International Journal of Applied Sciences: Current and Future Research Trends

(IJASCFRT)

ISSN: 2790-3990

https://ijascfrtjournal.isrra.org/index.php/Applied_Sciences_Journal

A Paradigm for Object Detection to Recognize and Classify Vehicles Using Computer Vision

Ville Pitkäkangas[a]*, Heikki Kaakinen[b], Tom Tuunainen[c], Olli Isohanni[d], Mitha Rachel Jose[e]

[a,b,c,d]*Centria University of Applied Sciences, Talonpojankatu 2, 67100, Kokkola, Finland*

[e]*Laurea University of Applied Sciences, Leppävaara Campus, Vanha maantie 9, 02650 Espoo, Finland*

[a]*Email: ville.pitkakangas@centria.fi,* [b]*Email: heikki.kaakinen@centria.fi,* [c]*Email: tom.tuunainen@centria.fi*

[d]*Email: olli.isohanni@centria.fi,* [e]*Email: mitha.jose@laurea.fi*

Abstract

Computer vision has emerged as a game-changing technology in the mining industry, revolutionizing operations and unlocking various use case scenarios. With increased trade facilities, ports are recognized as one of the most diligent work environments globally. The applications of machine learning and computer vision in ports offer improved security, efficient container management, intelligent traffic management, predictive maintenance, automated operations, and environmental monitoring. These advancements contribute to streamlined processes, cost reduction, enhanced safety, and overall optimization in port environments. This study proposes an approach to detect and classify vehicles in a port during the wintertime in Finland using computer vision and machine learning methods. Due to the high variability between seasons, particularly winter and summer in Finland, there might be a need to categorize images by time of year. The study is developed as a model to detect and classify vehicles in the port area, and the port used in the study acts as an international hub for various trades and industries, including but not limited to chemistry and mining.

Impact Statement

Ports play a crucial role in the international transportation of goods related to various industries. Due to their importance, harbors are large, diligent, and often complex environments featuring vehicles of multiple types. Presented here is an approach that can detect and recognize vehicles in the port area with an accuracy of over 90% utilizing computer vision and machine learning. In addition, methods for discerning the time of day and season from image streams are proposed.

Received: 2/1/2025
Accepted: 3/25/2025
Published: 4/8/2025

----

* Corresponding author.

These three processes, when combined, provide a comprehensive solution for enhanced harbor management that can revolutionize port operations. Using artificial intelligence in ports improves efficiency and safety, reduces costs, optimizes logistics and resource allocation, and facilitates sustainable economic growth. Furthermore, the vehicle detection and classification part of the proposed approach is designed in ways that can require relatively little computational power, storage space, and training time, thus leading to greener technology.

Keywords: Artificial Intelligence; Computer Vision; Edge Detection; Machine Learning.

1. Introduction

Ports play a crucial role in transportation hubs for various industries, and the integration of artificial intelligence (AI), specifically computer vision and machine learning, enables monitoring both the quantity and quality of goods stored and transported within the harbor vicinity. Utilizing image streams from port-installed webcams, AI employs many specialized algorithms, methods, and sub-routines to gather, process, and analyze diverse information types. The applications of machine learning and computer vision in ports bring significant benefits and improvements to various operations and processes.

The use of AI provides enhanced security, efficient container management, intelligent traffic management, predictive maintenance, automated operations, and environmental monitoring. Machine learning and computer vision enable advanced port surveillance systems, including object detection, anomaly detection, and facial recognition [1]. These technologies help detect potential security threats, monitor access points, and enhance safety and security. Machine learning algorithms can optimize container placement and tracking, leading to improved efficiency in port operations. Computer vision techniques can accurately identify and track containers, facilitating automated inventory management and reducing errors and delays. By analyzing real-time data from cameras and sensors, machine learning can optimize traffic flow within the port area. It can predict and mitigate congestion, identify bottlenecks, and suggest alternative routes, thereby improving overall logistics and reducing transit times. Machine learning algorithms can analyze sensor data from port equipment and predict maintenance needs. By detecting potential equipment failures in advance, maintenance activities can be scheduled proactively, thus minimizing downtime and optimizing equipment utilization [2].

Computer vision enables the automation of various port operations, such as automated guided vehicles (AGVs) for cargo handling, robotic crane systems, and autonomous drones for inspections. These technologies increase operational efficiency, reduce human error, and improve productivity. Computer vision combined with machine learning can assist in monitoring environmental aspects within ports, such as air quality, water pollution, and noise levels. These data can help identify and address environmental concerns, ensuring regulation compliance and promoting sustainable practices [3].

Increased container traffic in ports requires adjustments to port terminals and better connections. Ever larger ships have increased the pressure on ports and the cities they serve. In addition, growing environmental concerns have driven the sector to adapt to greener regulations. The global maritime industry is highly competitive internationally. AI can analyze the information to provide insights for better decision-making, improvement of safety and energy efficiency, and optimization of logistics.

Despite advancements in AI and computer vision technologies, effective monitoring and management of port operations – particularly vehicle detection and classification – remains challenging. Known current models do not account for significant seasonal variations, fluctuating lighting conditions, and highly varying vehicle types, leading to reduced accuracy and reliability of object detection and classification. To address this gap, we propose a novel computer vision and machine learning system that integrates time-of-day and seasonal detection as part of the vehicle classification process. The proposed model is adaptable to the high variability between winter and summer environments, offering a comprehensive solution for enhanced port management, potentially revolutionizing how ports handle traffic management and operations in adverse conditions.

The study was done for the Port of Kokkola, one of the leading ports of Finland and the only port in the Nordics to offer an all-weather terminal where goods can be loaded and unloaded in protection from wind and weather [4]. The Port of Kokkola serves the mining industry. It also acts as a link between different countries. Furthermore, the local industry, especially the chemical industry cluster in the nearby industrial park, and trade in Kokkola form significant customer groups. The Port of Kokkola consists of three ports: Hopeakivi (Silverstone Port), Kantasatama (General Port), and Syväsatama (Deep Port). Figure 1 illustrates the port locations.

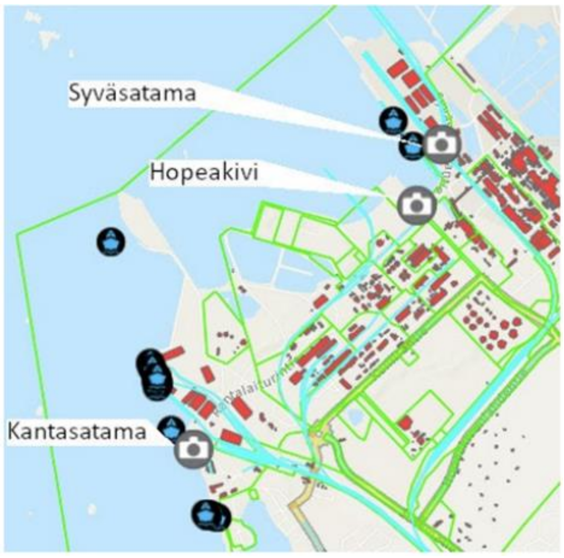


Figure 1: Three ports of the Port of Kokkola. The shortest distance between Kantasatama and Hopeakivi is approximately 1800 m and between Hopeakivi and Syväsatama approximately 200 m

The novelty of the study can be summed up with a few points:

- integration of time-of-day and seasonal detection as part of the vehicle classification process,
- lower requirements for processing power, training time, and storage space in comparison to more conventional models, such as neural networks, and
- compiling a timeline of the vehicle classification results for a more detailed analysis of the port activity and

operations.

The research question is: How can a computer vision system be adapted to reliably detect and classify vehicles in a port environment, particularly under varying seasonal and lighting conditions, to improve the efficiency of port operations?

2. Dataset

The data for creating, training, and testing AI consist of two image sets taken by the web cameras placed in the different locations of the port and saved on web servers. Each image has a 24-bit color and dimensions of 1280 × 720 pixels.

2.1. Image Set 1

The first set of images was captured between 4 January 2022, 9:48 p.m. and 10 January 2022, 12:53 p.m. The average sampling resolution was around 10 minutes, making the dataset consist of 2423 images or about 807–808 images per port. Figure 2 shows a simple data acquisition software developed for downloading the first set of images for training the AI. The program uses Selenium for automated tasks, such as logging in and downloading images [5].

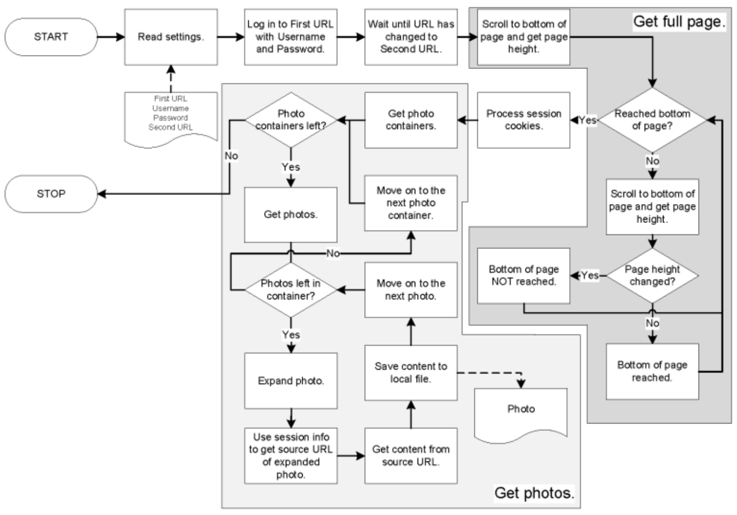

START
Read settings.
Log in to First URL with Username and Password.
Wait until URL has changed to Second URL.
Scroll to bottom of page and get page height.
Get full page.
First URL Username Password Second URL
Photo containers left?
Get photo containers.
Process session cookies.
Yes
Reached bottom of page?
No
STOP
No
Yes
Get photos.
Move on to the next photo container.
Scroll to bottom of page and get page height.
No
Photos left in container?
Move on to the next photo.
Bottom of page NOT reached.
Yes
Page height changed?
Yes
No
Expand photo.
Save content to local file.
Bottom of page reached.
Photo
Use session info to get source URL of expanded photo.
Get content from source URL.
Get photos.

Figure 2: Structure of the tool program that collected the first images

2.2. Image Set 2

The second set of images was captured in two periods. The first was between 16 June 2022, 7:33 a.m. and 17 June 2022, 1:57 p.m., and the second was between 20 June 2022, 6:55 a.m. and 30 June 2022, 11:27 a.m. The average sampling resolution was about 10 minutes. Data for the Syväsatama port were unavailable in June 2022, so the image set consists of 1888 photographs (973 for Hopeakivi and 915 for Kantasatama). A program was developed to obtain this second set of images. Figure 3 shows the structure of this tool. The sources are Uniform

Resource Locator (URL) addresses pointing to the most recent photographs of each port, and the "last modified" time is parsed from the response to the Hypertext Transfer Protocol (HTTP) request used for getting the image.

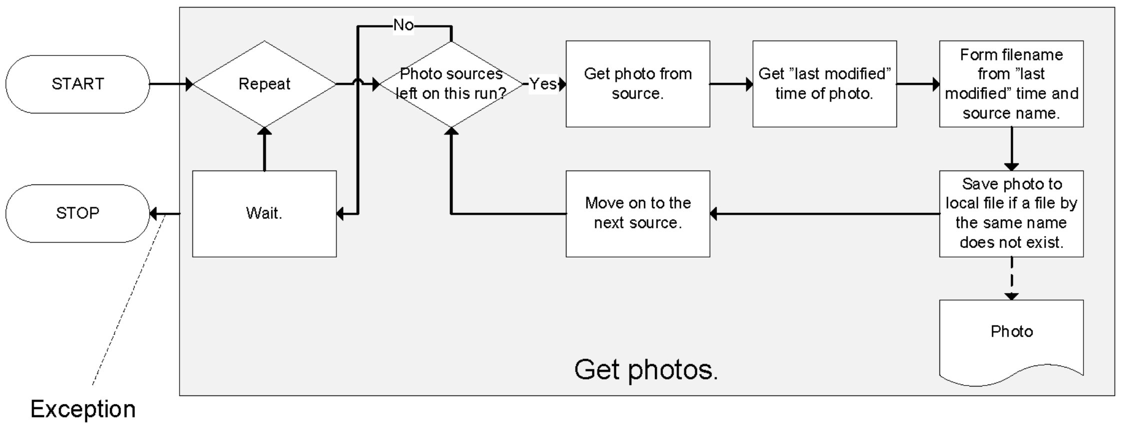

START
STOP
Repeat
No
Photo sources left on this run?
Yes
Get photo from source.
Get "last modified" time of photo.
Form filename from "last modified" time and source name.
Wait.
Move on to the next source.
Save photo to local file if a file by the same name does not exist.
Photo
Get photos.
Exception

Figure 3: Structure of the tool that collected the second set of images

3. Methods

To assess the feasibility of implementing a computer vision application to identify and classify vehicles in the port area, the following methods were used to evaluate the application of computer vision in a port system and the performance impact of the results on image set size and distribution. Applying time-of-day detection, season detection, and vehicle detection using computer vision in transport systems provides a comprehensive solution for dynamic scheduling, context-aware decision-making, and efficient resource allocation. This results in optimized traffic management, improved road safety, and enhanced efficiency.

3.1. Detection of time-of-day

The first algorithm developed was the differentiation between day and night — or, more precisely, between light and dark. The ability to discern times of day might enable the AI to choose optimal analysis methods for each lighting-dependent situation. The most crucial libraries in this implementation are OpenCV [6] and NumPy [7]. The algorithm detects and separates the sky from the image and compares the sky color information with threshold values, as shown in Figure 4.

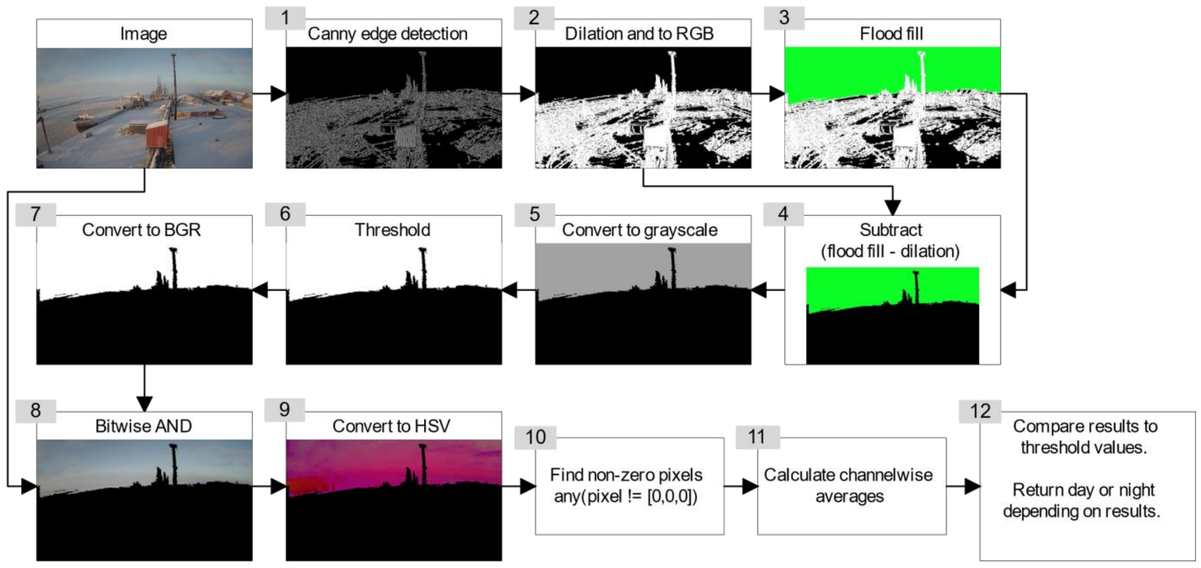


Figure 4: Day and night classification algorithm with example

The first step is to perform edge detection. This is based on the observations that there are typically few edges in the sky, and everything below the horizon often consists of many boundaries. The Canny edge-detection method [8] is used. The threshold parameters for the first and second hysteresis procedures are 16 and 64, respectively. Both hysteresis parameters were experimentally proved. The second step is to fill the gaps in the horizon. This is realized using morphological dilation with a structuring element of ones and the size of 3 × 1 pixel, or three columns and one row. The element shape — more columns than rows — is based on the typical horizon outline. The dilation is done three times, and the result is converted from grayscale to the red-green-blue (RGB) color space. The third step is flood-filling the dilated color image. Three seed point positions are chosen: the leftmost, the middle, and the rightmost pixel from the top row. It is critical for the fill color not to be the same as the background or the dilated edges so the sky can be extracted from the image. In this case, the experimentally chosen RGB values of 36, 255, and 12 were used. The fourth step is subtracting the dilated RGB image from the flood filled. This yields a picture with all edge information removed and only two components left: the sky and everything else. In the fifth step, the result is converted to grayscale.

A simple binary method is used in the sixth step for image thresholding, with 0 and 255 as parameters. In the seventh step, the result is converted to blue-green-red (BGR) color space. This way, the processed image (the mask) can be compared with the original. The eighth step is to perform a bitwise AND operation on the original and the processed images. The result is colorless, except for the sky that, in other words, is extracted from the image with the binary mask. The ninth step is to convert the sky image to hue-saturation-value (HSV) color space, simplifying brightness comparison with threshold levels. The tenth step is to select all HSV pixels with a non-zero color on at least one channel. In the eleventh step, a channel-wise average for Hue, Saturation, and Value is calculated from each pixel selected in step ten. The twelfth and final step is to compare the averages with threshold levels. A maximum of 99 was chosen for Saturation, and a minimum of 121 was picked for Value. Both points were selected experimentally. If these conditions are satisfied, the AI assumes the image was taken during the day. Otherwise, the picture was probably taken at night.

This comparison can be expressed as

$$t = \begin{cases} \text{"day"}, & \text{if } (\bar{v} > 120 \land \bar{s} < 100), \\ \text{"night"}, & \text{otherwise.} \end{cases} \qquad (1)$$

In (1), $\bar{v}$ and $\bar{s}$ denote the averages of the Value and Saturation channels of the selected pixels, and t is the time of day when the image was likely captured.

3.2. Season Detection

There is a need to classify images by the time of year due to the high variability between seasons, especially winter and summer, in Finland. Subsequently, this information can be used as a basis for selecting the ideal methods for further analysis of images – or choosing vehicle detection models in the case of this study (Figure 5). An image classification model for season detection was created with Lobe [9]. Lobe offers a powerful computer vision solution for image classification that is applicable for season detection, enabling accurate identification of seasons from images, which can be leveraged for various applications such as environmental monitoring, agricultural planning, and content creation. This model was trained on a dataset where the first and second image sets formed the winter and summer categories, respectively. The model was exported to TensorFlow [10].

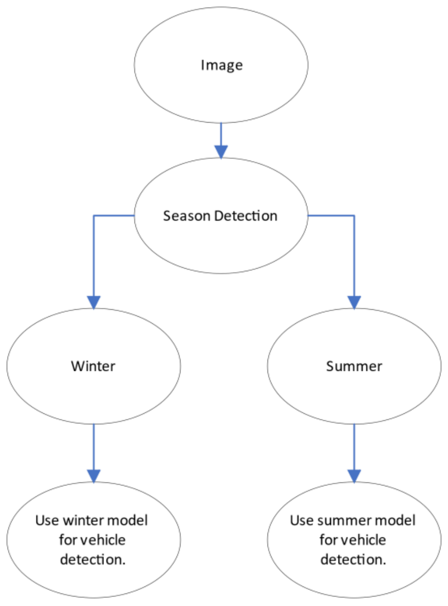


Image
Season Detection
Winter
Summer
Use winter model for vehicle detection.
Use summer model for vehicle detection.

Figure 5: Choosing the vehicle classification model based on season

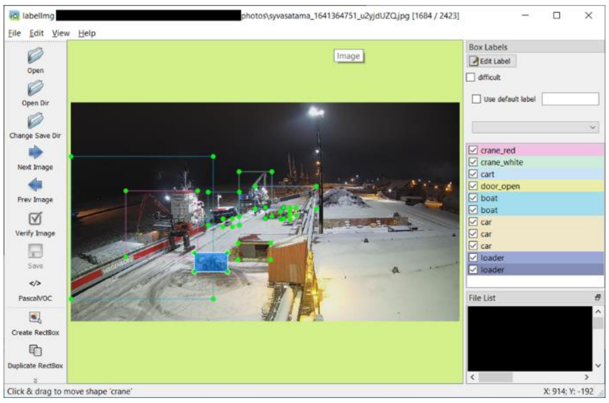
3.3. Vehicle Detection and Classification

Tracking the various types of vehicles in the port area is a vital responsibility all ports share. The photographs taken in the winter indicate the presence of cars, trucks, forklifts, and others. Vehicle category breakdown by port is listed in Table 1.

Table 1: Vehicle categories in the three ports

Category of Vehicle	***Hopeakivi***	***Kantasatama***	***Syväsatama***
Car	Yes	Yes	Yes
Truck	Yes	No	Yes
Forklift	No	No	Yes
Loader	No	No	Yes
Snowplow	No	No	Yes
Digger	No	No	Yes
Forklift/Snowplow	Yes	Yes	Yes

Before creating the model to recognize vehicles, each image from the dataset was scaled to 320 × 180, or 1/16th of its original size, to save the time needed for analysis. The resized images were processed using LabelImg software [11], an open-source graphical image annotation tool widely utilized in computer vision and machine learning projects for object detection and image classification tasks. This tool allows users to annotate objects within images by drawing bounding boxes around them and assigning labels. In LabelImg, users open a picture, define bounding boxes around the objects of interest, and set corresponding labels to them (a car, a truck, a forklift, etc.), ultimately generating annotated data for training machine learning models. An example is shown in Figure 6.


labelImg [redacted]photos\syvasatama_1641364751_u2yjdUZQ.jpg [1684 / 2423]

Click & drag to move shape 'crane'

X: 914; Y: -192

Figure 6: Example showing the use of LabelImg software

The study initially utilized pre-trained convolutional neural networks, such as YOLOv3 [12] and ResNet-50 models [13]. Model training, testing, and validation were done using ImageAI software [14,15]. When the vehicle detection algorithm in this study was under development, ImageAI was based on TensorFlow [10,14]. Since then, the backend technologies have been changed to PyTorch [15]. Some tests were done with another YOLOv3 implementation, based on the work of [16,17] using TensorFlow [10]. There were no significant differences between the results from YOLOv3 [12] and ResNet-50 [13], so the work with the latter was stopped after early tests. The ImageAI version was retained because it was easier to train the models with it. However, as seen in the next section, the results from the neural networks pushed for a thorough redesign of the vehicle detection and categorization model.

OpenCV [6] was used to visualize all the results.

4. Results and Discussions

The time-of-day classifier categorized the image set with 98–99% accuracy. Misclassified images often have unusual brightness or darkness for the time of day, or their visibility is low due to weather conditions.

As Lobe reported, the season classification model has an estimated 93% accuracy for a dataset comprising all study images.

The test parameters and results of the YOLOv3-based network are listed in Table 2.

Table 2: Parameters and results (separated by a dash) of the neural network trained for vehicle detection

Metric/Parameter	Value
Batch size	2
Number of experiments	7
Evaluation samples	262
Intersection over Union (IoU)	0.5
Object threshold	0.3
Non-Maximum Suppression (NMS)	0.5
Car	0.7815
Forklift/Snowplow	0.75
Digger, Forklift, Loader, Snowplow, Truck	0
Mean Average Precision (mAP)	0.2188

To address the unsatisfactory results obtained from the neural networks, a different strategy was devised using decision trees implemented via the Scikit-learn library [18]. This method involves converting the scaled images from BGR to grayscale and using two phases (passes) to divide the result into non-overlapping blocks (tiles) of equal size, as shown in Figure 7. The solid lines represent the first phase, and the dotted lines indicate the second phase. The processes for training, testing, and using this new model are reported in Figure 8. The block size in Figure 9 that demonstrates the algorithm is 16 × 16 pixels.

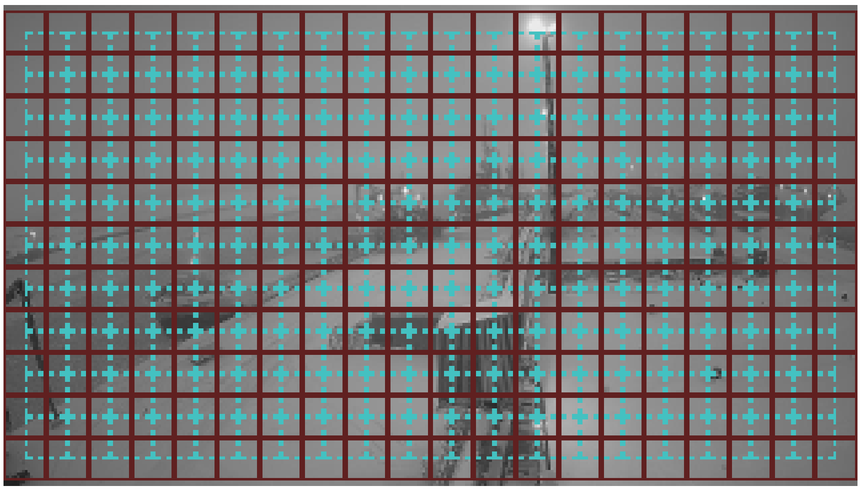


Figure 7: Example of splitting an image into blocks in two phases

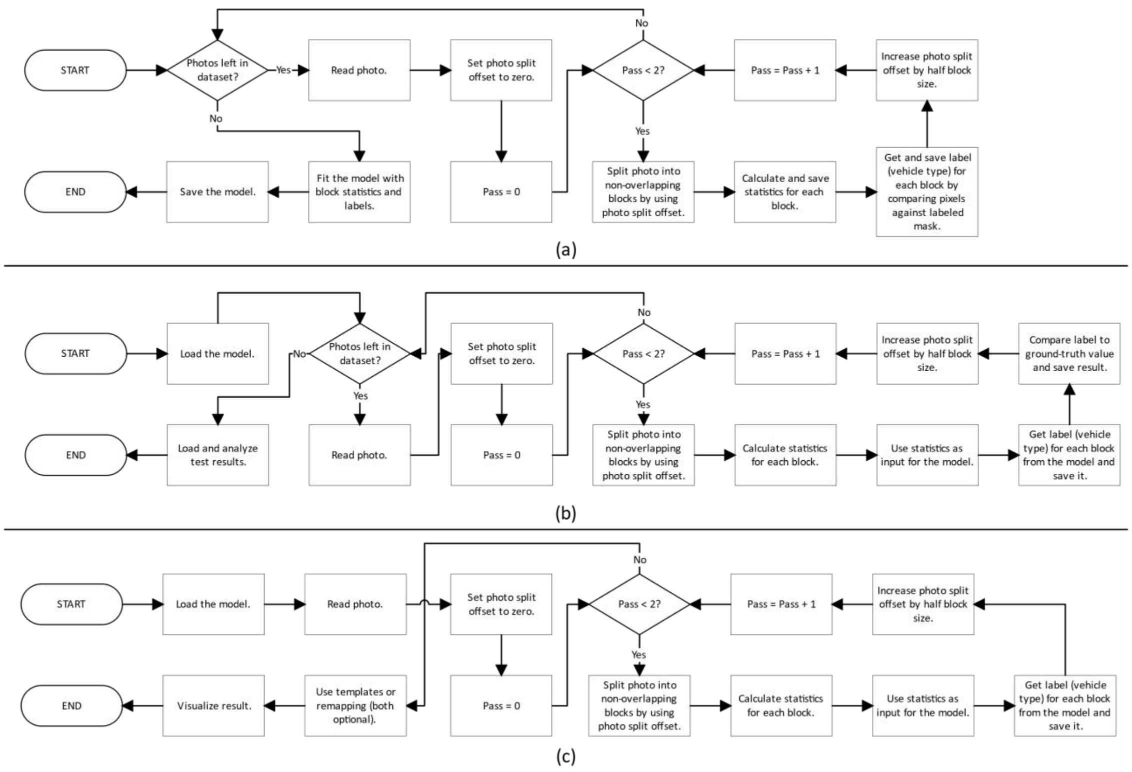

START
Photos left in dataset?
Yes
No
Read photo.
Set photo split offset to zero.
Pass < 2?
No
Yes
Pass = Pass + 1
Increase photo split offset by half block size.
END
Save the model.
Fit the model with block statistics and labels.
Pass = 0
Split photo into non-overlapping blocks by using photo split offset.
Calculate and save statistics for each block.
Get and save label (vehicle type) for each block by comparing pixels against labeled mask.
(a)
START
Load the model.
No
Photos left in dataset?
Yes
Set photo split offset to zero.
Pass < 2?
No
Yes
Pass = Pass + 1
Increase photo split offset by half block size.
Compare label to ground-truth value and save result.
END
Load and analyze test results.
Read photo.
Pass = 0
Split photo into non-overlapping blocks by using photo split offset.
Calculate statistics for each block.
Use statistics as input for the model.
Get label (vehicle type) for each block from the model and save it.
(b)
START
Load the model.
Read photo.
Set photo split offset to zero.
Pass < 2?
No
Yes
Pass = Pass + 1
Increase photo split offset by half block size.
END
Visualize result.
Use templates or remapping (both optional).
Pass = 0
Split photo into non-overlapping blocks by using photo split offset.
Calculate statistics for each block.
Use statistics as input for the model.
Get label (vehicle type) for each block from the model and save it.
(c)

Figure 8: Flowchart diagrams of developing and applying the model. (a) training, (b) testing, (c) using as part of an application

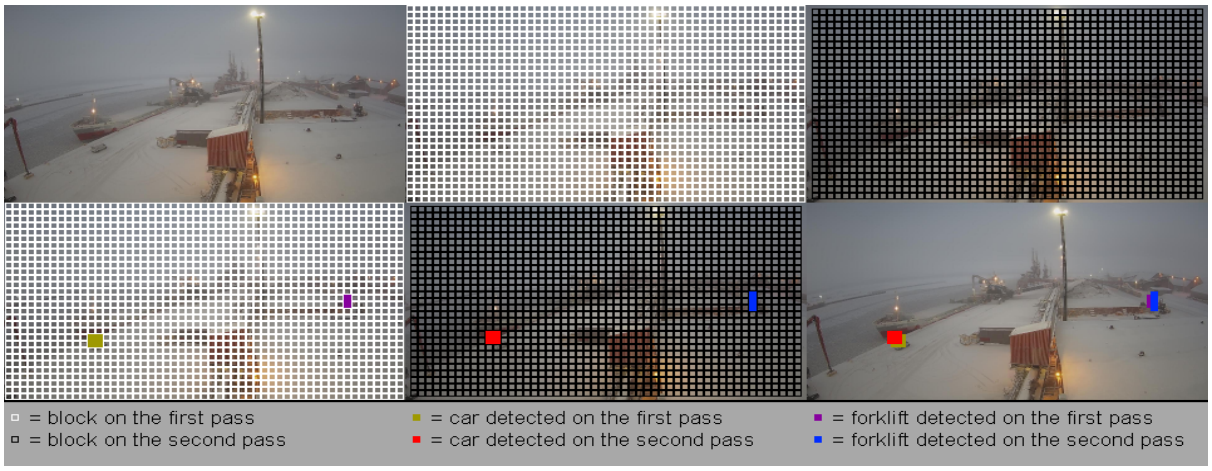

□ = block on the first pass
□ = block on the second pass
■ = car detected on the first pass
■ = car detected on the second pass
■ = forklift detected on the first pass
■ = forklift detected on the second pass

Figure 9: Creating and using data for decision trees to find and classify vehicles

Each tile is flattened to a 1D array with NumPy [7], and the statistics in Table 3 are then computed for it using the NumPy [7] and SciPy [19] libraries. For better accuracy, the block-wise process is repeated with a half-tile offset, resulting in a semi-overlapping-block-based method, as seen in Figure 7. Images with extents not divisible by the block size are centrally cropped.

Table 3: Block-wise statistics for the method

Feature	Definition
Average	The value returned by numpy.mean()
Standard deviation	The value returned by numpy.std()
Mean-to-variance ratio	Average divided by the square of the standard deviation if the standard deviation is not zero; otherwise, zero is used
Skewness	The value returned by scipy.stats.skew()
Kurtosis	The value returned by scipy.stats.kurtosis()
Entropy	The value returned by scipy.stats.entropy()
Range	The difference between the maximum and the minimum pixel value in the block
Smoothness	1-1/(1+(σ^2)), where σ is the standard deviation
Time of day	"day" / "night" or 1 / 0 as returned by the time-of-day detection method when given the complete BGR photograph as input

Labels generated using the LabelImg tool [11] work as references for vehicle positions and categories. These labels are documented in Table 1, although the "Forklift/Snowplow" category was used only with the neural networks. A masked image is created from the labels and bounding boxes, with each object represented by a solid rectangular polygon filled in the color based on the category label's ID. The ID value is obtained by adding two to the label index, with ID values zero and one reserved for the background and unidentifiable vehicles during the analysis phase. Each block is compared with the corresponding region in the masked image; if enough non-zero mask pixels exist within the tile, it is classified as belonging to a vehicle type indicated by the corresponding bounding box label ID. A block size of 6 × 6 pixels and a threshold value of 0.8 were used to determine whether a block encompasses a vehicle. This threshold implies that at least 80% of the mask pixels within the tile should be non-zero for the block to be classified as part of a vehicle. These parameters were chosen experimentally. A decision tree model is fitted with the block data and is saved for future use with the dump and load functions provided by [20]. The generation and application of data are illustrated in Figure 9. In Figure 9, 1) top left: scaled image; 2) top center: first-pass block division; 3) top right: second-pass block division; 4) bottom left: vehicles, such as a car and a forklift, were detected on the first pass; 5) bottom center: results from the second pass; and 6) bottom right: combined first- and second-pass results. The vehicle detection and classification method is used as follows: the image is first split into semi-overlapping blocks in the training phase that is shown in Figure 9 and Figure 10, and for each of these blocks, statistics in Table 3 are calculated and fed to the decision tree model that will return per-block labels. (See also Figure 8.) The decision trees were generated using a 70:30 train-test split on the dataset. The accuracy of comparing the test results with the actual test labels is 99% – a tremendous improvement over the 22% obtained from the neural network (Table 2). The confusion matrix is reported in Table 4, with correct predictions emphasized in gray. Precision, recall, and F1 scores from the confusion matrix and k-fold cross-validation (k=5) are listed in Table 5. These results demonstrate that the proposed model performs robustly under varying circumstances, with weighted F1-scores indicating high reliability across daytime and nighttime conditions.

Table 4: Test results of the decision trees as a confusion matrix

Total number of blocks (in test data): 2251937							
Category	**Non-vehicle block**	**Car**	**Truck**	**Forklift**	**Loader**	**Snowplow**	**Digger**
Non-vehicle block	2199000	2115	1	0	0	0	0
Car	10489	39510	0	0	0	0	0
Truck	136	0	265	0	0	0	0
Forklift	124	16	0	220	0	0	0
Loader	17	0	0	0	35	0	0
Snowplow	2	0	0	0	0	1	0
Digger	3	0	0	0	0	0	3

Table 5: Precision, recall, and F1 scores

Precision, Recall, and F1 scores from the Confusion Matrix			
	Precision	**Recall**	**F1**
Non-vehicle block	0.9951	0.9990	0.9971
Car	0.9488	0.7902	0.8623
Truck	0.9962	0.6608	0.7946
Forklift	1.0000	0.6111	0.7586
Loader	1.0000	0.6731	0.8046
Snowplow	1.0000	0.3333	0.5000
Digger	1.0000	0.5000	0.6667
K-fold Cross-validation Scores			
	Precision (weighted)	**Recall (weighted)**	**F1 (weighted)**
K-fold 1	0.9958	0.9959	0.9958
K-fold 2	0.9959	0.9960	0.9959
K-fold 3	0.9959	0.9960	0.9958
K-fold 4	0.9958	0.9959	0.9958
K-fold 5	0.9959	0.9960	0.9959
Average across 5 folds	0.9959	0.9960	0.9958

The result can be further refined by post-processing.

One way to improve the model accuracy is to use template matching, particularly near the image edges, where block coverage may be inadequate if the image size is not divisible by tile dimensions. The templates and the images they are matched on can be pre-processed with thresholding or edge detection for more generalizable results.

Another accuracy improvement method involves filtering out the detected objects whose size, position, and pixel count are irrational. For instance, in the dataset utilized for this study, cars typically are not exactly one tile wide and high. Furthermore, vehicles of any kind are considered only if their topmost y-coordinate falls below a quarter of the image's height. This is because the sky typically occupies the uppermost quadrant of the picture, and aircraft are not analyzed in this study. The third criterion is that at least one non-zero pixel needs to be in the area matching the vehicle position in the threshold image.

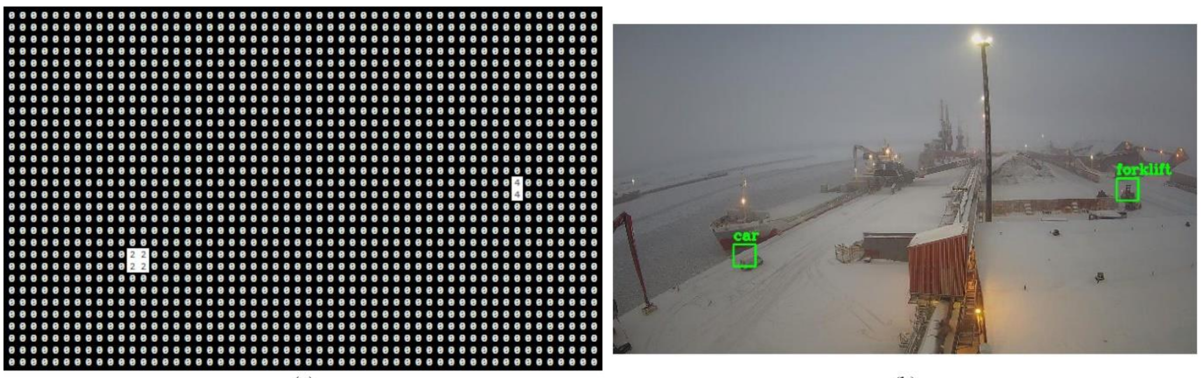
The thresholding method used here was developed by [21] and implemented in the Scikit-image library [22]. The window size for the thresholding was 5 × 5 pixels, chosen experimentally.

Yet another way to improve accuracy is bookkeeping and remapping. In the training phase, vehicle classes found in LabelImg-generated files can be automatically saved for each geo-location (port name). During the analysis, incorrect labels given by AI can be corrected with a remapping file. For example, diggers appear only in Syväsatama in the data used in this study. The AI model sometimes misclassified trucks and cars in Hopeakivi and Kantasatama as diggers. Hence, a remapping was created to relabel “diggers” as trucks in Hopeakivi and cars in Kantasatama.

Even with post-processing, the study has several limitations. The random-forest-based model, while efficient, may not generalize to other environments outside port settings, such as airports or logistics hubs, where vehicle types and environmental conditions differ. Additionally, the model performance on underrepresented vehicle types was limited, likely due to data imbalance, suggesting that additional data collection or class-specific tuning may be necessary. Finally, while computationally efficient, the model could face scalability challenges in high-traffic or larger environments, where implementations utilizing a graphical processing unit (GPU) may be required to achieve real-time performance.

Future work to improve the performance of the model on underrepresented vehicle types could include collecting additional data, implementing oversampling, exploring data augmentation methods, or using a cost-sensitive learning approach. Porting the model to GPU might efficiently rectify scalability challenges.

An example outcome of vehicle detection and classification is presented in Figure 10, where two vehicles were found: a car and a forklift. Figure 10 (a) shows the labeled regions as an array where 2 = car and 4 = forklift. Zeros represent the background tiles where no vehicles were found. In Figure 10 (b), a visualization of the result is applied to the original unscaled image.



(a) (b)

Figure 10: (a) Result of vehicle detection and classification. (b) Visualization of detection and classification results superimposed on the original photograph

The source image for the analysis in Figure 10 is shown on the top left of Figure 9. The AI model inaccurately found a small car between the two actual vehicles in an unprocessed result. This extra car was a piece of a wall in a building. The error was corrected with filtering based on object properties.

A random forest is used instead of a single decision tree. This reduces overfitting and false positives in images the model has not learned but may grow the model size and processing time as seen in Table 6. The table also demonstrates the advantages of the random-forest-based model over neural networks: increased training speed and accuracy and decreased need for temporary storage space. In addition, the central processing unit (CPU) speed of the optimized model is almost comparable to the graphical processing unit (GPU) speed of the more conventional approaches. Time-of-day classification and seasonal detection are excluded from the numbers in Table 6.

The results in Table 6 seem to reflect the assumption that previous studies on vehicle detection, such as those utilizing YOLOv3 and ResNet-50, have focused primarily on achieving high accuracy in general-purpose settings without specific adaptability to seasonal or lighting variations. In contrast, the proposed model incorporates a hybrid approach, combining CNN-based season detection with a random-forest-based classifier to enhance detection under changing environmental conditions. This design addresses limitations identified in prior studies regarding adaptability and computational efficiency, making it a suitable choice for applications where real-time performance and adaptability are crucial.

Table 6: Comparison of vehicle classification methods

	Neural Network (ResNet-50)[a]	**Neural Network (YOLOv3)**	**Decision Tree**[b]	**Random Forest (20 trees)**
Training and testing time	>6 hours[a]	40 hours	4 hours[b]	5 hours
Disk space	>700 MB[a]	4.83 GB (final model 235 MB)	25 MB[b]	270 MB
Reported precision	20%[a]	21.88%	99%[b, c]	99%[c]
Processing time per photo	1…2 seconds on the GPU[a]	<2 seconds on the GPU	2…3 seconds on the CPU[b,d]	6 seconds (unoptimized), 3 seconds (optimized) on the CPU[d]

[a]estimate, based on initial tests

[b]A single tree is impractical due to overfitting.

[c]post-processing excluded from precision values

[d]post-processing included in times

The analysis results can be converted to a time series where the presence of vehicles can be monitored over time; for instance, peak traffic hours of the port can be found. An example is given in Figure 11. The lower heatmap represents the calculations of how many percent of each photograph is occupied by vehicles; for example, a single car with a bounding box of 250 × 150 pixels would take up 2.44% of an image with dimensions of 1280 × 720 pixels. It is displayed in Figure 11 that during the observational period, the traffic in Kantasatama and Syväsatama peaked on 5 January 2022. The white lines in Figure 11 show timespans when photographic data were unavailable.

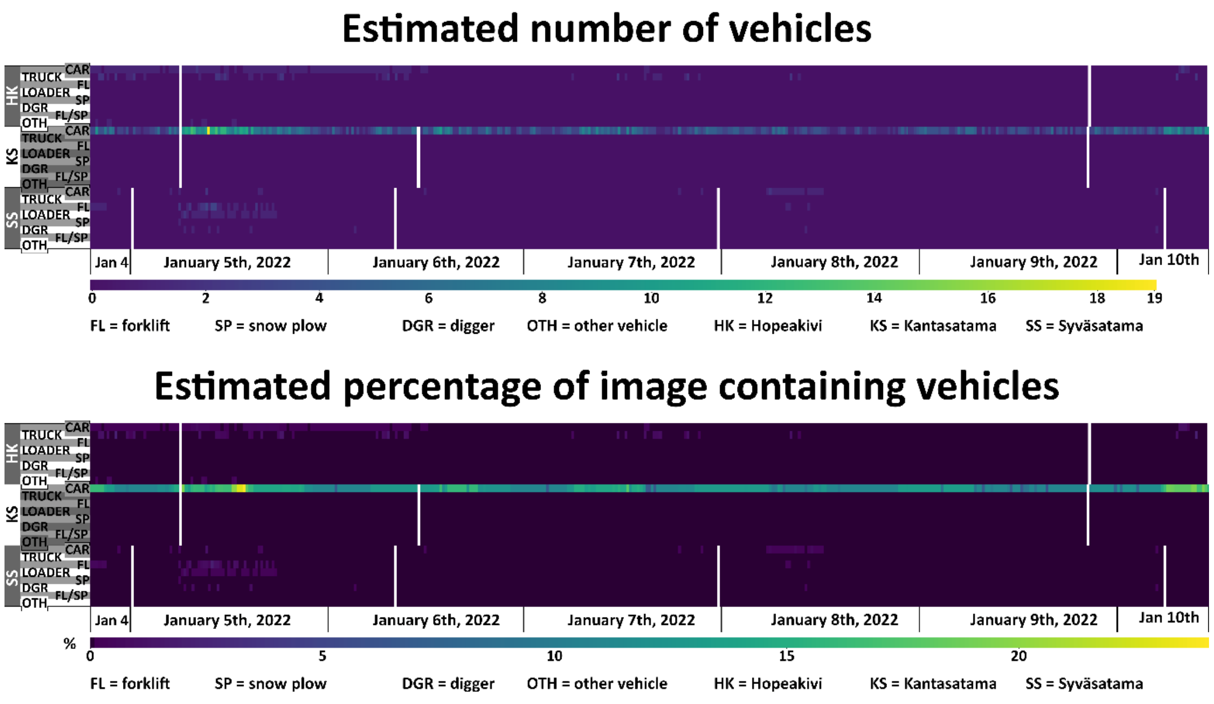


Figure 11: Time series from the analysis results for the winter image set

5. Conclusion

Computer vision-based vehicle detection and classification systems have numerous advantages. Modeled for detecting and classifying vehicles in the port area, the study demonstrates the potential of computer vision technology in revolutionizing port operations.

The proposed computer vision method successfully adapts to the challenges of varying seasonal and lighting conditions in port environments. With highly accurate time-of-day and season detection components, the system reliably detects and classifies vehicles even in adverse conditions such as nighttime and snow. This adaptability significantly improves the efficiency of port operations by ensuring accurate, near-real-time vehicle classification under diverse environmental conditions, making the model a valuable tool for modern port management systems.

The model demonstrates strong performance, as indicated by the high precision, recall, and F1 scores across most vehicle classes. The 5-fold cross-validation results confirm the stability of the model, with consistent weighted F1 scores near 0.996, indicating robust generalization. However, performance for certain underrepresented vehicle types, such as snowplows and diggers, shows room for improvement. These results suggest that further tuning and additional data collection could enhance the ability of the model to handle rare vehicle classes while maintaining total accuracy. Additionally, tuning parameters such as the number of trees, depth, or sample sizes could make the model more suitable for real-time applications. Speed improvements could also be achieved by porting the model to a GPU.

While the random-forest-based approach offers significant advantages in terms of reduced processing power, shorter training time, and lower storage requirements than CNNs, it may generalize less effectively across diverse datasets or unseen scenarios. CNNs, with their ability to automatically learn hierarchical features, tend to excel in environments with high data complexity and variability, such as object detection tasks requiring subtle feature distinctions. In contrast, the random forest relies on hand-crafted features, which may make it less adaptable to fine-grained variations without extensive feature engineering.

However, the efficiency and interpretability of the random forest mitigates this trade-off, making it a suitable solution for applications where real-time performance is needed but computational resources are limited. Additionally, specialized models for specific circumstances could further mitigate this trade-off. For example, an AI technique could dynamically select the most appropriate model, as seen with the CNN-based season detection used in this study.

Future work could expand on the hybrid approaches already explored in this study, such as integrating CNN-based season detection with random-forest-based vehicle detection and classification. Further research such as automated tuning of hyperparameters or incorporating dimensionality reduction could optimize the use of hybrid models to balance better generalization capabilities with practical deployment requirements. Additionally, the results of this study demonstrate the value of converting detection results into a time-series format, which offers long-term operational benefits by providing insights into how the model performs over time and supporting resource allocation and predictive maintenance in port management.

In conclusion, the proposed system successfully addresses the primary research question by providing a computer vision solution that adapts to seasonal and lighting variations, ensuring reliable vehicle detection in diverse environmental contexts. This adaptability is crucial for modern port management, where operational efficiency and safety depend on accurate, consistent vehicle detection. Beyond ports, the developed techniques can be applied to industries with similar environmental challenges, such as airports, logistics hubs, and mining operations, where real-time monitoring of vehicles in dynamic conditions is essential.

By improving efficiency, safety, and automation, the model opens possibilities for optimizing logistics, enhancing renewable energy operations, and facilitating sustainable economic growth. Continued research and development will further refine and expand the capabilities of computer vision systems in the port industry.

Acknowledgements

We would like to express our heartfelt gratitude to the European Regional Development Fund (ERDF), Leverage from the EU 2014–2020, Central Ostrobothnia Cooperative Region, and local companies for their generous financial support, without which the Biline2 project would not have been possible. The commitment to promoting innovation and regional development has played a pivotal role in the success of this endeavor. We would also like to express our sincere thanks to Centria University of Applied Sciences, for their unwavering collaboration, expertise, and resources. A special thanks to the senior lecturer from Laurea University of Applied Sciences for the invaluable guidance and support in shaping the article. All these involvements have greatly enriched the project's outcomes and broadened its impact. The project is a testament to what can be achieved through collaboration, innovation, and the support of visionary institutions. Thank you to all our partners for believing in our vision and making it a reality.

References

[1] A. Naumann, F. Hertlein, L. Dörr, S. Thoma, and K. Furmans, “Literature review: Computer vision applications in transportation logistics and warehousing,” 2023, DOI: 10.48550/arXiv.2304.06009.

[2] F. Hake, P. Lippmann, H. Alkhatib, V. Oettel, and I. Neumann, “Automated damage detection for port structures using machine learning algorithms in heightfields,” *Appl Geom*, vol. 15, no. 2, pp. 349–357, 2023, 10.1007/s12518-023-00493-z.

[3] C. V. Ribeiro, A. Paes, and D. de Oliveira, “AIS-based maritime anomaly traffic detection: A review,” *Expert Syst Appl*, vol. 231, 2023, 10.1016/j.eswa.2023.120561.

[4] Oy M. Rauanheimo Ab, “The port of Kokkola”, 2023, [Online]. Available: https://www.rauanheimo.com/en/the-port-of-kokkola/

[5] *Selenium*. (2023). Software Freedom Conservancy. [Online]. Available: https://www.selenium.dev/

[6] G. Bradski, “The OpenCV library,” *Dr Dobb's Journal of Software Tools*, vol. 25, no. 11, pp. 120–125, 2000.

[7] C. R. Harris, J. K. Millman, S. J. van der Walt, R. Gomers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. Del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, ”Array programming with NumPy,” *Nature*, vol. 585, no. 7825, pp. 357–362, 2020, 10.1038/s41586-020-2649-2.

[8] J. F. Canny, “A computational approach to edge detection,” *IEEE Trans Pattern Anal Mach Intell*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986, 10.1109/TPAMI.1986.4767851.

[9] *Lobe*. (2021). Microsoft. [Online]. Available: https://www.lobe.ai/

[10] D. A. Konovalov, S. Hillcoat, G. Williams, R. Alastair Birtles, N. Gardiner, and M. I. Curnock, “Individual minke whale recognition using deep learning convolutional neural networks,” *J Geosci Environ Prot*, vol. 6, pp. 25–36. 2018, 10.4236/gep.2018.65003.

[11] *LabelImg*. (2015). Tzutalin. [Online]. Available: https://github.com/tzutalin/labelImg

[12] J. Redmon, and A. Farhadi, “YOLOv3: An incremental improvement,” 2018, DOI: 10.48550/arXiv.1804.02767.

[13] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015, DOI: 10.48550/arXiv.1512.03385.

[14] *ImageAI*. (2018). M. Olafenwa. [Online]. Available: https://github.com/OlafenwaMoses/ImageAI

[15] M. Olafenwa, “Official English documentation for ImageAI! — ImageAI 3.0.2 documentation,” 2022, [Online]. Available: https://imageai. readthedocs.io/en/latest/

[16] J. Brownlee, “How to perform object detection with YOLOv3 in Keras,” Oct. 8, 2019, [Online]. Available: https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/

[17] A. N. Huynh, “Training and detecting objects with YOLO3,” 2018, [Online]. Available: https://github.com/experiencor/keras-yolo3

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, “Scikit-learn: Machine learning in Python,” *J Mach Learn Res*, vol. 12, pp.2825–2830, 2011.

[19] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, A. Vijaykumar, A. P. Bardelli, A. Rothberg, A. Hilboll, A. Kloeckner, A. Scopatz, A. Lee, A. Rokem, C. N. Woods, C. Fulton, C. Masson, C. Häggström, C. Fitzgerald, D. A. Nicholson, D. R. Hagen, D. V. Pasechnik, E. Olivetti, E. Martin, E. Wieser, F. Silva, F. Lenders, F. Wilhelm, G. Young, G. A. Price, G. Ingold, G. E. Allen, G. R. Lee, H. Audren, I. Probst, J. P. Dietrich, J. Silterra, J. T. Webber, J. Slavič, J. Nothman, J. Buchner, J. Kulick, J. L. Schönberger, J. V. de Miranda Cardoso, J. Reimer, J. Harrington, J. L. C. Rodríguez, J. Nunez-Iglesias, J. Kuczynski, K. Tritz, M. Thoma, M. Newville, M. Kümmerer, M. Bolingbroke, M. Tartre, M. Pak, N. J. Smith, N. Nowaczyk, N. Shebanov, O. Pavlyk, P. A. Brodtkorb, P. Lee, R. T. McGibbon,

R. Feldbauer, S. Lewis, S. Tygier, S. Sievert, S. Vigna, S. Peterson, S. More, T. Pudlik, T. Oshima, T. J. Pingel, T. P. Robitaille, T. Spura, T. R. Jones, T. Cera, T. Leslie, T. Zito, T. Krauss, U. Upadhyay, Y. O. Halchenko, Y. Vázquez-Baeza, and SciPy 1.0 Contributors, "SciPy 1.0: fundamental algorithms for scientific computing in Python," *Nat Methods*, vol. 17, no. 3, pp. 261–272, 2020, 10.1038/s41592-019-0686-2.

[20] Joblib developers, 2021, "Joblib: running Python functions as pipeline jobs — joblib 1.3.0. dev0 documentation," [Online]. Available: https://joblib.readthedocs.io/en/latest/

[21] J. Sauvola and M. Pietikäinen, "Adaptive document image binarization," *Pattern Recognit*, vol. 33, no. 2, pp. 225–236, 2000, 10.1016/S0031-3203(99)00055-2.

[22] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors, "scikit-image: image processing in Python," *PeerJ*, 2014, 10.7717/peerj.453.