time=1,
        map_style="satellite-v9"
    )
```

Appendix B: Model Training Code

Three-Class Model

```
# Set device to CPU since I don't have a NVIDIA GPU
device = torch.device('cpu')
print(f"Using device: {device}")

# Set random seeds for reproducibility
torch.manual_seed(42)
np.random.seed(42)

# Data directory as Path object for easier path handling
DATA_DIR = Path(r"C:\Users\..\three_class\dataset

# Hyperparameters
BATCH_SIZE = 16 # Smaller batch size for CPU training
LEARNING_RATE = 0.001
NUM_EPOCHS = 20
NUM_CLASSES = 3
NUM_WORKERS = 0

# Image parameters
IMG_SIZE = 224

# Class names
CLASS_NAMES = ['loading', 'rest', 'traffic']

# ImageNet normalization values
IMAGENET_MEAN = [0.485, 0.456, 0.406]
IMAGENET_STD  = [0.229, 0.224, 0.225]

# Definition of Data Augmentation for training and evaluation datasets
train_tfms = transforms.Compose([
 transforms.RandomResizedCrop(IMG_SIZE, scale=(0.8, 1.0)),
 transforms.RandomHorizontalFlip(p=0.5),
 transforms.RandomRotation(degrees=10),
 transforms.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1),
 transforms.RandomAffine(degrees=0, translate=(0.05, 0.05), shear=5),
 transforms.GaussianBlur(kernel_size=3, sigma=(0.1, 2.0)),
 transforms.ToTensor(),
 transforms.Normalize(IMAGENET_MEAN, IMAGENET_STD),
])

eval_tfms = transforms.Compose([
 transforms.Resize(int(IMG_SIZE * 1.14)),
 transforms.CenterCrop(IMG_SIZE),
 transforms.ToTensor(),
 transforms.Normalize(IMAGENET_MEAN, IMAGENET_STD),
])

# Create datasets
train_ds = datasets.ImageFolder(DATA_DIR / "train", transform=train_tfms)
val_ds   = datasets.ImageFolder(DATA_DIR / "val",   transform=eval_tfms)
```

```
test_ds  = datasets.ImageFolder(DATA_DIR / "test",  transform=eval_tfms)

# Create data loaders
train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True,
num_workers=NUM_WORKERS, pin_memory=False)
val_loader = DataLoader(val_ds,   batch_size=BATCH_SIZE, shuffle=False,
num_workers=NUM_WORKERS, pin_memory=False)
test_loader = DataLoader(test_ds,  batch_size=BATCH_SIZE, shuffle=False,
num_workers=NUM_WORKERS, pin_memory=False)

# Load pretrained ResNet50
model = models.resnet50(weights=models.ResNet50_Weights.IMAGENET1K_V2)

# Replace the final fully connected layer
num_features = model.fc.in_features
model.fc = nn.Sequential(
    nn.Dropout(0.4),
    nn.Linear(num_features, NUM_CLASSES)
)

model = model.to(device)

# Loss function
criterion = nn.CrossEntropyLoss()

# Optimizer - Adam for fine-tuning
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE, weight_decay=1e-3)

# Learning rate scheduler (optional but recommended)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)

# Training tracking
train_losses = []
train_accuracies = []
val_losses = []
val_accuracies = []
learning_rates = []

best_val_acc = 0.0
best_epoch = 0

def calculate_accuracy(outputs, labels):
    _, predicted = torch.max(outputs, 1)
    correct = (predicted == labels).sum().item()
    return correct / labels.size(0)

def freeze_backbone(model):
    """Freeze all layers except the final classifier"""
    for name, param in model.named_parameters():
        if 'fc' not in name:  # Don't freeze the final fully connected layer
            param.requires_grad = False

def unfreeze_backbone(model):
    """Unfreeze all layers"""
    for param in model.parameters():
        param.requires_grad = True
```

```
# Stage 1: Train only the classifier
print("=== STAGE 1: Training classifier only ===")
freeze_backbone(model)

# Reset optimizer for stage 1
optimizer_stage1 = optim.Adam(model.parameters(), lr=LEARNING_RATE,
weight_decay=1e-4)
scheduler_stage1 = optim.lr_scheduler.StepLR(optimizer_stage1, step_size=5,
gamma=0.5)

# Check trainable parameters
trainable_params = sum(p.numel() for p in model.parameters() if
p.requires_grad)

# Early stopping parameters
class EarlyStopping:
    def __init__(self, patience=5, min_delta=0.001):
        self.patience = patience
        self.min_delta = min_delta
        self.counter = 0
        self.best_loss = float('inf')

    def __call__(self, val_loss):
        if val_loss < self.best_loss - self.min_delta:
            self.best_loss = val_loss
            self.counter = 0
            return False  # Don't stop
        else:
            self.counter += 1
            return self.counter >= self.patience  # Stop if patience exceeded

# Stage 1 training loop
for epoch in range(NUM_EPOCHS//2):  # First half of epochs
    start_time = time.time()

    # Training phase
    model.train()
    train_loss = 0.0
    train_correct = 0
    train_total = 0

    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer_stage1.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer_stage1.step()

        train_loss += loss.item()
        train_correct += (torch.max(outputs, 1)[1] == labels).sum().item()
        train_total += labels.size(0)

    # Validation phase
    model.eval()
    val_loss = 0.0
```

```
    val_correct = 0
    val_total = 0

    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)

            val_loss += loss.item()
            val_correct += (torch.max(outputs, 1)[1] == labels).sum().item()
            val_total += labels.size(0)

    # Calculate averages
    train_loss_avg = train_loss / len(train_loader)
    train_acc = train_correct / train_total
    val_loss_avg = val_loss / len(val_loader)
    val_acc = val_correct / val_total

    # Store metrics
    train_losses.append(train_loss_avg)
    train_accuracies.append(train_acc)
    val_losses.append(val_loss_avg)
    val_accuracies.append(val_acc)
    learning_rates.append(optimizer_stage1.param_groups[0]['lr'])

    # Update best model
    if val_acc > best_val_acc:
        best_val_acc = val_acc
        best_epoch = epoch
        torch.save(model.state_dict(), 'best_model_stage1_V5.pth')

    # Early Stopping
    if early_stopping_stage1(val_loss_avg):
        print(f"Early stopping triggered at epoch {epoch+1}")
        break

    # Step scheduler
    scheduler_stage1.step()

print("Loading best model from stage 1...")
model.load_state_dict(torch.load('best_model_stage1_V5.pth'))
model.eval()

# Quick validation to confirm we loaded the right weights
val_correct = 0
val_total = 0
with torch.no_grad():
    for inputs, labels in val_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        val_correct += (torch.max(outputs, 1)[1] == labels).sum().item()
        val_total += labels.size(0)

loaded_val_acc = val_correct / val_total
```

```
print("=== STAGE 2: Fine-tuning entire network ===")

# Unfreeze all layers
unfreeze_backbone(model)

# Use different learning rates: lower for backbone, higher for classifier
backbone_lr = LEARNING_RATE / 10  # 10x lower for pretrained layers
classifier_lr = LEARNING_RATE / 5  # 5x lower for classifier (already trained)

optimizer_stage2 = optim.Adam([
    {'params': [p for n, p in model.named_parameters() if 'fc' not in n], 'lr':
backbone_lr},
    {'params': model.fc.parameters(), 'lr': classifier_lr}
], weight_decay=1e-3)

scheduler_stage2 = optim.lr_scheduler.StepLR(optimizer_stage2, step_size=5,
gamma=0.5)

trainable_params = sum(p.numel() for p in model.parameters() if
p.requires_grad)

# Continue tracking from stage 1
stage1_epochs = len(train_losses)
print(f"Continuing from epoch {stage1_epochs}, starting stage 2...")

# Stage 2 training loop
for epoch in range(NUM_EPOCHS//2):  # Second half of epochs
    start_time = time.time()
    actual_epoch = stage1_epochs + epoch

    # Training phase
    model.train()
    train_loss = 0.0
    train_correct = 0
    train_total = 0

    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer_stage2.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer_stage2.step()

        train_loss += loss.item()
        train_correct += (torch.max(outputs, 1)[1] == labels).sum().item()
        train_total += labels.size(0)

    # Validation phase
    model.eval()
    val_loss = 0.0
    val_correct = 0
    val_total = 0

    with torch.no_grad():
        for inputs, labels in val_loader:
```

```
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)

            val_loss += loss.item()
            val_correct += (torch.max(outputs, 1)[1] == labels).sum().item()
            val_total += labels.size(0)

    # Calculate averages
    train_loss_avg = train_loss / len(train_loader)
    train_acc = train_correct / train_total
    val_loss_avg = val_loss / len(val_loader)
    val_acc = val_correct / val_total

    # Store metrics
    train_losses.append(train_loss_avg)
    train_accuracies.append(train_acc)
    val_losses.append(val_loss_avg)
    val_accuracies.append(val_acc)
    learning_rates.append(optimizer_stage2.param_groups[0]['lr'])  # backbone
LR

    # Update best model
    if val_acc > best_val_acc:
        best_val_acc = val_acc
        best_epoch = actual_epoch
        torch.save(model.state_dict(), 'best_model_final_V5.pth')

    # Early Stopping
    if early_stopping_stage2(val_loss_avg):
        print(f"Early stopping triggered at epoch {actual_epoch+1}")
        break

    # Step scheduler
    scheduler_stage2.step()

# Load the best model
print("Loading best model for testing...")
model.load_state_dict(torch.load('best_model_final_V3.pth'))
model.eval()

# Test set evaluation
test_loss = 0.0
test_correct = 0
test_total = 0
all_predictions = []
all_labels = []

print("Evaluating on test set...")
with torch.no_grad():
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        test_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
```

```
        test_correct += (predicted == labels).sum().item()
        test_total += labels.size(0)

        # Store for confusion matrix
        all_predictions.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# Calculate final metrics
test_loss_avg = test_loss / len(test_loader)
test_accuracy = test_correct / test_total

print(f"\n=== FINAL TEST RESULTS ===")
print(f"Test Loss: {test_loss_avg:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f} ({test_accuracy*100:.2f}%)")
print(f"Correct predictions: {test_correct}/{test_total}")

# Per-class accuracy
from collections import Counter
correct_per_class = Counter()
total_per_class = Counter()

for true_label, pred_label in zip(all_labels, all_predictions):
    total_per_class[true_label] += 1
    if true_label == pred_label:
        correct_per_class[true_label] += 1

print(f"\n=== PER-CLASS ACCURACY ===")
for class_idx, class_name in enumerate(CLASS_NAMES):
    if total_per_class[class_idx] > 0:
        class_acc = correct_per_class[class_idx] / total_per_class[class_idx]
        print(f"{class_name}: {class_acc:.4f} ({class_acc*100:.2f}%) -
{correct_per_class[class_idx]}/{total_per_class[class_idx]}")
```

Four-Class Model

```
# Set device to CPU since I don't have a NVIDIA GPU
device = torch.device('cpu')
print(f"Using device: {device}")

# Set random seeds for reproducibility
torch.manual_seed(42)
np.random.seed(42)

# Data directory as Path object for easier path handling
DATA_DIR = Path(r"C:\Users\..\four_class\dataset")

# Hyperparameters
BATCH_SIZE = 16
LEARNING_RATE = 0.001
NUM_EPOCHS = 20
NUM_CLASSES = 4
NUM_WORKERS = 0

# Image parameters
IMG_SIZE = 224
```

```
# Class names
CLASS_NAMES = ['fuel', 'loading', 'rest', 'traffic']

# ImageNet normalization values
IMAGENET_MEAN = [0.485, 0.456, 0.406]
IMAGENET_STD  = [0.229, 0.224, 0.225]

# Definition of Data Augmentation for training and evaluation datasets
train_tfms = transforms.Compose([
    transforms.RandomResizedCrop(IMG_SIZE, scale=(0.8, 1.0)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=10),
    transforms.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1),
    transforms.RandomAffine(degrees=0, translate=(0.05, 0.05), shear=5),
    transforms.GaussianBlur(kernel_size=3, sigma=(0.1, 2.0)),
    transforms.ToTensor(),
    transforms.Normalize(IMAGENET_MEAN, IMAGENET_STD),
])

eval_tfms = transforms.Compose([
    transforms.Resize(int(IMG_SIZE * 1.14)),
    transforms.CenterCrop(IMG_SIZE),
    transforms.ToTensor(),
    transforms.Normalize(IMAGENET_MEAN, IMAGENET_STD),
])

# Create datasets
train_ds = datasets.ImageFolder(DATA_DIR / "train", transform=train_tfms)
val_ds   = datasets.ImageFolder(DATA_DIR / "val",   transform=eval_tfms)
test_ds  = datasets.ImageFolder(DATA_DIR / "test",  transform=eval_tfms)

# Create data loaders
train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True,
num_workers=NUM_WORKERS, pin_memory=False)
val_loader   = DataLoader(val_ds,   batch_size=BATCH_SIZE, shuffle=False,
num_workers=NUM_WORKERS, pin_memory=False)
test_loader  = DataLoader(test_ds,  batch_size=BATCH_SIZE, shuffle=False,
num_workers=NUM_WORKERS, pin_memory=False)

# Load pretrained ResNet50
model = models.resnet50(weights=models.ResNet50_Weights.IMAGENET1K_V2)

# Replace the final fully connected layer
num_features = model.fc.in_features
model.fc = nn.Sequential(
    nn.Dropout(0.5),
    nn.Linear(num_features, NUM_CLASSES)
)

model = model.to(device)

# Loss function
criterion = nn.CrossEntropyLoss()

# Optimizer - Adam for fine-tuning
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE, weight_decay=1e-3)
```

```
# Learning rate scheduler (optional but recommended)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)

# Training tracking
train_losses = []
train_accuracies = []
val_losses = []
val_accuracies = []
learning_rates = []

best_val_acc = 0.0
best_epoch = 0

def calculate_accuracy(outputs, labels):
    _, predicted = torch.max(outputs, 1)
    correct = (predicted == labels).sum().item()
    return correct / labels.size(0)

def freeze_backbone(model):
    """Freeze all layers except the final classifier"""
    for name, param in model.named_parameters():
        if 'fc' not in name:  # Don't freeze the final fully connected layer
            param.requires_grad = False

def unfreeze_backbone(model):
    """Unfreeze all layers"""
    for param in model.parameters():
        param.requires_grad = True

# Stage 1: Train only the classifier
print("=== STAGE 1: Training classifier only ===")
freeze_backbone(model)

# Reset optimizer for stage 1
optimizer_stage1 = optim.Adam(model.parameters(), lr=LEARNING_RATE,
weight_decay=1e-4)
scheduler_stage1 = optim.lr_scheduler.StepLR(optimizer_stage1, step_size=5,
gamma=0.5)

# Check trainable parameters
trainable_params = sum(p.numel() for p in model.parameters() if
p.requires_grad)

# Early stopping parameters
class EarlyStopping:
    def __init__(self, patience=5, min_delta=0.001):
        self.patience = patience
        self.min_delta = min_delta
        self.counter = 0
        self.best_loss = float('inf')

    def __call__(self, val_loss):
        if val_loss < self.best_loss - self.min_delta:
            self.best_loss = val_loss
            self.counter = 0
            return False  # Don't stop
        else:
```

```
            self.counter += 1
            return self.counter >= self.patience  # Stop if patience exceeded

# Initialize early stopping for both stages
early_stopping_stage1 = EarlyStopping(patience=5, min_delta=0.001)
early_stopping_stage2 = EarlyStopping(patience=3, min_delta=0.001)

# Stage 1 training loop
for epoch in range(NUM_EPOCHS//2):  # First half of epochs
    start_time = time.time()

    # Training phase
    model.train()
    train_loss = 0.0
    train_correct = 0
    train_total = 0

    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer_stage1.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer_stage1.step()

        train_loss += loss.item()
        train_correct += (torch.max(outputs, 1)[1] == labels).sum().item()
        train_total += labels.size(0)

    # Validation phase
    model.eval()
    val_loss = 0.0
    val_correct = 0
    val_total = 0

    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)

            val_loss += loss.item()
            val_correct += (torch.max(outputs, 1)[1] == labels).sum().item()
            val_total += labels.size(0)

    # Calculate averages
    train_loss_avg = train_loss / len(train_loader)
    train_acc = train_correct / train_total
    val_loss_avg = val_loss / len(val_loader)
    val_acc = val_correct / val_total

    # Store metrics
    train_losses.append(train_loss_avg)
    train_accuracies.append(train_acc)
    val_losses.append(val_loss_avg)
    val_accuracies.append(val_acc)
```

```
    learning_rates.append(optimizer_stage1.param_groups[0]['lr'])

    # Update best model
    if val_acc > best_val_acc:
        best_val_acc = val_acc
        best_epoch = epoch
        torch.save(model.state_dict(), 'best_model_stage1_4class_V2.pth')

    # Early Stopping
    if early_stopping_stage1(val_loss_avg):
        print(f"Early stopping triggered at epoch {epoch+1}")
        break

    # Step scheduler
    scheduler_stage1.step()

print("Loading best model from stage 1...")
model.load_state_dict(torch.load('best_model_stage1_4class_V2.pth'))
model.eval()

# Quick validation to confirm we loaded the right weights
val_correct = 0
val_total = 0
with torch.no_grad():
    for inputs, labels in val_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        val_correct += (torch.max(outputs, 1)[1] == labels).sum().item()
        val_total += labels.size(0)

loaded_val_acc = val_correct / val_total

print("=== STAGE 2: Fine-tuning entire network ===")

# Unfreeze all layers
unfreeze_backbone(model)

# Use different learning rates: lower for backbone, higher for classifier
backbone_lr = LEARNING_RATE / 10  # 10x lower for pretrained layers
classifier_lr = LEARNING_RATE / 5  # 5x lower for classifier (already trained)

optimizer_stage2 = optim.Adam([
    {'params': [p for n, p in model.named_parameters() if 'fc' not in n], 'lr':
backbone_lr},
    {'params': model.fc.parameters(), 'lr': classifier_lr}
], weight_decay=1e-3)

scheduler_stage2 = optim.lr_scheduler.StepLR(optimizer_stage2, step_size=5,
gamma=0.5)

trainable_params = sum(p.numel() for p in model.parameters() if
p.requires_grad)

# Continue tracking from stage 1
stage1_epochs = len(train_losses)
print(f"Continuing from epoch {stage1_epochs}, starting stage 2...")
```

```
# Stage 2 training loop
for epoch in range(NUM_EPOCHS//2):  # Second half of epochs
    start_time = time.time()
    actual_epoch = stage1_epochs + epoch

    # Training phase
    model.train()
    train_loss = 0.0
    train_correct = 0
    train_total = 0

    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer_stage2.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer_stage2.step()

        train_loss += loss.item()
        train_correct += (torch.max(outputs, 1)[1] == labels).sum().item()
        train_total += labels.size(0)

    # Validation phase
    model.eval()
    val_loss = 0.0
    val_correct = 0
    val_total = 0

    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)

            val_loss += loss.item()
            val_correct += (torch.max(outputs, 1)[1] == labels).sum().item()
            val_total += labels.size(0)

    # Calculate averages
    train_loss_avg = train_loss / len(train_loader)
    train_acc = train_correct / train_total
    val_loss_avg = val_loss / len(val_loader)
    val_acc = val_correct / val_total

    # Store metrics
    train_losses.append(train_loss_avg)
    train_accuracies.append(train_acc)
    val_losses.append(val_loss_avg)
    val_accuracies.append(val_acc)
    learning_rates.append(optimizer_stage2.param_groups[0]['lr'])  # backbone
LR

    # Update best model
    if val_acc > best_val_acc:
        best_val_acc = val_acc
```

```
        best_epoch = actual_epoch
        torch.save(model.state_dict(), 'best_model_final_4class_V2.pth')

    # Early Stopping
    if early_stopping_stage2(val_loss_avg):
        print(f"Early stopping triggered at epoch {actual_epoch+1}")
        break

    # Step scheduler
    scheduler_stage2.step()

# Load the best model
print("Loading best model for testing...")
model.load_state_dict(torch.load('best_model_final_4class_V2.pth'))
model.eval()

# Test set evaluation
test_loss = 0.0
test_correct = 0
test_total = 0
all_predictions = []
all_labels = []

print("Evaluating on test set...")
with torch.no_grad():
 for inputs, labels in test_loader:
 inputs, labels = inputs.to(device), labels.to(device)
 outputs = model(inputs)
 loss = criterion(outputs, labels)
 loss += loss.item()
 _, predicted = torch.max(outputs, 1)
 test_correct += (predicted == labels).sum().item()
 test_total += labels.size(0)
 re for confusion matrix
 all_predictions.extend(predicted.cpu().numpy())
 all_labels.extend(labels.cpu().numpy())

# Calculate final metrics
test_loss_avg = test_loss / len(test_loader)
test_accuracy = test_correct / test_total

print(f"\n=== FINAL TEST RESULTS ===")
print(f"Test Loss: {test_loss_avg:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f} ({test_accuracy*100:.2f}%)")
print(f"Correct predictions: {test_correct}/{test_total}")

# Per-class accuracy
from collections import Counter
correct_per_class = Counter()
total_per_class = Counter()

for true_label, pred_label in zip(all_labels, all_predictions):
 total_per_class[true_label] += 1
 if true_label == pred_label:
 correct_per_class[true_label] += 1

print(f"\n=== PER-CLASS ACCURACY ===")
```

```
for class_idx, class_name in enumerate(CLASS_NAMES):
    if total_per_class[class_idx] > 0:
        class_acc = correct_per_class[class_idx] / total_per_class[class_idx]
        print(f"{class_name}: {class_acc:.4f} ({class_acc*100:.2f}%) -
{correct_per_class[class_idx]}/{total_per_class[class_idx]}")
```

Appendix C: GNSS Data Pipeline Code

Data Pipeline

```
# Cell 2: Define input and output directories
input_dir = r"C:\Users\..\lastWeek_CSV"
output_dir = r"C:\Users\..\data_processed"

# Ensure the output directory exists
os.makedirs(output_dir, exist_ok=True)

# Get all CSV files in the input directory
csv_files = [f for f in os.listdir(input_dir) if f.endswith('.csv')]

# Speed threshold for filtering
speed_threshold = 5.0
speed_error_threshold = 120.0

# Initialize counters for point summary
total_original_points = 0
total_filtered_points = 0
total_speed_errors_removed = 0

# Lists to store data for visualization
file_names = []
original_points_list = []
filtered_points_list = []

# Lists to store frequency data
avg_freq_list = []
max_freq_list = []

# Create list to store all processed dataframes
all_files_data = []
total_missing_speed = 0
total_points_loaded = 0

for idx, filename in enumerate(csv_files, 1):
    #print(f"Processing file {idx}/{len(csv_files)}: {filename}")

    file_path = os.path.join(input_dir, filename)

    try:
        # Load the file
        df = pd.read_csv(file_path)

        # Keep only relevant columns
        df = df[['timestamp', 'latitude', 'longitude', 'speed']].copy()

        # Count total points
        total_points_loaded += len(df)

        # Count missing speed values
        missing_speed = df['speed'].isna().sum()
```

```
        total_missing_speed += missing_speed

        # Handle missing speed values
        df['speed'] = df['speed'].fillna(0)

        # Add TID column from filename
        tid = os.path.splitext(filename)[0].split('_')[0]
        df['TID'] = tid

        # Add a new 'activity' column for later
        df['activity'] = None

        # Convert timestamp to datetime
        df["timestamp"] = pd.to_datetime(df["timestamp"])

        # Sort df by timestamp
        df = df.sort_values("timestamp").reset_index(drop=True)

        # Calculate temporal metrics
        df["time_diff"] = df["timestamp"].diff()
        df["freq_sec"] = df["time_diff"].dt.total_seconds()
        df['elapsed_time'] = df['timestamp'] - df['timestamp'].iloc[0]
        df['elapsed_seconds'] = df['elapsed_time'].dt.total_seconds()

        # Store processed dataframe
        all_files_data.append(df)
        file_names.append(filename)

    except Exception as e:
        print(f"  ERROR processing {filename}: {str(e)}")
        continue

# Cell 5: Remove speed errors (speed > 120 km/h)
all_files_cleaned = []

for idx, df in enumerate(all_files_data, 1):
    # Count points before removal
    points_before = len(df)

    # Mark speed errors before removing (filtering)
    df.loc[df['speed'] > speed_error_threshold, 'activity'] = 'speed error'

    # Filter out speed errors (keep only speed <= 120 km/h)
    df_cleaned = df[df['speed'] <= speed_error_threshold].copy()

    # Reset index after filtering
    df_cleaned = df_cleaned.reset_index(drop=True)

    # Count points after removal
    points_after = len(df_cleaned)
    points_removed = points_before - points_after
    total_speed_errors_removed += points_removed

    # Store cleaned dataframe
    all_files_cleaned.append(df_cleaned)
    original_points_list.append(points_after)
```

```
# Cell 6: Reproject coordinates from WGS84 to ETRS89
all_files_reprojected = []

for idx, df in enumerate(all_files_cleaned, 1):
    #print(f"Reprojecting file {idx}/{len(all_files_cleaned)}: {file_names[idx-
1]}")

    # Create GeoDataFrame from lat/lon coordinates
    gdf = gpd.GeoDataFrame(
        df,
        geometry=gpd.points_from_xy(df.longitude, df.latitude),
        crs="EPSG:4326"  # WGS84
    )

    # Reproject to ETRS89-LAEA
    gdf = gdf.to_crs("EPSG:3035")

    # Extract x and y coordinates
    df['x'] = gdf.geometry.x
    df['y'] = gdf.geometry.y

    # Store reprojected dataframe
    all_files_reprojected.append(df)

# Cell 7: Apply speed threshold to identify stop candidates
all_files_stops = []

for idx, df in enumerate(all_files_reprojected, 1):
    # Mark points above speed threshold as 'driving'
    df.loc[df['speed'] >= speed_threshold, 'activity'] = 'driving'

    # Apply speed threshold for stop candidate identification
    df_pot_stops = df[df['speed'] < speed_threshold].copy()
    df_pot_stops = df_pot_stops.reset_index(drop=True)

    # Store filtered data
    all_files_stops.append(df_pot_stops)
    filtered_points_list.append(len(df_pot_stops))

    # Update totals
    total_original_points += len(df)
    total_filtered_points += len(df_pot_stops)

# Cell 8: Spatial DBSCAN Cluster Analysis

print("="*60)
print("SPATIAL DBSCAN CLUSTER ANALYSIS")
print("="*60)

# Define DBSCAN parameters
eps_spatial = 150  # meters
min_samples = computed_min_samples

# Store results
all_files_clustered = []
total_clusters = 0
total_noise_points = 0
```

```
for idx, df in enumerate(all_files_stops, 1):
    if len(df) == 0:
        print(f"File {idx}: {file_names[idx-1]} - No stop candidates,
skipping")
        all_files_clustered.append(df)
        continue

    print(f"Processing file {idx}/{len(all_files_stops)}: {file_names[idx-1]}")

    try:
        # Create a copy to avoid modifying original
        df_copy = df.copy()

        # Prepare spatial data only (x, y coordinates)
        data = df_copy[['x', 'y']].values

        print(f"  Data shape: {data.shape}")
        print(f"  Data range - X: [{data[:,0].min():.0f},
{data[:,0].max():.0f}], Y: [{data[:,1].min():.0f}, {data[:,1].max():.0f}]")

        N = len(df_copy)
        computed_min_samples = max(1, int(np.ceil(np.log(N))))

        # Create DBSCAN instance
        dbscan = DBSCAN(eps=eps_spatial, min_samples=computed_min_samples)

        # Fit the model
        print(f"  Fitting DBSCAN...")
        labels = dbscan.fit_predict(data)

        print(f"  Labels type: {type(labels)}, shape/len: {len(labels)}")

        # Assign cluster labels
        df_copy['cluster'] = labels

        # Count clusters and noise
        unique_labels = set(labels)
        n_clusters = len(unique_labels) - (1 if -1 in unique_labels else 0)
        n_noise = np.sum(labels == -1)

        total_clusters += n_clusters
        total_noise_points += n_noise

        print(f"  Points: {len(df_copy)}")
        print(f"  Clusters found: {n_clusters}")
        print(f"  Noise points: {n_noise} ({n_noise/len(df_copy)*100:.1f}%)\n")

        # Store clustered dataframe
        all_files_clustered.append(df_copy)

    except Exception as e:
        print(f"  ERROR during clustering: {str(e)}")
        print(f"  Exception type: {type(e).__name__}")
        import traceback
        traceback.print_exc()
        # Store unclustered dataframe
```

```
        all_files_clustered.append(df.copy())
        continue

# Cell 9: Label noise as 'traffic', filter out, and get cluster centroids
print("="*60)
print("FILTERING NOISE AND CALCULATING CLUSTER CENTROIDS")
print("="*60)

all_files_clusters_only = []
all_cluster_centroids = []

total_traffic_points = 0
total_clusters_remaining = 0

for idx, df in enumerate(all_files_clustered, 1):
    if len(df) == 0:
        print(f"File {idx}: {file_names[idx-1]} - Empty dataframe, skipping")
        all_files_clusters_only.append(df)
        continue

    # Check if cluster column exists
    if 'cluster' not in df.columns:
        print(f"File {idx}: {file_names[idx-1]} - No cluster column, skipping")
        all_files_clusters_only.append(df)
        continue

    # Label noise points as 'traffic'
    noise_mask = df['cluster'] == -1
    n_noise = noise_mask.sum()
    df.loc[noise_mask, 'activity'] = 'traffic'
    total_traffic_points += n_noise

    # Filter out noise points (keep only valid clusters)
    df_clusters = df[df['cluster'] != -1].copy()
    df_clusters = df_clusters.reset_index(drop=True)

    # Calculate cluster centroids
    if len(df_clusters) > 0:
        cluster_stats = df_clusters.groupby('cluster').agg({
            'latitude': 'mean',
            'longitude': 'mean',
            'x': 'mean',
            'y': 'mean',
            'cluster': 'count'  # number of points per cluster
        }).rename(columns={'cluster': 'n_points'})

        # Add TID information
        cluster_stats['TID'] = df_clusters['TID'].iloc[0]

        # Reset index to make cluster a column
        cluster_stats = cluster_stats.reset_index()

        n_clusters = len(cluster_stats)
        total_clusters_remaining += n_clusters

        print(f"File {idx}/{len(all_files_clustered)}: {file_names[idx-1]}")
        print(f"  Traffic points (noise) removed: {n_noise}")
```

```
        print(f"  Clusters remaining: {n_clusters}")
        print(f"  Points in clusters: {len(df_clusters)}")

        # Store cluster centroids
        all_cluster_centroids.append(cluster_stats)
    else:
        print(f"File {idx}/{len(all_files_clustered)}: {file_names[idx-1]}")
        print(f"  Traffic points (noise) removed: {n_noise}")
        print(f"  No clusters remaining")

    # Store filtered dataframe
    all_files_clusters_only.append(df_clusters)

print("="*60)
print("SUMMARY")
print("="*60)
print(f"Total traffic points removed: {total_traffic_points:,}")
print(f"Total clusters remaining: {total_clusters_remaining}")
print(f"Total points in clusters: {sum(len(df) for df in
all_files_clusters_only):,}")

# Combine all cluster centroids into one dataframe
if all_cluster_centroids:
    df_all_centroids = pd.concat(all_cluster_centroids, ignore_index=True)
    print(f"\nTotal centroid records: {len(df_all_centroids)}")
    print("\nCentroid dataframe columns:")
    print(df_all_centroids.columns.tolist())
    print("\nFirst few centroids:")
    print(df_all_centroids.head())
else:
    df_all_centroids = pd.DataFrame()

# Cell 10: Export cluster centroids to CSV
print("="*60)
print("EXPORTING CLUSTER CENTROIDS")
print("="*60)

if len(all_cluster_centroids) > 0 and len(df_all_centroids) > 0:
    # Define output filename
    centroids_filename = "cluster_centroids.csv"
    centroids_path = os.path.join(output_dir, centroids_filename)

    # Export to CSV
    df_all_centroids.to_csv(centroids_path, index=False)

    print(f"Cluster centroids exported successfully!")
    print(f"File: {centroids_filename}")
    print(f"Path: {centroids_path}")
    print(f"Total centroids: {len(df_all_centroids)}")
    print(f"\nColumns exported: {list(df_all_centroids.columns)}")
    print(f"\nFile size: {os.path.getsize(centroids_path) / 1024:.2f} KB")
else:
    print("No cluster centroids to export (no clusters found)")

print("="*60)
```

```
# --- 1) Load + parse classification results ------------------------------
classification_file = os.path.join(output_dir, "classification_results.csv")
dfc = pd.read_csv(classification_file, dtype=str)

# drop literal header rows if the file was concatenated multiple times
hdr_mask = (dfc['filename'] == 'filename') | (dfc['predicted_label'] ==
'predicted_label')
dfc = dfc[~hdr_mask].copy()

# TID from filename (everything before "_cluster_"), cluster id after it
dfc['TID'] = dfc['filename'].str.split('_cluster_').str[0]
dfc['cluster'] = (
    dfc['filename'].str.extract(r'_cluster_(\d+(?:\.\d+)?)')[0]
    .astype(float).astype(int)
)

# normalize helpers
def norm_tid(s: pd.Series) -> pd.Series:
    return s.astype(str).str.replace(r'\.0$', '', regex=True)

dfc['TID'] = norm_tid(dfc['TID'])
dfc['predicted_class'] = pd.to_numeric(dfc['predicted_class'],
errors='coerce').astype('Int64')
dfc['confidence'] = pd.to_numeric(dfc['confidence'], errors='coerce')
dfc =
dfc[['TID','cluster','predicted_class','confidence','predicted_label']].drop_du
plicates()

# --- 2) Gather all DBSCAN-labeled point data -----------------------------
#   all_files_clustered: each df already has a 'cluster' column (with -1 for
noise)
dfs_with_cluster = [df for df in all_files_clustered if isinstance(df,
pd.DataFrame) and 'cluster' in df.columns]
df_points = pd.concat(dfs_with_cluster, ignore_index=True)

# --- 3) Keep only points that belong to actual clusters (not noise) -------
df_points_in_clusters = df_points[df_points['cluster'] != -1].copy()

# normalize join keys
df_points_in_clusters['TID'] = norm_tid(df_points_in_clusters['TID'])
df_points_in_clusters['cluster'] =
pd.to_numeric(df_points_in_clusters['cluster'], errors='coerce').astype(int)

# --- 4) Merge predicted labels onto those clustered points ----------------
df_points_labeled = df_points_in_clusters.merge(dfc, on=['TID','cluster'],
how='left')

# --- 5) Write labels into 'activity' for clustered points -----------------
# keep existing 'driving' (speed >= threshold) and 'traffic' (noise) untouched
# here we only touch rows we selected (cluster != -1), so just set to predicted
label
df_points_labeled['activity'] = df_points_labeled['predicted_label'].where(
    df_points_labeled['predicted_label'].notna(),
    df_points_labeled.get('activity')  # fallback to prior activity if any
)

print("Rows in clusters:", len(df_points_in_clusters))
```

```
print("Labeled rows:", df_points_labeled['predicted_label'].notna().sum())
print("Still-unlabeled rows (check TID/cluster mismatch):",
      df_points_labeled['predicted_label'].isna().sum())

# (optional) Save
labeled_points_path = os.path.join(output_dir,
"points_with_cluster_labels.csv")
df_points_labeled.to_csv(labeled_points_path, index=False)
print("Saved:", labeled_points_path)

print(df_points_labeled.head(10))
df_points_labeled.tail(10)

# --- Step 1: concatenate all original files (Cell 4) ---
df_all = pd.concat(all_files_data, ignore_index=True)
print("Original rows:", len(df_all))  # expect 784,308

# --- Step 2: mark speed errors ---
df_all.loc[df_all['speed'] > speed_error_threshold, 'activity'] = 'speed error'

# --- Step 3: mark driving (speed >= threshold) ---
df_all.loc[df_all['speed'] >= speed_threshold, 'activity'] = 'driving'

# --- Step 4: add clusters back from DBSCAN (Cell 8 results) ---
df_clustered = pd.concat(all_files_clustered, ignore_index=True)
df_clustered['TID'] = df_clustered['TID'].astype(str).str.replace(r'\.0$', '',
regex=True)

# keep only cluster assignments
df_all = df_all.merge(
    df_clustered[['timestamp','TID','cluster']],
    on=['timestamp','TID'],
    how='left'
)

# --- Step 5: label traffic (noise) ---
df_all.loc[df_all['cluster'] == -1, 'activity'] = 'traffic'

# --- Step 6: merge predictions for clustered points ---
df_all = df_all.merge(
    dfc[['TID','cluster','predicted_label']],
    on=['TID','cluster'],
    how='left'
)

# assign predicted labels only for clustered points
mask_clustered = df_all['cluster'].notna() & (df_all['cluster'] != -1)
df_all.loc[mask_clustered, 'activity'] = df_all.loc[mask_clustered,
'predicted_label']

print("Final rows:", len(df_all))  # should be 784,308
print(df_all['activity'].value_counts(dropna=False).head(10))
df_all.shape

labeled_dir = os.path.join(output_dir, "lastWeek_labeled")
os.makedirs(labeled_dir, exist_ok=True)
```

```
def norm_tid(s):  # same helper you used before
    return str(s).replace('.0','')

for i, (fname, df_orig, df_clustered) in enumerate(zip(file_names,
all_files_data, all_files_clustered), 1):
    df_out = df_orig.copy()

    # 1) base labels from speed
    df_out.loc[df_out['speed'] > speed_error_threshold, 'activity'] = 'speed
error'
    df_out.loc[df_out['speed'] >= speed_threshold, 'activity'] = 'driving'

    # 2) merge cluster assignments for this file by timestamp
    df_clus = df_clustered[['timestamp','cluster']].copy()
    df_out = df_out.merge(df_clus, on='timestamp', how='left')

    # 3) traffic (DBSCAN noise)
    df_out.loc[df_out['cluster'] == -1, 'activity'] = 'traffic'

    # 4) predicted labels for clustered points
    tid = norm_tid(df_out['TID'].iloc[0])
    df_labels = dfc[dfc['TID'] == tid][['cluster','predicted_label']].copy()
    df_out = df_out.merge(df_labels, on='cluster', how='left')

    mask_clustered = df_out['cluster'].notna() & (df_out['cluster'] != -1)
    df_out.loc[mask_clustered, 'activity'] = df_out.loc[mask_clustered,
'predicted_label']

    # 5) export
    out_name = f"{os.path.splitext(fname)[0]}_labeled.csv"
    out_path = os.path.join(labeled_dir, out_name)
    df_out.to_csv(out_path, index=False)
    print(f"Saved ({i}/{len(file_names)}): {out_path} | rows: {len(df_out)}")
```

Centroid Image Extraction

```
# Input CSV file with cluster centroids
input_csv = r"C:\Users\..\cluster_centroids.csv"

# Output directory for images
output_dir = r"C:\Users\..\stop_classification_images"

# Create output directory if it doesn't exist
os.makedirs(output_dir, exist_ok=True)

# Mapbox access token
access_token = 'p..g'  # Replace with actual token

# Image parameters
zoom = 17
width, height = 300, 300
mapbox_style = 'satellite-v9'

df_centroids = pd.read_csv(input_csv)
```

```
# Cell 4: Image extraction function
def download_mapbox_image(longitude, latitude, zoom, width, height,
access_token,
                          output_path, style='satellite-v9'):
    """
    Download a satellite image from Mapbox Static Images API

    Parameters:
    -----------
    longitude : float
        Center longitude
    latitude : float
        Center latitude
    zoom : int
        Zoom level (0-22)
    width : int
        Image width in pixels
    height : int
        Image height in pixels
    access_token : str
        Mapbox access token
    output_path : str
        Full path where image will be saved
    style : str
        Mapbox style (default: 'satellite-v9')

    Returns:
    --------
    bool : True if successful, False otherwise
    """
    # Construct the URL
    url = f"https://api.mapbox.com/styles/v1/mapbox/{style}/static/{longitude},
{latitude},{zoom},0,0/{width}x{height}?access_token={access_token}"

    try:
        # Fetch the image
        response = requests.get(url, timeout=30)

        if response.status_code == 200:
            # Save the image
            with open(output_path, "wb") as f:
                f.write(response.content)
            return True
        else:
            print(f"  Failed: Status code {response.status_code}")
            return False

    except Exception as e:
        print(f"  Error: {str(e)}")
        return False

# Cell 5: Extract images for all centroids
print("\n" + "="*60)
print("STARTING IMAGE EXTRACTION")
print("="*60)

# Counters
```

```
total_centroids = len(df_centroids)
successful_downloads = 0
failed_downloads = 0
skipped_existing = 0

# Track failed downloads
failed_records = []

# Iterate through all centroids
for idx, row in df_centroids.iterrows():
    # Extract coordinates
    longitude = row['longitude']
    latitude = row['latitude']
    tid = row['TID']
    cluster_id = row['cluster']

    # Create unique filename: TID_clusterID.png
    filename = f"{tid}_cluster_{cluster_id}.png"
    output_path = os.path.join(output_dir, filename)

    # Check if file already exists
    if os.path.exists(output_path):
        print(f"[{idx+1}/{total_centroids}] {filename} - Already exists,
skipping")
        skipped_existing += 1
        continue

    # Download image
    print(f"[{idx+1}/{total_centroids}] Downloading {filename}...", end=" ")

    success = download_mapbox_image(
        longitude=longitude,
        latitude=latitude,
        zoom=zoom,
        width=width,
        height=height,
        access_token=access_token,
        output_path=output_path,
        style=mapbox_style
    )

    if success:
        print("✓ Success")
        successful_downloads += 1
    else:
        failed_downloads += 1
        failed_records.append({
            'index': idx,
            'TID': tid,
            'cluster': cluster_id,
            'longitude': longitude,
            'latitude': latitude,
            'filename': filename
        })

    # Rate limiting: wait between requests to avoid hitting API limits
```

```
    # Mapbox free tier: 100,000 requests/month
    time.sleep(0.1)  # 100ms delay between requests
```

Image Classification

```
# Define directories
image_dir = r"C:\Users\..\stop_classification_images"
model_path = r"C:\Users\..\best_model_final_V3.pth"
output_dir = r"C:\Users\..\data_processed"

# Ensure output directory exists
os.makedirs(output_dir, exist_ok=True)

# Define image transformations (adjust if your model used different
preprocessing)
transform = transforms.Compose([
    transforms.Resize(256),  # int(224 * 1.14) = 256
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

# Load the pretrained model
print("Loading pretrained model...")
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Initialize model architecture
model = models.resnet50(pretrained=False)
num_classes = 3

# Match your training architecture exactly
num_features = model.fc.in_features
model.fc = nn.Sequential(
    nn.Dropout(0.4),
    nn.Linear(num_features, num_classes)
)

# Load trained weights
model.load_state_dict(torch.load(model_path, map_location=device))
model = model.to(device)
model.eval()

# Get all image files
image_files = [f for f in os.listdir(image_dir) if f.lower().endswith(('.png',
'.jpg', '.jpeg'))]
print(f"Found {len(image_files)} images to classify\n")

# Store predictions
results = []

# Process each image
print("Classifying images...")
with torch.no_grad():
    for idx, img_file in enumerate(image_files, 1):
```

```
        try:
            # Load and preprocess image
            img_path = os.path.join(image_dir, img_file)
            image = Image.open(img_path).convert('RGB')
            image_tensor = transform(image).unsqueeze(0).to(device)

            # Get prediction
            output = model(image_tensor)
            probabilities = torch.softmax(output, dim=1)
            predicted_class = torch.argmax(probabilities, dim=1).item()
            confidence = probabilities[0][predicted_class].item()

            # Extract TID and cluster from filename (assuming format:
TID_cluster.png)
            filename_parts = os.path.splitext(img_file)[0].split('_')
            tid = filename_parts[0] if len(filename_parts) > 0 else 'unknown'
            cluster = filename_parts[1] if len(filename_parts) > 1 else
'unknown'

            # Store result
            results.append({
                'filename': img_file,
                'TID': tid,
                'cluster': cluster,
                'predicted_class': predicted_class,
                'confidence': confidence
            })

            if idx % 100 == 0:
                print(f"Processed {idx}/{len(image_files)} images...")

        except Exception as e:
            print(f"Error processing {img_file}: {str(e)}")
            continue

print(f"\nSuccessfully classified {len(results)} images")

# Create results dataframe
df_predictions = pd.DataFrame(results)

# Add readable class labels
class_labels = {0: 'loading', 1: 'rest', 2: 'traffic'}
df_predictions['predicted_label'] =
df_predictions['predicted_class'].map(class_labels)

# Export results
output_file = os.path.join(output_dir, "classification_results.csv")
df_predictions.to_csv(output_file, index=False)
```