1.

|  |
| --- |
| Multi-Step Object Re-Identification on Edge  Devices: A Pipeline for Vehicle Re-Identification |

Tomass Zutis, Anželika Bureka, Jānis Judvaitis, Pēteris Račinskis, Jānis Ārents, Modris Greitāns

Institute of Electronics and Computer science, Latvia

**Abstract**

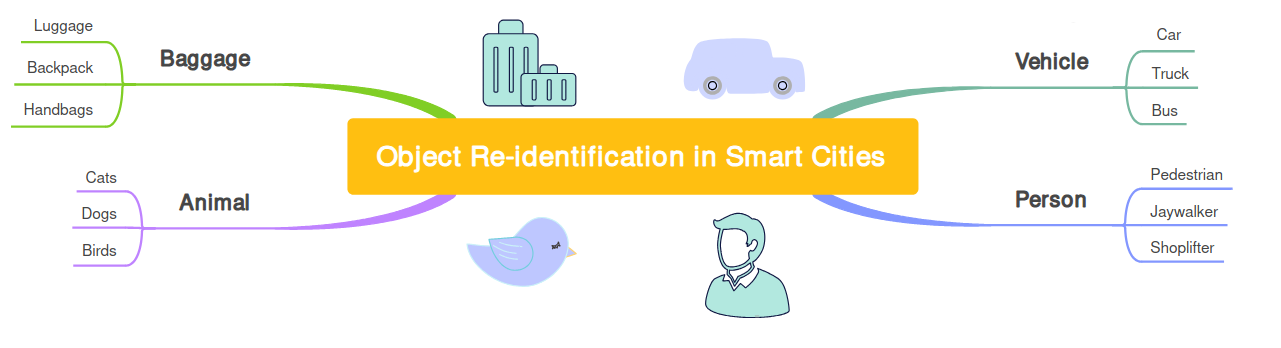
A method that leverages a multi-step process focused on extracting and using object features for object re-identification is described. The proposed pipeline includes steps such as: detecting an object, converting its features into a vector embedding, storing this embedding in a vector database, and then querying the database to find the same or similar objects based on their feature embeddings. This approach allows for the identification of the same object across different images or cameras, even in varying locations, such as in Vehicle Re-Identification scenarios. For a process like Vehicle Re-identification implementing this process on edge devices becomes crucial. Therefore, ways to tailor the pipeline and its outputs for edge devices are outlined. The paper provides a detailed explanation of the pipeline’s structure along with the experimental setup demonstrating its application, particularly in vehicle re-identification.

**Keywords** Re-Identification, Feature extraction, Vector embeddings, Neural networks, Edge devices, Vector databases, Traffic monitoring.

# Introduction

Object recognition in photos and videos has long been a key area of research, with significant advancement driven by computer vision [1]. Initially, image classification addressed the question, "What is in the image?" followed by object detection answering, "Where and what are the objects?"—largely thanks to Convolutional Neural Networks (CNNs) and their variants [2][3]. This paper focuses on object Re-Identification, a sub-problem of image retrieval, which seeks to distinguish between instances of the same class (e.g., vehicles, people - See fig. 1.1) across different scenes, despite changes in conditions (like scene, lighting conditions, object pose and others) [4][8]. At the same time, we are faced with the need to tailor these solutions for edge computing to preserve relevance in global technology and research trends.

Because of the increase in smart devices and big data, edge computing is becoming essential for real-time re-identification, especially in applications like traffic monitoring, where network cameras could transmit only processed results [5][6]. The volume of available video footage, and the influx of sensory data have made the large-scale accumulation of big data inevitable [7]. This is why fully automated systems are needed to process data and re-identify patterns and objects in the smart cities environment, for no human or group of humans can be employed to process this data manually and more crucially - in real time. We propose a pipeline to handle these tasks, efficiently processing live video feeds and identifying objects across multiple scenes.

Figure 1.1 Re-identifiable classes include people, vehicles, animals and even luggage in transit hubs.

# Related work and state of the art

## Object detection

CNNs have been incremental in computer vision tasks, including object detection. YOLO - the "You Only Look Once" model is one of the best performing and regularly maintained choices. The latest YOLO v8 version has shown significant improvements in accuracy and speed [9], which is crucial for real-time applications like ours.

## Object feature extraction

Feature extraction maps an image from its colour space to a higher-dimensional feature space [10]. Before feature extraction, multiple pre-processing stages are usually employed: normalization, thresholding, binarization, resizing and others. We can expect a model to extract colour, texture, shape, motion and localization features. A model learns intra-class variations as features when training a model on objects of one class like face features in the case of facial recognition.

As of 2024 CNNs have become predominant choices in object feature extraction thanks to their strong representation power and the ability to learn deep invariant embeddings [11].

## Vehicle re-identification

There are multiple benchmarks for Vehicle Re-ID. MBR4B-LAI model by Almeida *et al.* [12] tops the VeRi-776 benchmark, "A strong baseline" model by Huynh *et al.* [13] tops the CityFlow benchmark and the Vehiclenet model by Zheng *et al.* [14] is best at the VeRi benchmark. We pay particular interest to the paper by Zheng *et al.* because of the baseline model that is applicable to all types of object re-identification and feature extraction. In [15] Zheng *et al.* first introduces us to their baseline model and its architecture and demonstrates its capabilities specifically in pedestrian re-identification. In further papers, however, Zheng *et al.* demonstrate tailoring of this baseline to vehicle re-identification [14], [16] and person re-identification [11]. We underline this baseline models usefulness by the versatility of its use in publications, its open code base and customizability and its entry into most benchmarks. The model is successful in Rank1 precision, according to the benchmark results on Papers with Code [17]. The model is implemented in Pytorch and is based on ResNet50 pre-trained on ImageNet, although this backbone is customizable. We use the surrounding project code for this model available on [27]. The author has provisioned tools to train, finetune, test and visualize the results of the inference process.

We will further refer to this model of interest as "baseline model by Zheng *et al*.”

## Available datasets

The VehicleID introduced in [22] has each image associated with a vehicle ID. It is the dataset with one of the biggest unique ID collections and features pictures with different resolutions and quality of visibility as well as vehicles in motion state. Vehicles are mostly seen, however, from the front and the back only. The VeRi-776 was introduced in [23]. Each image is attached with vehicle ID, bounding box, type, colour, brand. The dataset has a smaller set of unique id's compared to the VehicleID, but has better quality pictures, better visibility and vehicles from all angles not just back and front. The CityFlow dataset introduced in [24] is a traffic camera dataset consisting of synchronized HD videos from 40 cameras. The quality of images is high, and vehicles can be seen from many angles. The difference between this and the previous two widely used datasets can be seen in fig. 2.1.

The VehicleX synthetic can supplement these three datasets. It contains generated images with domain adaptation from VehicleID, VeRi-776 and CityFlow [26].

 **Figure 2.1** The difference between the distribution of vehicles for the (from the left) VehicleID [22], VeRi-776 [23] and CityFlow datasets.

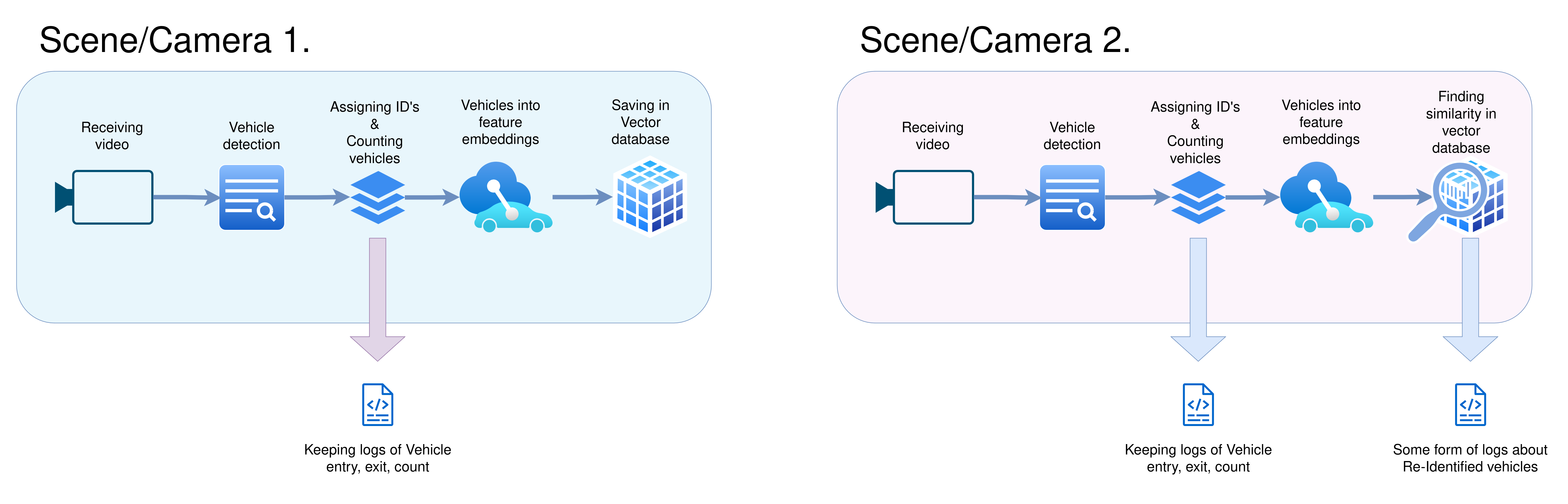
## Edge implementation

Requirements for a real-time system include implementing a network that follows the edge-computing paradigm. The edge-computing paradigm means that the video analytics run directly on the device and only the processed results and analytics are transmitted [5]. Barthélemy *et al.* have developed a pilot project where they use mobility trackers using live CCTV feeds, with twenty sensors deployed over the city with the objective of citywide traffic monitoring in real-time. The devices had the ability to transmit the outputs either over Ethernet or LoRaWAN networks and had two main components: 1.) an NVIDIA Jetson TX2 high performance and power efficient embedded computing device with special units for accelerating neural network computations used for image processing and running Ubuntu 16.04 LTS and 2.) a Pycom LoPy 4 module handling the LoRaWAN communications.

# Proposed methodology

In this chapter we propose a sequence of processes for object re-identification in the context of a smart city environment - the Vehicle ReID pipeline. It takes in video frames from camera *x*, detects the vehicles in the frames, crops images of the vehicles and saves them, turns the images into feature embeddings and saves them in a vector database. The same process would be repeated for a different camera *y ... z*, so that vehicles could be re-identified from camera *x* to *y ... z* or vice versa (See fig. 3.1).

During development, we split the pipeline into two parts – Vehicle counting and tracking and Vehicle Re-Identification.

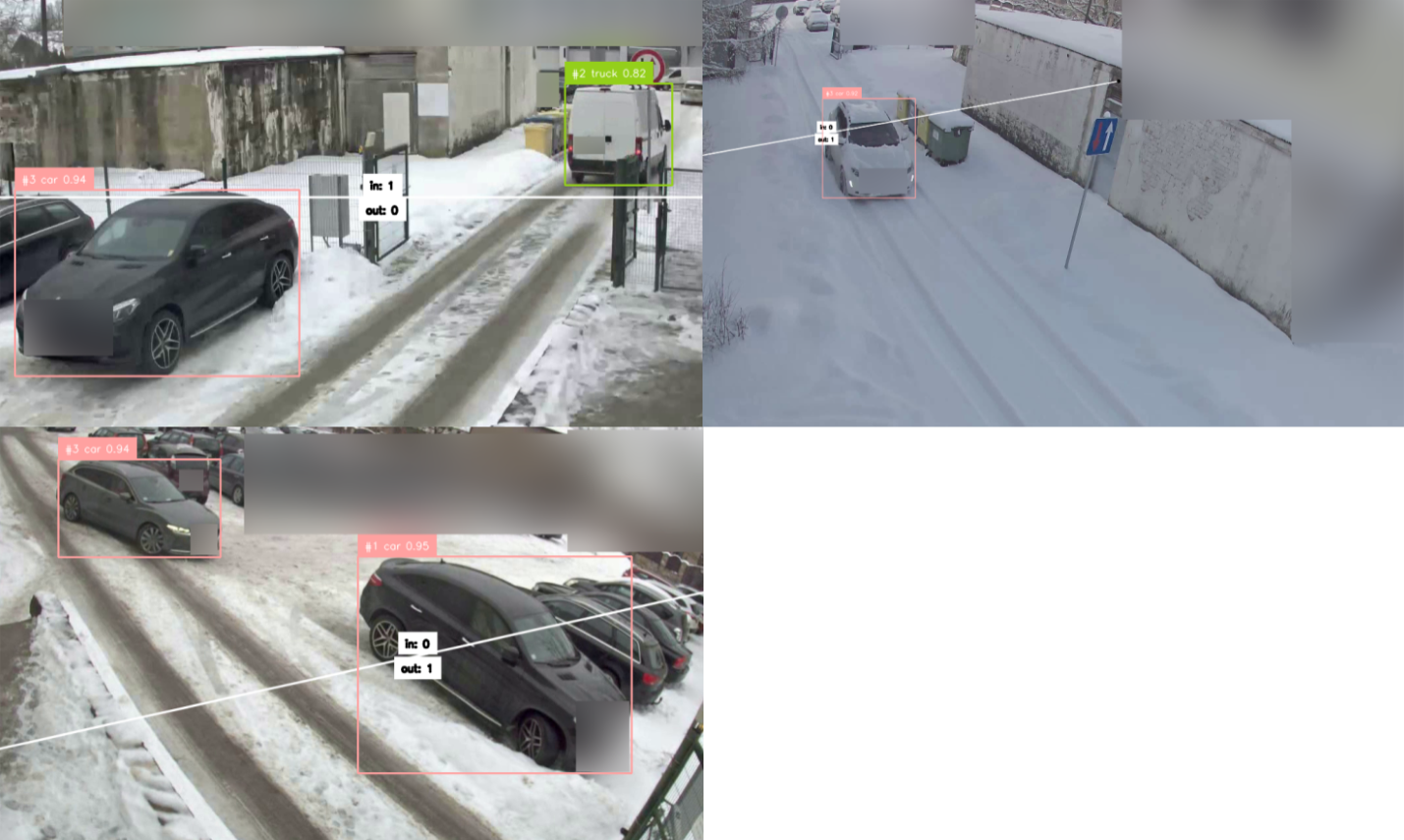
**Figure 3.1** The proposed structure of the re-identification pipeline.

## Vehicle detection, tracking and counting

First step in the larger pipeline is object detection. We receive the videos from a network camera, detect the vehicles, get their bounding boxes and start tracking them. We use YOLO v8 model (see [2.1)](#_Object_detection) for detection and we use the *ByteTrack* tracking package [32] to assign IDs to vehicles and track them through consecutive frames.

Further we establish counting criteria for incoming and outgoing cars (for example entries and exits in an intersection). This can be done with the built-in functions of *ByteTrack*, for example, *drawLine* - counting when a car drives past a drawn line (See fig. 3.2). The count, entry and exit times of cars should be continuously logged.

Before testing re-identification, verifying the accuracy of the vehicle counting step is essential. However, as this falls outside the paper's scope, we have intentionally excluded these sections.



**Figure 3.2** The implementation of the counting lines and ByteTrack in cameras (from the left, upper row) 1.,3. and (lower row) 2.

## Vehicle feature extraction and storage

Here vehicles should be cropped out of the frame. Then the baseline model by Zheng *et al.* (see [2.3](#_Vehicle_re-identification￼)) will turn the object features into n-dimensional vectors that consist of natural, real or complex numbers, where one number represents a feature or a part of a feature [28].

We require a Vector database, so vectors can be stored and queried efficiently [29]. The database must contain cosine similarity search complemented by metadata filters. Cosine similarity function is widely used and requires an input of at least two unit-length normalized vector inputs to output a vector distance [28]. We aim to store data in the form of

Key:Value = ObjectId:ObjectFeaturesVector

while more fields should be easy to add.

For this we have chosen LanceDB – an opensource database for vector search, built for efficiency in handling vector data and integration with Python [33]. It is flexible in saving and querying data.

We also introduce the following points of action:

### Datasets

We create our own “real-world” dataset for testing from footage we have gathered from our network cameras. In our custom dataset we aim to collect images of as many vehicles as the limited service-road traffic flow allows us to. We also aim to have a similar number of images per vehicle (i.e. 4-6 images, not more or less). These shots should be evenly distributed between far, medium and close distance and low, medium and high-resolution images respectively.

We will also use the three widely used and publicized datasets (mentioned in [2.4](#_Available_datasets)), to see which fits best for our real-world data and then combine the best performing standalone dataset with the Vehicle X synthetic data.

### Training hyper-parameters

We find the training parameters used and their default values in [37]. There are specific parameters that Zheng *et al.* focused on: Backbone, Learning rate, Warm epochs, Batch size, and Erasing probability.

Zheng *et al.* didn’t mention testing or the impact of other parameters that we have taken interest in. These could in theory influence the model’s performance and warrants experimentation. We have chosen to train the model with variant values for the following hyper-parameters:

* Colour jitter – enabled or disabled (disabled by default);
* Size of last linear layer – 256, 512 or 1024 (512 by default);
* Cosine learning rate – enabled or disabled (disabled by default);
* Stride – 1, 2 or 3 (2 by default).

## Edge device considerations

While the pipeline has been tested on a desktop computer with an 8GB GPU, this setup provides a rough estimate of the computational load and performance we might expect on edge devices like the NVIDIA Jetson series [30]. Current work suggests that despite differences in power consumption and architecture, the constraints with desktop testing can offer insights for edge deployment that we wish to implement in the future.

# Experimental settings

## Receiving video from a Network camera



**Figure 4.1** The same car visible in cameras (from the left) 1., 2. and 3., respectively.

To record "real-world" footage for creating a test dataset (See [3.2.1](#_Datasets)), we will use 3 AXIS P1427-LE Network cameras [31] that record footage from the same service road and a parking lot. We are receiving the video in 1280x960 resolution with ~ 2 fps. The view from the cameras (See fig. 4.1) are summarized in table 4.1.

Table 4.1 The 3 network cameras used

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Altitude (Above ground)** | **Optimal Focal zone**  **(Position, distance from cam.)** | **Direction** | **Sides of car seen** |
| 1. | ~ 3m | centre of the frame, 4m | → | Front, Back, Sides and roof (for lower cars) |
| 2. | ~ 3m | further up the road from centre, 5-6 m | ← | More from the front and the back, but skewed sides and roof are visible |
| 3. | 6-8 m | Wider area around centre, 6-8 m | ← | front/back and roof of the car visible well, sides in poor quality |
|  |  |  |  |  |

When a vehicle is at the gates, camera 1. and 2. will see the car from the opposite sides (front and rear). Camera 3. however, is located deeper into the territory and generally sees the path of a vehicle driving down the service road with the gate in a far distance.

## Vehicle re-identification

### Testing and data annotation

We have chosen the following datasets to conduct our experiments on.

* **Benchmark datasets.** By using benchmark datasets (See [2.4](bookmark://_Available_datasets)), we can access reliable test data, standardize our test metrics and evaluate the re-identification model itself.
* **CityFlow test track video.** We test the whole re-identification part of the pipeline and simulate an intersection scenario where we are re-identifying vehicles with CityFlow test tracks (See [2.4](bookmark://_Available_datasets)). We will be using the scenario Nr. 1 (intersection S01) in this dataset, to re-identify vehicles from camera 1. to camera 4. [24]. There are around 2000 frames in each video and 91 unique vehicles seen. Both cameras point to the same intersection but from vastly different locations.
* **Custom test data.** We have recorded footage from 3 of our cameras. All of them cover overlapping sections of a service road inside a closed territory. The vehicles have been cropped from these videos and saved into 3 folders, each for its own camera (See [3.2.1](#_Datasets)). This dataset contains 70-100 images from each camera, with ~ 25 unique vehicle identities. We will use this data to, first and foremost, test the generalisation of our trained models to the actual data that we will use this pipeline on.

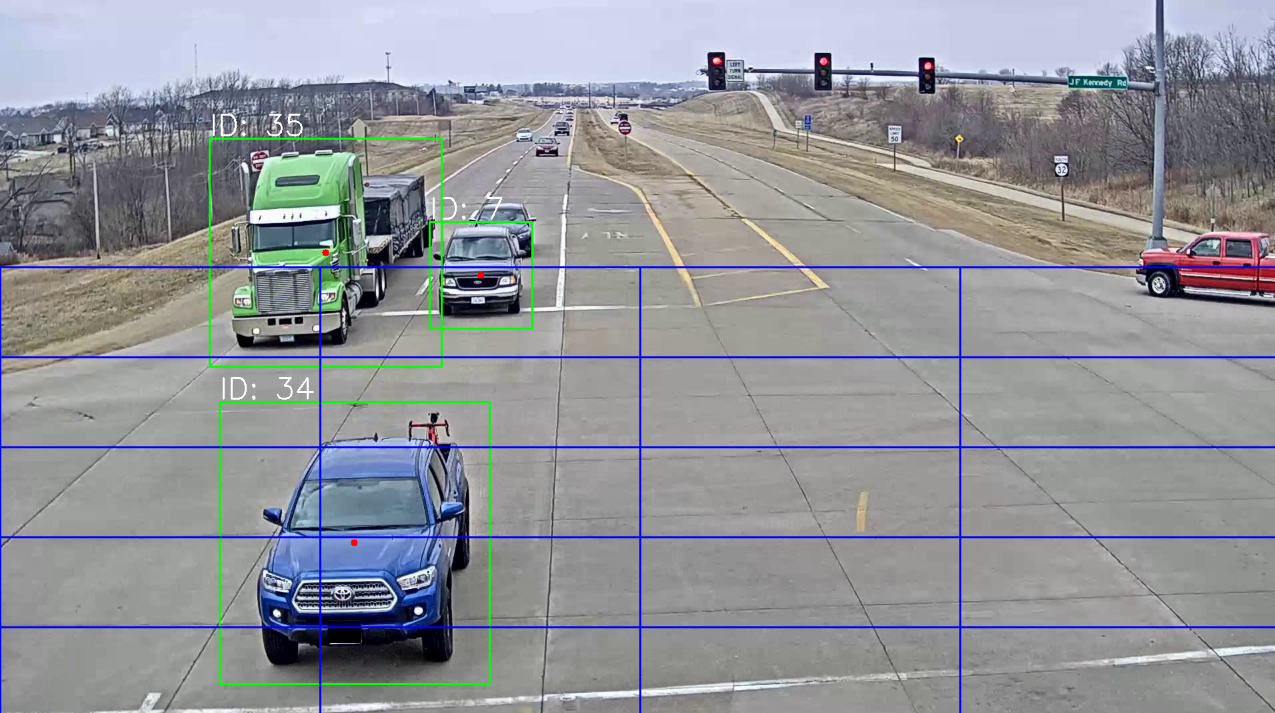
### Saving the feature extractions

We will experiment with four methods (See table 4.2) for dealing with cropping vehicles from the frame and saving them into the database:

1. **Basic Frame-by-Frame Saving**: In this method, a feature extraction of a vehicle is saved in every frame a vehicle is detected. These vectors are saved separately under the same vehicle ID in the database. Hence, there are multiple feature embeddings for the same vehicle.
2. **Vector Summing**: Instead of storing every feature vector separately, we maintain a single vector per vehicle. Each new embedding for a vehicle is summed with the existing vector, and the result is divided by the total number of updates, averaging the embeddings over time. This process keeps track of how many times the vector has been updated by adding an additional field in the database.
3. **Zone-Based Saving**: The frame is divided into zones using a grid, and a vehicle's feature embedding is saved once per zone it passes through (See figure 4.2). Typically, this results in 4-6 saved vectors per vehicle, depending on its trajectory as opposed to many more vectors when saving in each frame. Each instance of feature extraction is stored as a separate vector under the same vehicle ID.
4. **Zone-Based with Vector Summing**: Similar to the previous method, but here we apply vector summing. The vehicle’s vector is summed for every frame in which a vehicle has changed zones.

Table 4.2 Vehicle Cropping and Saving Strategies

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Number of saved vectors** | **Saving strategy** | **Embedding management** |
| Basic frame-by-frame saving | Multiple (all frames) | Per Frame | New vector for each detection |
| Vector Summing | 1 | Per Vehicle | Sum and average of all vectors |
| Zone-Based Saving | 4-6 (based on zones) | Per Zone | New vector for each detection in new zone |
| Zone-Based with Vector Summing | 1 | Per Vehicle | Sum and average for each detection in new zone |

**Figure 4.2** The implementation of the saving zones that let the pipeline capture vehicles in multiple positions as seen on the CityFlow test track video.

# Results

In this section we go over the results of the experiments.

## Performance metrics

* **Rank-1, Rank-X Accuracy**: Measures the proportion of examples for which the predicted label matches the single target label (Rank-1) or any of the top X predictions match the label (Rank-X) [34].
* **mAP (Mean Average Precision)**: Average precision (AP) is the average of accuracy values at all rankings where relevant objects are retrieved, and mAP - the average of all APs provided [35].
* **Micro Precision and Micro Recall**: Calculated by aggregating true positives, false positives, and false negatives across every query (instance) and doesn't consider possible class over/under representation [36].
* **Macro Precision and Macro Recall**: Precision and recall calculated separately for each class and then averaged, giving equal weight to all classes regardless of their size [36].
* **Validation Loss**: A measure of how well a model is learning during validation step of training. For training we use Cross Entropy Loss and Triplet Margin Loss together (summed).

## Dataset generalization

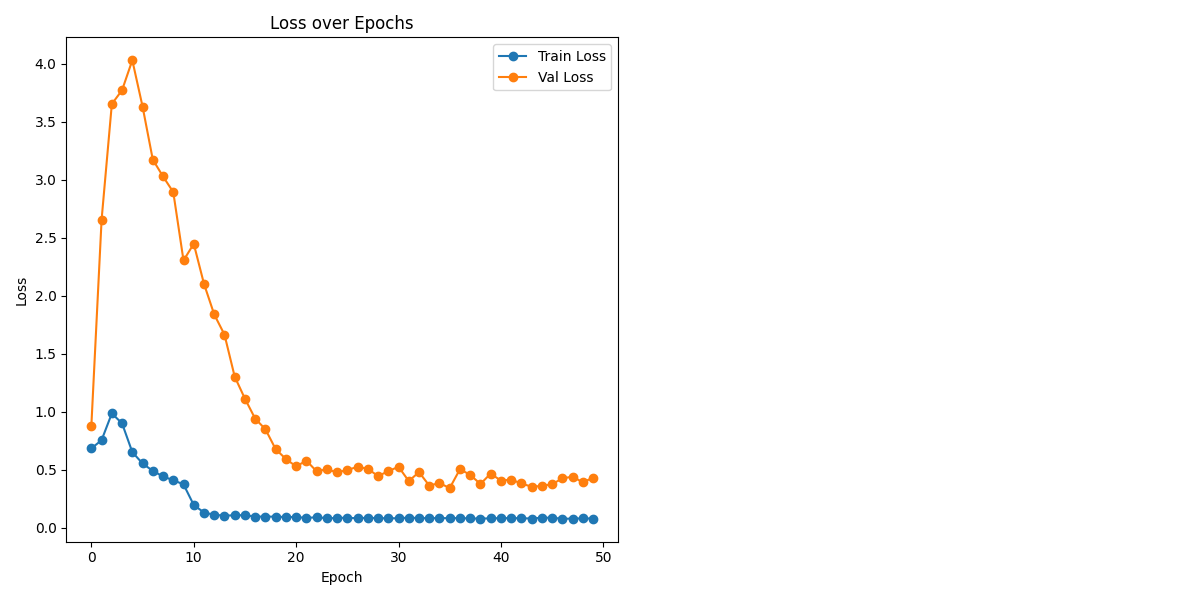
We aim to find the dataset, that makes the model generalize best on our own recorded data (See [3.2.1)](bookmark://_Training_the_ReID). For this we will use the Rank-1 accuracy (See [5.1](#_Performance_metrics_used)), that has been averaged from 3 tests on our custom dataset.

We use the assumed default training parameters as seen in [37].

Table 5.1 Testing on our custom data, collected from our Network cameras

|  |  |  |  |
| --- | --- | --- | --- |
| **Dataset used** | Veri | Vehicle-ID | CityFlow |
| **Averaged Rank-1 accuracy, %** | **65.81** | 44.85 | 65.44 |

As we show in table 5.1 - training our baseline model on the Veri dataset has the best result (we compare values at epoch 19. to reduce training times, see fig. 5.1). Even though the difference between VeRi and Cityflow datasets may be insignificant, visually the vehicles seen in the Veri dataset seem to resemble our collected data more.



**Figure 5.1** Model training loss values on the VeRi dataset. We find values plateauing after 20 epochs.

## Hyper-parameters

We have trained the model on all combinations of the parameters of interest (see [3.2.2)](bookmark://_Training_hyper-parameters) over 15. epochs and recorded their loss values. Training the model for 36 times in total, consisting of 540 epochs. The rest of the training parameters have been left with the default values. The 10 entries with the lowest validation loss value (see [5.1](#_Performance_metrics_used)) we showcase in table 5.2.

Table 5.2 Validation results during training

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nr.** | **Loss** | **Stride** | **Lin. Num.** | **Color jitter** | **Cosine learning rate** | **Epoch** |
| 1 | 0.0082 | **1** | **256** | **1** | 0 | 15. |
| 2 | 0.0083 | **1** | **256** | **1** | 1 | 15. |
| 3 | 0.0084 | 2 | **256** | 0 | 0 | 15. |
| 4 | 0.0084 | 3 | **256** | **1** | 1 | 15. |
| 5 | 0.0086 | 3 | **256** | 0 | 0 | 15. |
| 6 | 0.009 | **1** | **256** | 0 | 0 | 13. |
| 7 | 0.0092 | **1** | **256** | 0 | 0 | 15. |
| 8 | 0.0092 | **1** | **256** | **1** | 1 | 13. |
| 9 | 0.0092 | **1** | 512 | **1** | 0 | 11. |
| 10 | 0.0093 | **1** | **256** | **1** | 0 | 11. |

Parameters of importance seem to be Stride, Linear number and Colour jitter. A specific option for colour jitter is represented 6/10 , stride – 7/10 and linear number – 9/10 times in the top 10 entries, signalling their importance. We see that the first entries in the table suggest that the model generalises best to validation data with a Stride of 1, Number of linear layers of 256 and Color Jitter enabled.

## Performance on the VeRi-776 benchmark

We use the VehicleNet Paper [14] reported results as a benchmark, but we do not directly compare them with the baseline model by Zheng *et. al.* used by us.

In our experiments, we initially trained our model using the VeRi dataset (See [5.2](#_Dataset_generalization)). As shown in Table 5.3, while using the default training hyper-parameters we evaluated the model's performance over 20 and 50 epochs on this dataset, where the model trained on 50 epochs outperforms the one with 20.

Table 5.3 VeRi Trained Model comparisons on the VeRi-776 benchmark

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Rank-1** | **Rank-5** | **Rank-10** | **mAP** |
| VehicleNet reported results | *96.78* | - | - | *83.41* |
| Model trained on Veri, 19th epoch | 93.80 | 97.61 | 98.74 | 68.96 |
| Model trained on Veri, 49th epoch | 94.04 | 97.49 | 98.80 | 69.4 |
| Model trained on Veri, 49th epoch with updated parameters | 95.47 | 97.62 | 98.51 | 71.70 |
| Model trained on Veri and VehicleX, 49th epoch with updated parameters | **95.83** | **98.09** | **98.87** | **73.64** |

Training the model with updated parameters (see [5.3](#_Hyper-parameters)) demonstrates further improvement on the benchmark.

Lastly, we conducted additional training by incorporating the VehicleX synthetic data into the training process (see [2.4](#_Available_datasets) and [3.2.1](#_Datasets)).

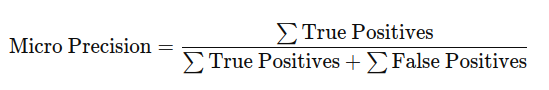
All together our efforts have increased mAP by 4.24% and Rank-1 by 1.79% (See [5.1](#_Performance_metrics_used)) from “Model trained on Veri, 49th epoch” to “Model trained on Veri and VehicleX, 49th epoch with updated parameters”.

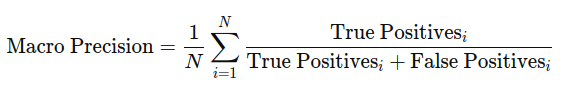
## Re-identification testing on test data from our cameras

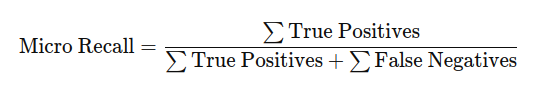
In this section we test the model performance on the test data gathered from our network cameras (See [4.1](#_Receiving_video_from) and [4.2.1](#_Testing_and_data)).

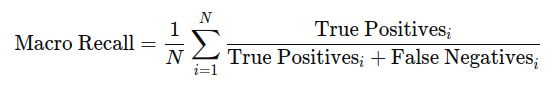
### Camera to camera re-identification

Table 5.4 contains the Micro and Macro precision values and table 5.5 contains the Micro and Macro recall values (See [5.1](#_Performance_metrics_used)), when re-identifying from a query camera *x* (rows) to gallery camera *y* (columns).

(5.1)

(5.2)

(5.3)

 (5.4)

Where: *N* is the total number of classes,

True positivesi are the number of correct predictions for class *i*,

False negativesi are the incorrect predictions for class *i*.

Table 5.4 Precision when re-identifying from a query camera to a gallery camera

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Micro pr.** | **Macro pr.** | **Camera 1** | | **Camera 2** | | **Camera 3** | |
| **Camera 1** | |  |  | 64.37 | 65.09 | 78.08 | 77.29 |
| **Camera 2** | | 66.13 | 57.58 |  |  | 100 | 100 |
| **Camera 3** | | 85.71 | 78.68 | 73.58 | 76.98 |  |  |

Table 5.5 Recall when re-identifying from a query camera to a gallery camera

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Micro r.** | **Macro r.** | **Camera 1** | | **Camera 2** | | **Camera 3** | |
| **Camera 1** | |  |  | 64.37 | 66.00 | 78.08 | 76.28 |
| **Camera 2** | | 66.13 | 46.22 |  |  | 100 | 100 |
| **Camera 3** | | 85.71 | 75.75 | 73.58 | 76.67 |  |  |

Judging by the precision and recall values we can establish the following conclusions:

* The model generalizes reasonably well to our custom testing data
* The percentages and differences in micro/macro values suggest the model may be overfitting to certain well-represented vehicle identities in camera 2 and hints at difficulty in recognizing rare vehicles elsewhere.
* Overall, it seems the model can generalise camera 3. feature extractions the best against all scenes but has trouble generalising camera 1. and 2. feature extractions against each other.

### Sets of cameras

We can even out the results of the different cameras if a car has passed through at least two cameras, which has allowed us to gather more data in the database of any given car. Let’s use a test, where we query vehicles of one camera from a gallery of two cameras. The cameras column specifies the camera *x* (query), who’s images are queried in the camera *y,z* (gallery) images.

Table 5.6 Macro, micro precision and recall when re-identifying from a query camera to a gallery of two cameras

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Cameras** | **Micro Precision, %** | **Macro Precision, %** | **Micro Recall, %** | **Macro Recall, %** |
| **1 → 2,3** | 73.39 | 80.63 | 73.39 | 74.96 |
| **2 → 1,3** | 75.81 | 67.09 | 75.81 | 59.38 |
| **3 → 2,1** | 85.71 | 89.21 | 85.71 | 88.17 |

Table 5.6 shows that re-identification for vehicles that have passed through at least two cameras is more reliable. Camera 3 shows the highest performance with both micro and macro values exceeding 85%, indicating strong generalization across vehicle identities. In contrast, re-identifying from camera 2 to cameras 1 and 3 results in the lowest macro recall (59.38%), suggesting difficulty in retrieving true matches for less represented vehicle identities. Overall, the model performs well, but the noticeable drop in macro values for camera 2 queries points to possible need to ease class imbalance in the future.

In practice the results may indicate that cameras that have a slightly zoomed out or aerial view of the roads or intersections (Camera 3.) can produce feature extractions that generalize better than cameras that see the vehicles up close. It can also be observed that camera 2. had a 100% re-identification precision when querying against camera 3., which could be explained by the fact that both cameras 2. and 3. had similar angles with only height varying. What is harder to conclude is why the same was not true for camera 3. querying against camera 2. Similarly, it can be observed that cameras 1. and 2. have opposite viewpoints, so that even if the car went back and forth through both cameras, each camera would only have a flipped image of what the other camera has. This explains the poorer performance between camera 1. and 2.

## Testing the whole re-identification part of the pipeline

We test our model on the CityFlow video tracks - re-identifying vehicles from (intersection) S01 traffic camera 4. to camera 1.

To make the process as close to real world as possible we will use the Rank-1 accuracy (See [5.1](#_Performance_metrics_used)) to measure the accuracy of the pipeline, since in a scenario like this we simply care for whether the vehicle has been re-identified correctly or not.

As mentioned in section [4.2.2](#_Saving_the_feature) , we test the 4 approaches of capturing the feature embeddings and saving them into a database.

Table 5.7 Testing methods of the re-identification pipeline on CityFlow video tracks

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test scenario** | Basic Frame-by-Frame Saving: | Vector Summing | Zone-Based Saving | **Zone-Based with Vector Summing:** |
| **Rank-1 accuracy in %** | 65.52 | 62.78 | 72.90 | **74.13** |
| **Test duration in seconds** | 1107.80 | 1085.30 | 330.12 | **328.90** |

Overall, the tests on CityFlow video tracks show that capturing vehicles in distinct zones improves both accuracy and efficiency (See table 5.7).

# Future research

Although this paper proposes a complete pipeline for object re-identification on edge devices, multiple areas remain for future research. First, optimizing the pipeline for specific edge devices like the Nvidia Jetson by profiling performance and applying techniques such as model pruning, quantization, or TensorRT for efficient inference. Additionally, fine-tuning models trained on public datasets with our in-house data could reduce distribution shifts and improve real-world performance. Finally, a more in-depth analysis of feature vectors is needed to better understand which components are relevant for re-identification accuracy and which are redundant.

# Conclusion

In this paper, we presented a multi-step pipeline for object re-identification, focusing on real-time applications using edge devices. The pipeline handles object detection, feature extraction, and matching through a vector database, demonstrating reliable vehicle re-identification across various scenes. Performance dynamics were evaluated by comparing different datasets, models, pipeline processes. The authors observed the position and angle of cameras significantly influencing the accuracy of vehicle re-identification, with higher or wider vantage points producing better feature extractions for generalization across scenes.

Edge computing is required to achieve real-time performance - it allows the processing to occur locally on the device, reducing latency and data transmission needs. Our future work will focus on optimizing this pipeline for low-power edge devices, which are essential for deployment in smart cities. The results indicate that, with further refinements, this system can significantly enhance object re-identification processes for custom scenarios.

**References**

1. S. J. Prince, *Computer vision: models, learning, and inference.* Cam-bridge University Press, 2012.
2. D. Lu and Q. Weng, “A survey of image classification methods and tech-niques for improving classification performance,” *International journal of Remote sensing*, vol. 28, no. 5, pp. 823–870, 2007.
3. J. Du, “Understanding of object detection based on cnn family and yolo,” in *Journal of Physics: Conference Series*, vol. 1004. IOP Publishing, 2018, p. 012029.
4. X. Li, Z. Zhou, P. L. Mazzeo, S. Ramakrishnan, and P. Spagnolo, “Object re-identification based on deep learning,” Visual Object Tracking with Deep Neural Networks, 2019.
5. J. Barthélemy, N. Verstaevel, H. Forehead, and P. Perez, “Edge-computing video analytics for real-time traffic monitoring in a smart city,” Sensors, vol. 19, no. 9, p. 2048, 2019.
6. C. Wang, Y. Yang, M. Qi, and H. Ma, “Efficient cloud-edge collaborative inference for object re-identification,” arXiv preprint arXiv:2401.02041, 2024.
7. K. Cao, Y. Liu, G. Meng, and Q. Sun, “An overview on edge computing research,” IEEE access, vol. 8, pp. 85 714–85 728, 2020.
8. R. Kuma, E. Weill, F. Aghdasi, and P. Sriram, “Vehicle re-identification: an efficient baseline using triplet embedding,” in 2019 International Joint Conference on Neural Networks (IJCNN). IEEE, 2019, pp. 1–9.
9. S. Rath, “Yolov8 : Comprehensive guide to state of the art object detection,” <https://learnopencv.com/ultralytics-yolov8/>#YOLOv8-vs-YOLOv5, 2023, accessed: September 23, 2024.
10. A. O. Salau and S. Jain, “Feature extraction: a survey of the types, techniques, applications,” in 2019 international conference on signal processing and communication (ICSC). IEEE, 2019, pp. 158–164.
11. Z. Zheng, X. Yang, Z. Yu, L. Zheng, Y. Yang, and J. Kautz, “Joint discriminative and generative learning for person re-identification,” in proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019, pp. 2138–2147.
12. E. Almeida, B. Silva, and J. Batista, “Strength in diversity: Multi-branch representation learning for vehicle re-identification,” in 2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC). IEEE, 2023, pp. 4690–4696.
13. S. V. Huynh, “A strong baseline for vehicle re-identification,” in Pro-ceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2021, pp. 4147–4154.
14. Z. Zheng, T. Ruan, Y. Wei, and Y. Yang, “Vehiclenet: Learning robust feature representation for vehicle re-identification.” in CVPR Workshops, vol. 2, no. 3, 2019.
15. Z. Zheng, L. Zheng, and Y. Yang, “A discriminatively learned cnn embedding for person reidentification,” ACM transactions on multimedia computing, communications, and applications (TOMM), vol. 14, no. 1, pp. 1–20, 2017.
16. Z. Zheng, T. Ruan, Y. Wei, Y. Yang, and T. Mei, “Vehiclenet: Learning robust visual representation for vehicle re-identification,” IEEE Trans-actions on Multimedia, vol. 23, pp. 2683–2693, 2020.
17. Papers with Code, “Vehicle re-identification benchmarks,” <https://paperswithcode.com/task/vehicle-re-identification#benchmarks>, accessed: September 23, 2024.
18. H. Liu, Y. Tian, Y. Yang, L. Pang, and T. Huang, “Deep relative distance learning: Tell the difference between similar vehicles,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2167–2175.
19. X. Liu, W. Liu, T. Mei, and H. Ma, “A deep learning-based approach to progressive vehicle re-identification for urban surveillance,” in Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14. Springer, 2016, pp. 869–884.
20. Z. Tang, M. Naphade, M.-Y. Liu, X. Yang, S. Birchfield, S. Wang, R. Kumar, D. Anastasiu, and J.-N. Hwang, “Cityflow: A city-scale benchmark for multi-target multi-camera vehicle tracking and re-identification,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 8797–8806.
21. Y. Yao, L. Zheng, X. Yang, M. Naphade, and T. Gedeon, “Simulating content consistent vehicle datasets with attribute descent,” in Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VI 16. Springer, 2020, pp. 775–791.
22. layumi, “Pytorch reid github repository,” https://github.com/layumi/Person reID baseline pytorch/, accessed: September 24, 2024.
23. R. J. Bayardo, Y. Ma, and R. Srikant, “Scaling up all pairs similarity search,” in Proceedings of the 16th international conference on World Wide Web, 2007, pp. 131–140.
24. T. Taipalus, “Vector database management systems: Fundamental concepts, use-cases, and current challenges,” Cognitive Systems Research, vol. 85, p. 101216, 2024.
25. S. Valladares, M. Toscano, R. Tufiño, P. Morillo, and D. Vallejo-Huanga, “Performance evaluation of the nvidia jetson nano through a real-time machine learning application,” in Intelligent Human Systems Integration 2021: Proceedings of the 4th International Conference on Intelligent Human Systems Integration (IHSI 2021): Integrating People and Intelligent Systems, February 22-24, 2021, Palermo, Italy. Springer, 2021, pp. 343–349.
26. AXIS communications, “Product support for axis p1427-le network camera,” <https://www.axis.com/products/axis-p1427-le/support>, accessed: September 26, 2024.
27. Y. Zhang et al., “ByteTrack: Multi-Object Tracking by Associating Every Detection Box,” *arXiv (Cornell University)*, Oct. 2021, doi: <https://doi.org/10.48550/arxiv.2110.06864>.
28. lancedb, “GitHub - lancedb/lancedb: Developer-friendly, serverless vector database for AI applications. Easily add long-term memory to your LLM apps!,” *GitHub*, Sep. 24, 2024. https://github.com/lancedb/lancedb (accessed Oct. 03, 2024).
29. V. Shankar, R. Roelofs, H. Mania, A. Fang, B. Recht, and L. Schmidt, “Evaluating Machine Accuracy on ImageNet,” vol. 1, pp. 8634–8644, Jul. 2020.
30. M. Talha, Mian Muhammad, et al. Deep learning in news recommender systems: A comprehensive survey, challenges and future trends. Neurocomputing, 2023, 126881.
31. Jean-Charles Lamirel, Maha Ghribi, and Pascal Cuxac, “Unsupervised recall and precision measures: a step towards new efficient clustering quality indexes,” Aug. 2010.
32. regob, “GitHub - regob/vehicle\_reid: Vehicle Re-identification,” GitHub, 2022. https://github.com/regob/vehicle\_reid/tree/master (accessed Oct. 14, 2024).