11

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

Tomass Zutis, Peteris Racinskis, Anzelika Bureka, Janis Judvaitis, Janis Arents, and Modris Greitans

Institute of Electronics and Computer Science (EDI), Latvia

Abstract

In modern computer vision tasks, the ability to identify and track objects across different scenes and environments has become important for numerous applications, especially in transportation. Inspired by this need, we propose a method that leverages a multi-step process focused on extracting and using object features for object re-identification.The proposed pipeline includes the following steps: 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 enables us to identify the same object across different images or cameras, even in varying locations. This is essential in scenarios like Vehicle Re-Identification. For such scenario, implementing this process on edge devices is crucial. Therefore, ways to tailor the pipeline and its outputs for edge devices are outlined. The paper details the pipeline's structure along with the experimental setup demonstrating its application, particularly in vehicle re-identification. The pipeline achieves 70-80% re-identification precision when dealing with vehicle images from our network cameras and above 70% Rank-1 accuracy when dealing with a CityFlow video track scenario.

Keywords: vehicle re-identification, feature extraction, re-identification pipeline, computer vision, traffic monitoring.

11.1 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, which is a sub-problem of image retrieval. Object re-identification aims to distinguish instances of the same class, such as vehicles and persons (as illustrated in Figure 11.1), across different scenes despite changes in conditions like scene, lighting , or object pose [4][5]. However, truly impactful re-identification research in 2024 must support edge computing.

Because of the increase in processing latency and big data, edge computing is becoming essential for real-time re-identification, especially in applications like traffic monitoring, where network cameras should transmit only processed results [6][7]. The volume of available video footage, and the influx of sensory data have made the large-scale accumulation of big data inevitable [8]. This is why fully automated systems are needed for processing data and re-identifying objects in smart cities. Manual processing by humans is not feasible, especially when real-time decisions are required. We propose a pipeline to handle these tasks of efficiently processing live video feeds and identifying objects across multiple scenes. The novelty of this research lies in the integration of state-of-the-art methods into a unified pipeline, tested on

Figure 11.1 Re-identifiable classes in smart city environments.

real-world vehicle re-identification scenarios that fit into smart city initiatives and traffic management using edge computing solutions.

11.2 Related work and state of the art

11.2.1 Object detection

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

11.2.2 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 became the predominant choice in object feature extraction thanks to their strong representation power and their ability to learn deep invariant embeddings [11].

11.2.3 Vehicle re-identification

There are multiple benchmarks for Vehicle Re-ID. MBR4B-LAI model [12] tops the VeRi-776 benchmark, "A strong baseline" model [13] tops the CityFlow benchmark and the VehicleNet model [14] is best at the VeRi benchmark. We pay particular interest to [14] by Zheng *et al.* because of the baseline model that is applicable to all types of object re-identification and feature extraction. In [15] the same author 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 [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 [18]. The author has provisioned tools to train, finetune, test and visualize the results of the inference process. We further refer to this model as "baseline model".

11.2.4 Available datasets

The VehicleID introduced in [19] 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 [20]. 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 [21] 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 Figure 11.2.

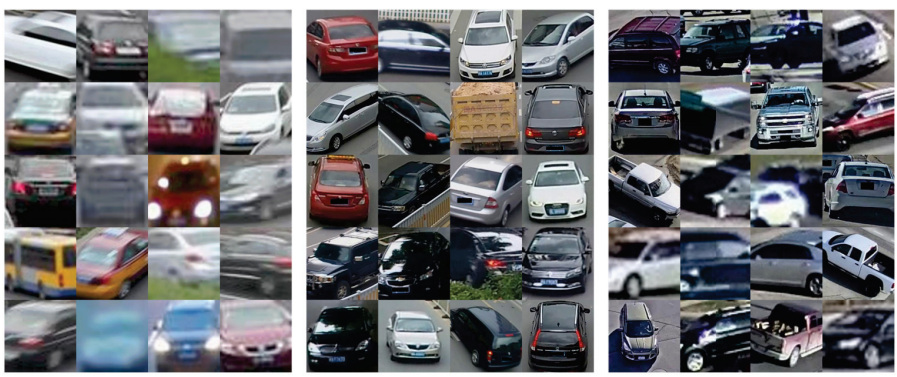

Figure 11.2 The difference between the distribution of vehicles for the (from the left) VehicleID, VeRi and CityFlow datasets.

The VehicleX synthetic data can supplement these three datasets. It contains generated images with domain adaptation from VehicleID, VeRi-776 and CityFlow [22].

11.2.5 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. [6] 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.

11.3 Proposed methodology

We propose a sequence of processes for object re-identification in the context of a smart city environment. 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 is repeated for a different camera y ... z, so that vehicles can be re-identified from camera x to y ... z or vice versa. This has been illustrated in Figure 11.3.

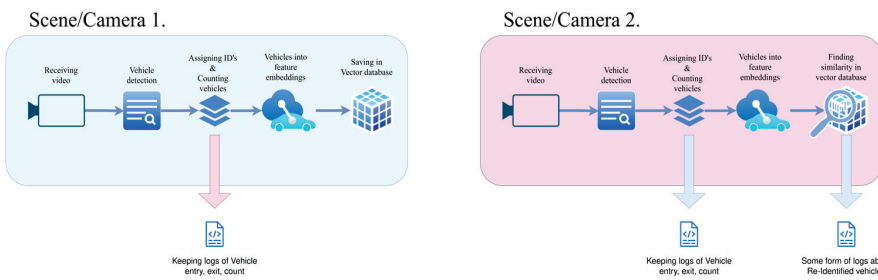

Figure 11.3 The proposed structure of the re-identification pipeline.

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

11.3.1 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 the YOLO v8 model for detection and the *ByteTrack* tracking package [23] 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 as seen in Figure 11.4. 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.

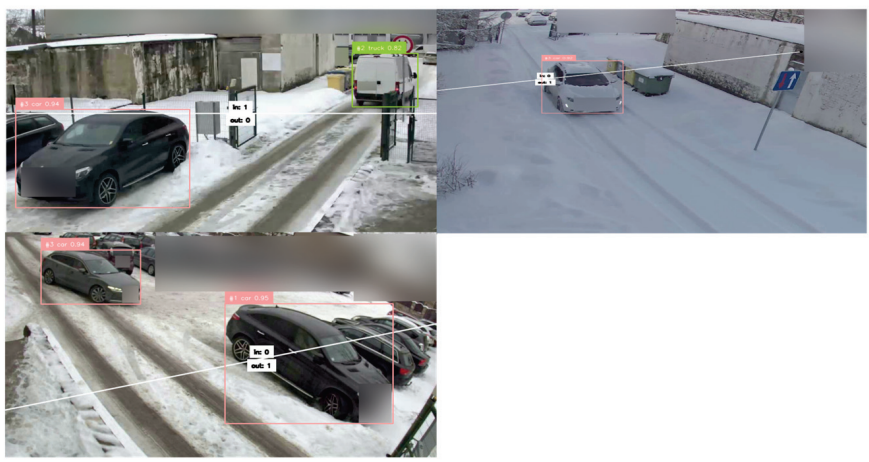

Figure 11.4 The implementation of the counting lines and ByteTrack in cameras (from the left, upper row) 1.,3. and (lower row) 2.

11.3.2 Vehicle feature extraction and storage

Consequently vehicles should be cropped out of the frame. Then the re-identification model 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 [24].

We require a Vector database, so vectors can be stored and queried efficiently [25]. The database must contain cosine similarity search complemented by metadata filters. The cosine similarity function is widely used and requires an input of at least two unit-length normalized vector inputs to output a vector distance [24]. We aim to store data in the form of

$$\text{Key:Value} = \text{ObjectId:ObjectFeaturesVector}$$

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 [26]. It is flexible in saving and querying data.

We also introduce the following points of action:

11.3.2.1 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 public datasets that we've discussed in 2.4, to see which fits best for our real-world data and then combine the best performing standalone dataset with the Vehicle X synthetic data.

11.3.2.2 Training hyper-parameters

The training parameters and their default values are found in [27], including Backbone, Learning rate, Warm epochs, Batch size, and Erasing probability. To further explore the model's performance, we conducted experiments by varying additional hyperparameters, specifically:

- Colour jitter: enabled or disabled;
- Size of the last linear layer: 256, 512, or 1024;
- Cosine learning rate: enabled or disabled;
- Stride: 1, 2, or 3.

These variations were designed to evaluate the potential impact of these hyperparameters on training outcomes.

11.3.3 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 [28]. 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.

11.4 Experimental settings

11.4.1 Receiving video from a Network camera

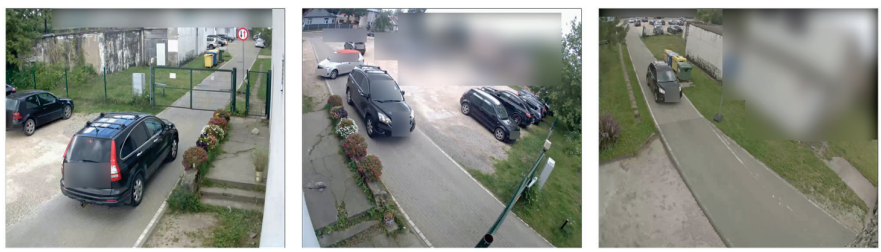

Figure 11.5 The same car visible in our network cameras (from the left) 1., 2. and 3., respectively.

To record "real-world" footage for the dataset we've envisioned in section 3.2.1, we will use 3 AXIS P1427-LE Network cameras [29] that record footage from the same service road and a parking lot. We are receiving the video in 1280×960 resolution with $\sim$ 2 fps. The view from the cameras illustrated in Figure 11.5. are summarized in Table 11.1.

When a vehicle is at the gates, camera 1 and 2 will see the car from the opposite sides (front and rear). Camera 3 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.

Table 11.1 The 3 network cameras used

#	Altitude (Above ground)	Optimal Focal zone (Position, distance from cam.)	Direction	Sides of car seen
1.	$\sim$ 3m	centre of the frame, 4m	$\rightarrow$	Front, Back, Sides and roof (for lower cars)
2.	$\sim$ 3m	further up the road from centre, 5-6 m	$\leftarrow$	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	$\leftarrow$	front/back and roof of the car visible well, sides in poor quality

11.4.2 Vehicle re-identification

11.4.2.1 Testing and data annotation

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

- **Benchmark datasets.** By using the three benchmark datasets we discussed in section 2.4, 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. We will be using the scenario Nr. 1 (intersection S01) in this dataset, to re-identify vehicles from camera 1 to camera 4 [21]. 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 were cropped from these videos and saved into 3 folders, each for its own camera. This dataset contains 70-100 images from each camera, with $\sim$ 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.

11.4.2.2 Saving the feature extractions

We will experiment with four methods (See their comparison in Table 11.2) 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. The zones can be seen illustrated blue in Figure 11.6. 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.

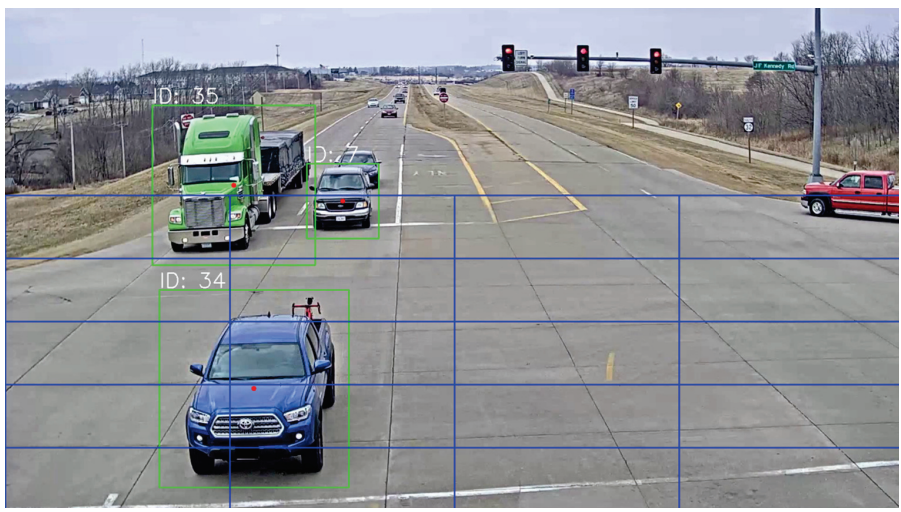

Figure 11.6 The implementation of the saving zones (drawn as blue rectangles) as seen on the CityFlow test track video.

Table 11.2 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

11.5 Results

This section discusses the results of the experiments.

11.5.1 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) [30].
- **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 [31].
- **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 [32].
- **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 [32].
- **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).

11.5.2 Dataset generalization

We aim to find the benchmark dataset, that makes the baseline model generalize best on our own recorded data. For this we will use the Rank-1 accuracy metric as described in section 5.1, that has been averaged from 3 tests on our custom dataset.

Table 11.3 Testing on our custom dataset

Training dataset	Veri	Vehicle-ID	CityFlow
Averaged Rank-1 accuracy, %	**65.81**	44.85	65.44

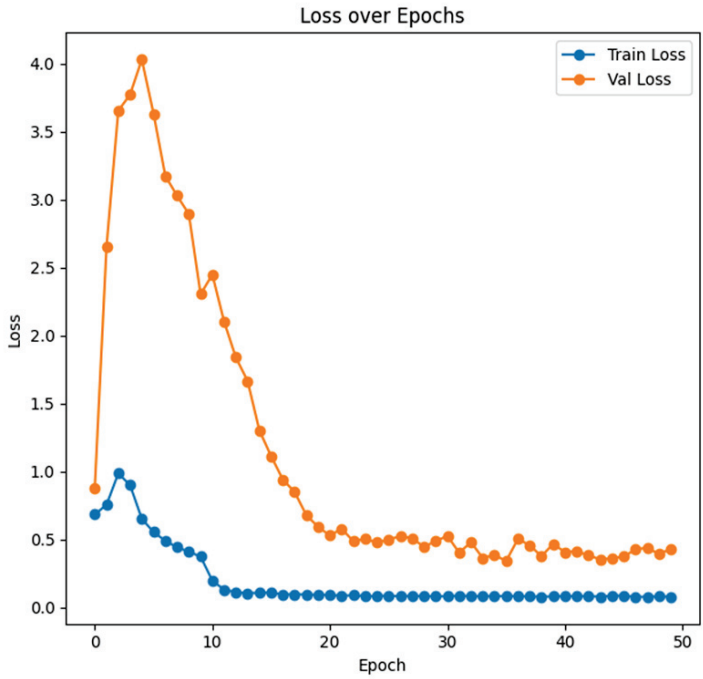

Figure 11.7 Model loss values during training on the VeRi dataset. We find values plateauing after 20 epochs.

We use the assumed default training parameters as seen in [27].

Table 11.1 indicates that training the baseline model on the Veri dataset provides the best result (we compare values at epoch 19. to reduce training times as depicted in Figure 11.7). Even if the difference between VeRi and Cityflow datasets is insignificant, visually the vehicles seen in the Veri dataset resembles our collected data more.

11.5.3 Hyper-parameters

We have trained the model on all combinations of the hyperparameters that we have outlined in section 3.2.2 over 15 epochs and recorded their loss

Table 11.4 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.

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 we showcase in Table 11.4.

The parameters that appear to have the greatest impact are Stride, Linear Layer Size, and Colour Jitter. Colour Jitter was enabled in 6 out of the 10 best results, Stride was set to 1 in 7 of the top results, and the Linear Layer Size was set to 256 in 9 out of the top 10 results. This highlights the significance of these parameters in achieving best performance.We will further train the model with a stride of 1, number of linear layers of 256 and color jitter enabled.

11.5.4 Performance on the VeRi-776 benchmark

We use the VeRi-776 dataset as a benchmark. We compare the results of our improvements to the unmodified baseline model.

In our experiments, we initially trained the baseline model using the VeRi dataset. As shown in Table 11.5 we used the default training hyper-parameters and evaluated the model's performance over 20 and 50 epochs. The model trained on 50 epochs outperforms the one with 20.

Training the model with the updated parameters of stride, linear number and color jitter (see Section 5.3) demonstrates further improvement on the benchmark.

Table 11.5 VeRi Trained Model comparisons on the VeRi-776 benchmark

Model	Rank-1	Rank-5	Rank-10	mAP
Baseline trained on Veri, 19^{th} epoch	93.80	97.61	98.74	68.96
Baseline trained on Veri, 49^{th} epoch	94.04	97.49	98.80	69.4
Model with updated parameters trained on Veri, 49^{th} epoch	95.47	97.62	98.51	71.70
Model with updated parameters trained on Veri and VehicleX, 49^{th} epoch	**95.83**	**98.09**	**98.87**	**73.64**

Lastly, we conducted additional training by incorporating the VehicleX synthetic data into the training process.

All together our efforts have increased mAP by 4.24% and Rank-1 by 1.79% from "Model trained on Veri, 49th epoch" to "Model trained on Veri and VehicleX, 49th epoch with updated parameters".

11.5.5 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.

11.5.5.1 Camera to camera re-identification

Table 11.4 contains the Micro and Macro precision values and table 5.5 contains the Micro and Macro recall values (see the description of these metrics in section 5.1), when re-identifying from a query camera *x* (rows) to gallery camera *y* (columns).

$$\text{Micro Precision} = \frac{\sum \text{True Positives}}{\sum \text{True Positives} + \sum \text{False Positives}} \tag{11.1}$$

$$\text{Macro Precision} = \frac{1}{N}\sum_{i=1}^{N} \frac{\text{True}^{N}\,\text{Positives}_{i}}{\text{True Positives}_{i} + \text{False Positives}_{i}^{\text{Pa}}}. \tag{11.2}$$

$$\text{Micro Recall} = \frac{\sum \text{True Positives}}{\sum \text{True Positives} + \sum \text{False Negatives}} \tag{11.3}$$

$$\text{Macro Recall} = \frac{1}{N}\sum_{i=1}^{N} \frac{\text{True}^{\text{Positives}_{i}}}{\text{True Positives}_{i} + \text{False Negatives}_{i}^{\text{Nat}}} \tag{11.4}$$

Table 11.6 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 11.7 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		

Where: N is the total number of classes,
True positives$_i$ are the number of correct predictions for class i,
False negatives$_i$ are the incorrect predictions for class i.

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.

Table 11.8 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

11.5.5.2 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 11.8 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.

11.5.6 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

Table 11.9 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**

as close to real world as possible we will use the Rank-1 accuracy 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 , we test the 4 approaches of capturing the feature embeddings and saving them into a database.

Overall, the tests on CityFlow video tracks show that capturing vehicles in distinct zones improves both accuracy and efficiency, as clearly illustrated in Table 11.7.

11.6 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.

11.7 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.

The findings demonstrate that while we successfully constructed an object re-identification pipeline by combining state-of-the-art methods, its effectiveness and constraints are highly dependent on the specific application scenario. Re-identification performance is influenced by factors such as the suitability of training data for real-world generalization, the model training approach, and the careful tuning of hyperparameters. For vehicle re-identification, it is crucial to consider where and when the vehicles are detected, the characteristics of the road or intersection, and the specific features of the camera recording the footage. From this, we can conclude that achieving high performance in vehicle re-identification requires not only advanced methodologies but also scenario-specific adaptations in data preparation, model optimization, and detection configuration. We also conclude that obtaining multiple embeddings of the same object in different poses and locations help re-identifying it later.

Acknowledgements

This work was supported by Chips Joint Undertaking EdgeAI project. The project EdgeAI "Edge AI Technologies for Optimised Performance Embedded Processing" is supported by the Chips Joint Undertaking and its members including top-up funding by Austria, Belgium, France, Greece, Italy, Latvia, Netherlands, and Norway under grant agreement No 101097300.

References

[1] S. J. Prince, *Computer vision: models, learning, and inference.* Cambridge University Press, 2012.

[2] D. Lu and Q. Weng, "A survey of image classification methods and techniques for improving classification performance," *International Journal of Remote Sensing*, vol. 28, no. 5, pp. 823–870, Mar. 2007, doi:https://doi.org/10.1080/01431160600746456.

[3] J. Du, "Understanding of Object Detection Based on CNN Family and YOLO," *Journal of Physics: Conference Series*, vol. 1004, p. 012029, Apr. 2018, doi:https://doi.org/10.1088/1742-6596/1004/1/012029.

[4] X. Li and Z. Zhou, "Object Re-Identification Based on Deep Learning," *IntechOpen eBooks*, Jul. 2019, doi:https://doi.org/10.5772/intechopen.86564.

[5] R. Kuma, E. Weill, P. Sriram, and F. Aghdasi, "Vehicle re-identification: an efficient baseline using triplet embedding," presented at the 2019 International Joint Conference on Neural Networks (IJCNN), IEEE, Jul. 2019, pp. 1–9.

[6] 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, May 2019, doi: https://doi.org/10.3390/s19092048.

[7] C. Wang, Y. Yang, M. Qi, and H. Ma, "Efficient Cloud-edge Collaborative Inference for Object Re-identification," *arXiv preprint*, vol. arXiv:2401.02041, 2024.

[8] K. Cao, Y. Liu, G. Meng, and Q. Sun, "An Overview on Edge Computing Research," *IEEE Access*, vol. 8, no. 1, pp. 85714–85728, 2020, doi: https://doi.org/10.1109/access.2020.2991734.

[9] "YOLOv8 Ultralytics: State-of-the-Art YOLO Models," *learnopencv.com*, Jan. 10, 2023. https://learnopencv.com/ultralytics-yolov8/#YOLOv8-vs-YOLOv5 (accessed Sep. 23, 2024).

[10] A. O. Salau and S. Jain, "Feature Extraction: A Survey of the Types, Techniques, Applications," *IEEE Xplore*, Mar. 01, 2019. https://ieeexplore.ieee.org/abstract/document/8938371 (accessed Sep. 23, 2024).

[11] Z. Zheng, X. Yang, Z. Yu, L. Zheng, Y. Yang, and J. Kautz, "Joint Discriminative and Generative Learning for Person Re-identification." Accessed: Jan. 10, 2025. [Online]. Available: https://openaccess.thecvf.com/content_CVPR_2019/papers/Zheng_Joint_Discriminative_and_Generative_Learning_for_Person_Re-Identification_CVPR_2019_paper.pdf

[12] E. Almeida, B. Silva, and J. Batista, "Strength in Diversity: Multi-Branch Representation Learning for Vehicle Re-Identification," *arXiv (Cornell University)*, Jan. 2023, doi: https://doi.org/10.48550/arxiv.2310.01129.

[13] S. V. Huynh, N. H. Nguyen, N. T. Nguyen, Vinh Tq. Nguyen, C. Huynh, and C. Nguyen, "A Strong Baseline for Vehicle Re-Identification," *arXiv (Cornell University)*, Jun. 2021, doi: https://doi.org/10.1109/cvprw53098.2021.00468.

[14] Z. Zheng, T. Ruan, Y. Wei, and Y. Yang, "VehicleNet: Learning Robust Feature Representation for Vehicle Re-identification," *Computer Vision and Pattern Recognition*, pp. 1–4, Jan. 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*, vol. 14, no. 1, pp. 1–20, Dec. 2017, doi:https://doi.org/10.1145/3159171.

[16] Z. Zheng, T. Ruan, Y. Wei, Y. Yang, and T. Mei, "VehicleNet: Learning Robust Visual Representation for Vehicle Re-identification," Apr. 2020.

[17] "Papers with Code - Vehicle Re-Identification," *Paperswithcode.com*, 2022. https://paperswithcode.com/task/vehicle-re-identification#benchmarks (accessed Sept. 23, 2024).

[18] layumi, "GitHub - layumi/Person_reID_baseline_pytorch: Pytorch ReID: A tiny, friendly, strong pytorch implement of person re-id / vehicle re-id baseline.," *GitHub*, Jul. 25, 2022. https://github.com/layumi/Person_reID_baseline_pytorch/ (accessed Sept. 24, 2024).

[19] H. Liu, Y. Tian, Y. Wang, L. Pang, and T. Huang, "Deep Relative Distance Learning: Tell the Difference between Similar Vehicles," *IEEE Xplore*, Jun. 01, 2016. https://ieeexplore.ieee.org/document/7780607

[20] X. Liu, W. Liu, T. Mei, and H. Ma, "A Deep Learning-Based Approach to Progressive Vehicle Re-identification for Urban Surveillance," *Computer Vision – ECCV 2016*, pp. 869–884, 2016, doi:https://doi.org/10.1007/978-3-319-46475-6_53.

[21] Z. Tang *et al.*, "CityFlow: A City-Scale Benchmark for Multi-Target Multi-Camera Vehicle Tracking and Re-Identification," *arXiv (Cornell University)*, Jun. 2019, doi:https://doi.org/10.1109/cvpr.2019.00900.

[22] Y. Yao, L. Zheng, X. Yang, Milind Naphade, and T. Gedeon, "Simulating Content Consistent Vehicle Datasets with Attribute Descent," *Lecture notes in computer science*, pp. 775–791, Jan. 2020, doi: https://doi.org/10.1007/978-3-030-58539-6_46.

[23] Y. Zhang *et al.*, "ByteTrack: Multi-object Tracking by Associating Every Detection Box," pp. 1–21, Oct. 2021, doi:https://doi.org/10.1007/978-3-031-20047-2_1.

[24] R. J. Bayardo, Y. Ma, and Ramakrishnan Srikant, "Scaling up all pairs similarity search," *The Web Conference*, May 2007, doi:https://doi.org/10.1145/1242572.1242591.

[25] T. Taipalus, "Vector database management systems: Fundamental concepts, use-cases, and current challenges," *Cognitive Systems Research*, vol. 85, p. 101216, Jun. 2024, doi:https://doi.org/10.1016/j.cogsys.2024.101216.

[26] 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).

[27] regob, "GitHub - regob/vehicle_reid: Vehicle Re-identification," GitHub, 2022. https://github.com/regob/vehicle_reid/tree/master(accessed Oct. 14, 2024).

[28] 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," *Advances in Intelligent Systems and Computing*, pp. 343–349, 2021, doi: https://doi.org/10.1007/978-3-030-68017-6_51.

[29] "AXIS P1427-LE Network Camera - Product support | Axis Communications," *Axis.com*, 2023. https://www.axis.com/products/axis-p1427-le/support (accessed: September 26, 2024).

[30] V. Shankar, R. Roelofs, H. Mania, A. Fang, B. Recht, and L. Schmidt, "Evaluating Machine Accuracy on ImageNet," *PMLR*, pp. 8634–8644, Nov. 2020, Accessed: Sept. 26, 2024. [Online]. Available: http://proceedings.mlr.press/v119/shankar20c.html?ref=https://githubhelp.com

[31] Mian Muhammad Talha, Hikmat Ullah Khan, S. Iqbal, M. Alghobiri, T. Iqbal, and M. Fayyaz, "Deep learning in news recommender systems: A comprehensive survey, challenges and future trends," *Neurocomputing*, vol. 562, pp. 126881–126881, Dec. 2023, doi: https://doi.org/10.1016/j.neucom.2023.126881.

[32] Jean-Charles Lamirel, Maha Ghribi, and Pascal Cuxac, "Unsupervised recall and precision measures: a step towards new efficient clustering quality indexes," Aug. 2010.