1.

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

Tomass Zutis, Anzelika Bureka, Janis Judvaitis, Peteris Račinskis, Janis Arents, Modris Greitans

Institute of Electronics and Computer science, Latvia

**Abstract**

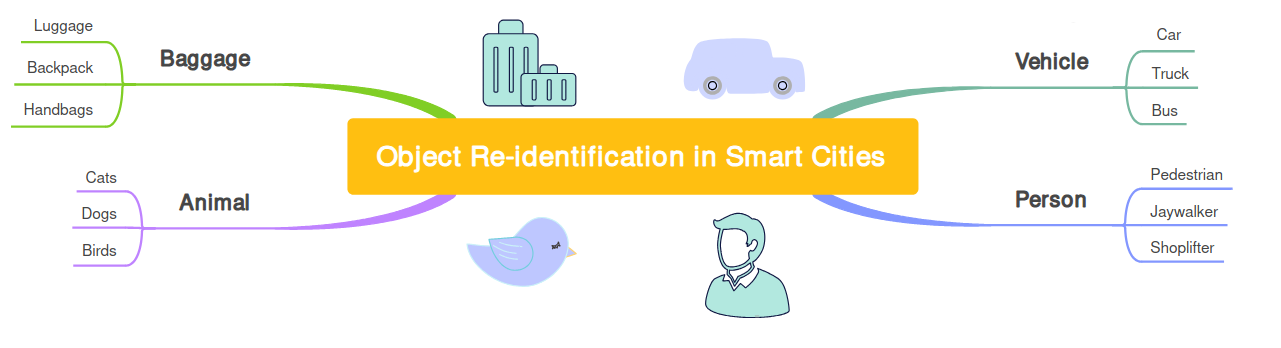
A method that leverages a multi-step process focused on extracting and using object features for object re-identification is described. The proposed pipeline includes steps such as: detecting an object, converting its features into a vector embedding, storing this embedding in a vector database, and 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 situations where re-identification needs to happen in outdoor environments or on-the-go, implementing this process on edge devices becomes crucial. Therefore, multiple 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 extractions, Vector embeddings, Neural networks, Edge devices, Vector databases, Traffic monitoring.

# Introduction

Monitoring and recognizing objects in photos and videos has long been a field of interest. It has become a well researched topic ever since computer vision has accelerated it’s possibilities, in the last decade especially [1]. With computer vision came image classification, which helped us classify images based on the object depicted. That gave us the answer to the question: ”What is the object in the image?” [2]. Later came great progress in object detection. That could answer the questions of: ”Where are the objects in the image?” and ”What are the objects in the image”. This was advanced by the Convolutional Neural Network (CNN) and its variants [3]. In this article, however, the authors are looking for a solution that could see objects of a single class and distinguish between them, using the specific features that exist only for a single individual in a class. This problem is called object Re-Identification - widely regarded as a sub-problem of image retrieval [4]. In imagery from at least two different cameras (scenes), object re-identification aims to correctly label the same individual after imaging conditions (like scene, lighting conditions, object pose and others) have changed. These could be images of vehicles, people or even animals or simple in-animate objects that have great intra-class variations (See fig. 1.1). We are also in a unique position where there is still a lot of research to be done on re-identification itself, however, 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. This is because some of the most common applications of re-identification are in traffic and security that use network cameras and cloud computing [5] [6]. Cities are looking for edge-computing devices using computer vision and deep neural networks to track real-time events in the public space. With the development of the Internet of Everything (IoE), the number of smart devices connected to the internet, 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 all this data manually and more crucially - in real time.

In the scenario of traffic monitoring, there is the need to re-identify vehicles on the go to: first, be able to track them in a road network and, second, model future traffic based on the existing patterns [5]. Edge computing proposes a hardware and software solution that would do this in real time – live video feed would be received from a network camera and the video analytics (i.e. detection and re-identification algorithms) are run directly on the edge device and only the results of the processing are transmitted.

Figure 1.1 There are multiple uses for object re-identification in the smart cities context. Some of them are people, vehicles, animals and even luggage in transit hubs.

The natural solution to the object re-identification problem in these circumstances is a pipeline consisting of algorithms designed to cover all the necessary steps for re-identification to work. We argue that constructing a pipeline that receives live video feed, processes the frames to extract objects from each frame, saves these objects into memory and recognizes the same objects in a different scene’s live video feed is possible and we aim to describe it in detail.

# Related work and state of the art

Object re-identification to this day is heavily reliant on extracting robust feature representations for the objects we are trying to save. Differences in lighting, angle, occlusions, multiple models of the same object (vehicle) are an obstacle in getting reliable predictions [8]. However, there is more to a working re-identification pipeline than just looking for the best way to produce vector embeddings. We have looked at the state of the art for multiple components of this problem, such ass: Object detection, Object re-identification and feature extractions, Person re-Identification, Vehicle re-identification, Available datasets, Synthetic datasets and Edge implementation.

## Object detection

Convolutional neural networks (CNNs) have been incredibly 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 (in the authors opinion v8 is the latest model with widespread and public support) has shown significant improvements in accuracy and speed [9], which is crucial for real-time applications like ours. YOLO v8 leverages the experience from the previous YOLO versions, having around 10% better mean average precision in object detection than the previously popular v5 (when comparing medium model size).

## Object feature extraction

Feature extraction is used to extract the most distinct features from an image, which is used to describe it. We try to save these features in a low dimensional vector space [10]. Before image feature extraction, multiple pre-processing stages are usually employed like normalization, thresholding, binarization, resizing and others. We can expect a model to extract: colour, texture, shape, motion and localization features. However, when dealing with one class objects with small intra-class variations specific types of features can be learned by the model like face features in the case of facial recognition.

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

## Vehicle re-identification

It is the case that in re-identification, there seems to be a better tailored model for each benchmark out there. For example, in the realm of Vehicle Re-ID benchmarks 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. However, we have chosen to 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. Even though it tops only one of them, it always keeps close to the top in terms of the 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 will further refer to this model of interest as "baseline model by Zheng *et al*.”

## Person re-identification

The goal of Person re-identification is to match a person's visual identity across many different cameras or scenes. The SOTA in person re-identification can also be judged by the benchmark leaderboards available on Papers with code [18]. Some of the most important datasets-turned-benchmarks are: the Market-1501, where the current best Rank-1 accuracy is [19]; the DukeMTMC-reID where the current best Rank-1 accuracy is [20]; and the CUHK03 where the current best Rank-1 accuracy is [19]. A modified version of the baseline model by Zheng *et. al.* has made an entry in the CUHK03 leaderboard 6th place in Rank-1 accuracy with 89.63% and 3rd place in Rank-5 accuracy with 99.01%.

## Available datasets

One of the amplest vehicle detection datasets - UA-DETRAC comes from Wen *et al.* [21]. The UA-DETRAC benchmark dataset consists of 100 challenging video sequences captured from real-world traffic scenes. It can be used for object detection model training or fine-tuning.

Some of the vehicle or person Re-ID datasets are already previously mentioned in the context of their usage in benchmarks. The VehicleID (PKU VehicleID) dataset has been introduced by Liu *et al.* in [22]. It includes 26'267 unique vehicles with 221'763 images in total. Each image is attached with an ID label corresponding to the vehicle identity. 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 introduced by Xinchen Liu *et al.* in [23] contains 49'357 images of 776 unique vehicles from 20 different cameras. Each image is attached with vehicle ID, bounding box, type, color, brand. The dataset makes up for its smaller set of unique id's compared to the VehicleID with better quality pictures, better visibility and vehicles from all angles not just back and front. The CityFlow dataset introduced by T Tang *et al.* in [24] is part of the AI City challenge and is a traffic camera dataset consisting of more than 3 hours of synchronized HD videos from 40 cameras. The dataset contains 229'680 images and around 700 unique vehicles. The 2020 AI City challenge [25] includes this dataset as videos, but also as cropped vehicle images for ReID model training. 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 is the types of vehicles seen, namely, US American versus Chinese, which creates a great parameter shift (See fig. 2.1).

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

## Synthetic datasets

When creating synthetic standalone datasets, there are risks learning features that do not generalize to real life images, however, when creating additions to already existing datasets, there is high risk of creating an appendix that has a great parameter shift from the rest of the dataset, thus causing problems with convergence.

The VehicleX synthetic dataset is created with these problems in mind. It contains generated images with domain adaptation from VehicleID, VeRi-776 and CityFlow datasets [26]. Each of these adapted versions can be paired with the original dataset to enhance learning results.

## Edge implementation

If we want to meet the requirements of a real-time system, we need to implement 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 in Liverpool, Australia, 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 describe the proposed sequence of processes that would enable object re-identification: Object detection, feature extraction and saving and querying for an object in a vector database. We will touch on how these components fit together and how each part could be improved to better serve the common outcome. We will also introduce the application of this methodology in a smart city environment - the Vehicle ReID pipeline.

## Object detection

When we have objects of interest included in images, but these images are not cropped to the dimensions of the object, we need to find the objects in the image and decide on the extent of their size. This is done with an object detection algorithm.

In theory, one could train or fine-tune their own object detection model to tailor for a specific object class. However, latest off the shelf detection models can detect vehicles or people, for example, with very high precision. As mentioned in the [2.1](#_Object_detection) section YOLO v8 model is regarded as very accurate and suitable for real-time applications.

We will be using the YOLO v8 model to detect the objects of a specific class, determine their bounding boxes and saving cropped out images of these object bounds.

## Feature extraction

Once we have cropped images of the objects, we need to extract their features and turn them into vector embeddings. This can be done with a re-identification model, that has learned to extract the features of specific objects and save them in an embedding. For this we will use the baseline model by Zheng *et al.* As described in section [2.3](#_Vehicle_re-identification￼), this baseline can be trained to fit vehicle, person or any other object re-identification. The model is implemented in Pytorch and can be readily customized. We will also 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. The model code is available and can be modified in any stage.

## Saving feature extractions

Once we have managed to extract features from such rich data like images, we need to find a way to store and compare the extractions in a computationally inexpensive way.

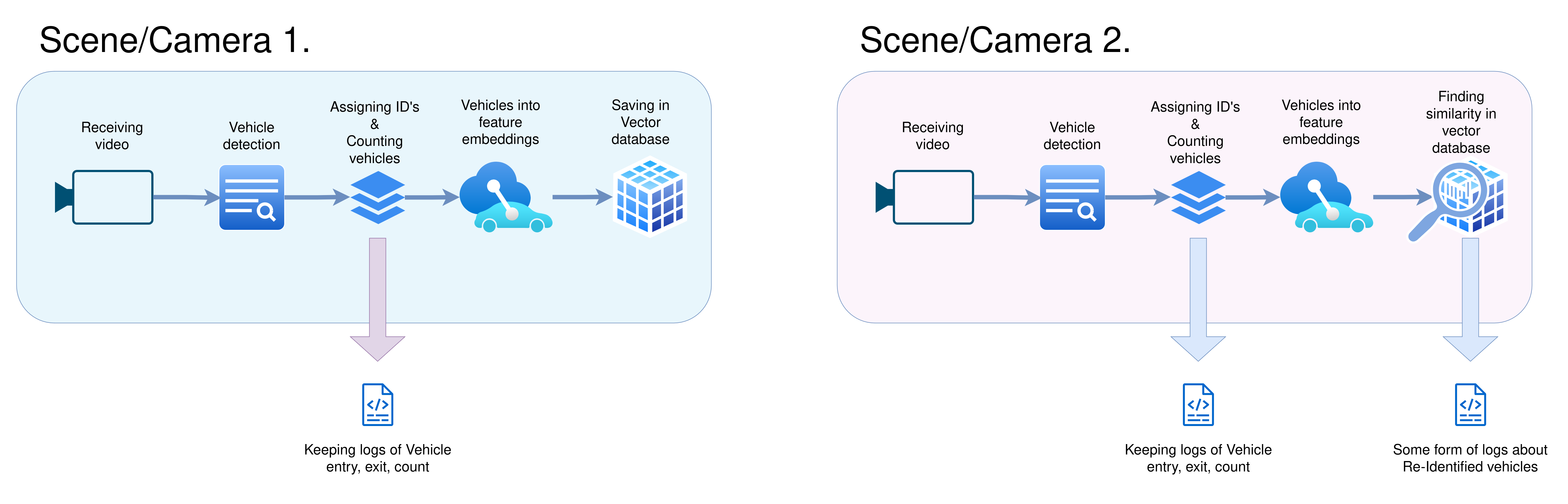
These features have been turned into n-dimensional vectors that consist of natural, real or complex numbers, where one number represents a feature or a part of a feature. We should be able to compare these vectors for their similarity, to be able to retrieve the most similar object to the one being queried for. We can do this by using the Cosine similarity function. Cosine similarity is widely used and requires an input of at least two unit-length normalized vector inputs to output a vector distance [28].

Efficient management of vector data would require a Vector database system, so vectors could be stored and queried as efficiently as possible [29]. This vector database will have to contain vector similarity search complemented by metadata filters. For a very simple interpretation of our task the database entries would simply be in the form of Key:Value = ObjectId:ObjectFeaturesVector while more fields for entries should be easy to add.

## The Vehicle ReID pipeline

Having outlined the general components of object re-identification—such as object detection, feature extraction, and storing embeddings—this subsection will introduce the methodology that is specific to a vehicle re-identification pipeline that is currently under development.

We propose a pipeline that would take in video frames from a network camera *x,* detect the vehicles in the frames, crop images of the vehicles and save them, turn these cropped images into feature embeddings and then save 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).

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

To make sure we are developing a functional pipeline and can monitor the performance of each step we need to split the process into two parts, at least during the development stage:

* **Vehicle counting and tracking**

The purpose of this process will be to count, track and save information about the vehicles that have passed through the view of a single camera. It should receive the videos from a network camera, detect the vehicles, start tracking them and establish the sectors where we count incoming and outgoing cars (for example entries and exits in an intersection). The count, entry and exit times of cars should be continuously logged.

* **Vehicle Re-Identification**

Once we have vehicles detected and tracked, we need to decide on an approach about where and how often images of vehicles should be cropped out of the frame. Then we need to save these images, create feature extractions and save them in a vector database.

We will also need to make sure that the methods that make up the pipeline are well tailored for our purpose. We introduce the following points of action:

### Establishing counting and tracking solutions

After receiving the video frames from the network camera, we need to use the YOLO v8 model to detect the vehicles and get their bounding boxes. Further we will use the ByteTrack tracking package [32] to assign IDs to the vehicles and track them through the following frames until they leave the view of the camera.

We believe this will allow the following ReID process to run smoothly. We also need to keep a reliable log for supplementing the accuracy of ReID and analysis of the traffic flow.

### Finding the best approach to save images of vehicles

To store and query feature embeddings, we have chosen LanceDB – an opensource database for vector search, built for efficiency in handling large-scale vector data, low-latency queries, and seamless integration with Python [33]. Its flexibility in saving and querying data—such as the incremental updating of embeddings—makes it ideal for our re-identification pipeline.

Additionally, we must come up with methods for deciding where and how often should vehicles be cropped out of the frame. A specific position in the camera-view might be more advantageous to save the vehicle and could produce a better feature vector. Complimentary to that, saving the vehicle in multiple positions in the same camera might make it more recognizable in a different camera.

### Training the ReID model on different datasets

We will try all the most widely used and publicized datasets (mentioned in [2.5](#_Available_datasets)), to see which fits best for our pipeline.

We will train our model separately on the VehicleID, VeRi-776 and CityFlow datasets and based on the results, decide which one fits best for the test data we have gathered from our network cameras. It is worth training a model on a combined Veri-776 and VehicleID dataset, because of their similarity and check whether it provides a better result.

We will also combine the best performing standalone dataset with the Vehicle X synthetic data. The Vehicle X data has three versions – each tailored to one of these datasets. We will add the tailored synthetic data to the train portion of the base dataset.

### Going over training hyper-parameters

It is valuable to consider the difference in model performance that hyper-parameters play during model training.

When reviewing the model's hyperparameters during training, it's observed that the author focused primarily on parameters such as:

* Backbone,
* Learning rate,
* Warm epochs,
* Batch size, and
* Erasing probability,

but didn’t mention testing or the impact of other crucial parameters, like:

* Colour jitter,
* Size of the final linear layer (i.e., length of output),
* Cosine learning rate or
* Stride

These parameters, though not emphasized in the published work by Zheng *et al* and associated GitHub repository, could in theory influence the model’s performance and warrants experimentation.

## Testing the proposed methodology

Since the pipeline is in development, we will first need to conduct a series of tests for individual components and use cases.

* **Testing on a dataset**

We need to test the model itself on a dataset to see the accuracy with which the model makes predictions in a very controlled environment such as cropped pictures of vehicles. This includes testing on benchmark datasets and a test dataset with vehicles from our network cameras.

* **Testing on video tracks**

To test all the components of the re-identification process part of the pipeline we might choose a pre-recorded scene of traffic that has ground truth available. We could skip the object detection and tracking part, hence the ground truth would let the pipeline know where the cars in the frame are and what is their pre-assigned ID. In this case only the bounding box cropping, feature extraction, saving and querying the database could be tested.

Before testing the re-identification functionality, it is important to test the vehicle counting step to make sure the process is accurate. This, however, falls slightly outside of the scope of the paper, so we have purposefully chosen to leave these sections out.

At the end of successful testing of all steps of the pipeline, one should create a video test scenario that starts with object detection and does not count on the ground truth bounding box and pre-assigned vehicle id data, encompassing both: 1.) vehicle counting and tracking and 2.) vehicle re-identification functionalities.

## 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

While the general steps remain consistent with the proposed methodology, the experimental settings section will detail the practical set-up of mentioned concepts, fitted specifically to the use-cases of vehicle ReID in a smart city setting. We will describe the settings for the Vehicle counting and tracking part of the pipeline, Vehicle Re-Identification part of the pipeline, the available datasets used for training, synthetic data used to augment the datasets, going over hyper-parameters, creating and executing tests.

## Vehicle counting and tracking

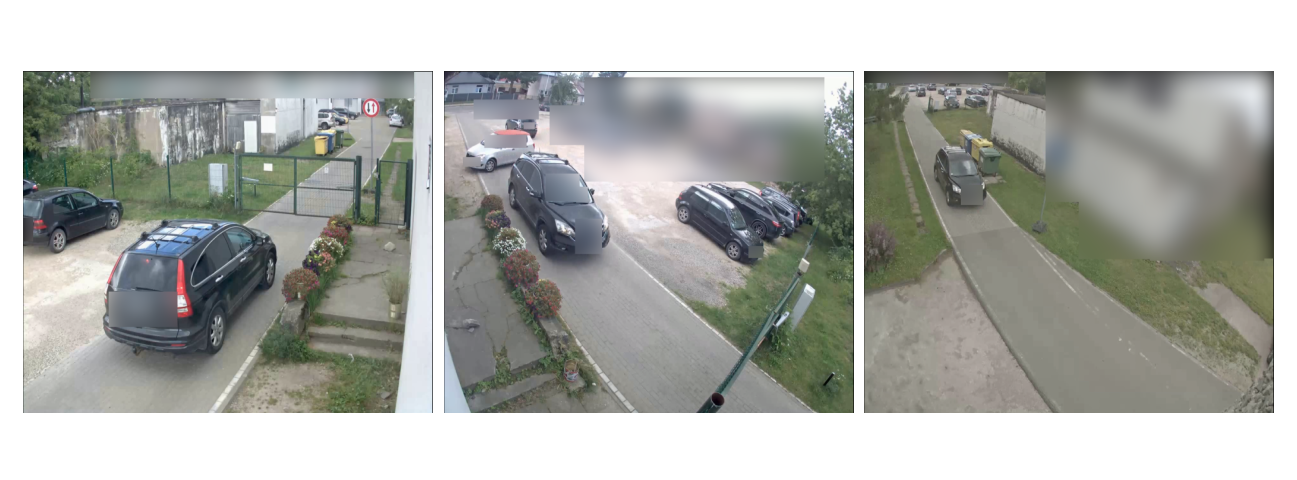
In this sub-section we will elaborate on the implementation of the counting and tracking part of the pipeline

### Receiving video from a Network camera

To simulate a traffic scenario for tracking and ReID applications, 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 over RTSP, however, HTTP works as well. A 25-30% compression rate is employed as well for better data transmission.

The view from the cameras (See fig. 4.1) can be summarized in the following way:



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

* **Camera 1.**

This camera sees cars entering and exiting the inner service road from the parking lot from outside of the territory. Gates are also visible. It is located about 3 to 4 meters above ground and generally allows us to see both the front/back of the car and the sides of the car. For non-cargo vehicles, the roof is also visible.

* **Camera 2.**

This camera is located above the gates and can see into the parking lot. We can see the opposite side of the car as opposed to camera 1. It is also located about 3 to 4 meters above ground, however, cars are seen more from the front and the back, but skewed sides and roof are visible.

* **Camera 3.**

This camera is located deeper into the territory and generally sees the whole service road with the parking lot in a far distance. The camera is much higher than 1. and 2. and is about 6 to 8 meters above ground. We can see the front/back and roof of the car well, however, the sides are not visible in such quality because of the altitude and angle of the camera.

### Object detection, counting and tracking

With ByteTrack (see section [3.4.1](#_Establishing_counting_and)) we now know when the vehicles have left or entered specific parts of the scene. We can use this to count the cars, by using the built in functions of Bytetrack, for example, *drawLine* - counting when a car drives past a drawn line (See Fig. 4.1).



**Figure 4.1** The implementation of the counting lines and ByteTrack in cameras (from the left, upper row) 1.,3. and (lower row) 2.

In our case, however, modifications to the package had to be made, because the built-in function counted a car once it had fully been seen before the drawn line and subsequently completely after the line. In our cameras, however, the field of view is slightly limited when vehicles approach close to the gate, so we had to include the following provisions:

* A vehicle is counted also if they were first detected when already on the counting line (connection or resolution problems)
* A vehicle is counted also when it is no longer detected, but didn’t cross the line fully (most of the vehicle's area crossed the line before disappearing – can happen, when vehicle is large and is close to the camera)
* A vehicle was neither fully detected before or after the line, but switched location while existing on the line, by having most of its area shift from one side to the other (can happen if there are disruptions in the video).

These can also be important positions for cropping of the vehicles out of the frame. We could crop a few images of the vehicle even before it has been counted and in case it doesn't cross the line – we discard the images we would have otherwise saved.

### Modes of testing

As previously mentioned in section [3.5](#_Testing_the_proposed) - the vehicle counting step itself falls shortly out of the scope of the paper. In general, however, testing the counting accuracy could be done by recording videos from our network cameras or using publicly available video tracks of vehicles. When recording test footage from our own network cameras, most of the testing had to be done manually i.e. by visually inspecting the footage, as there is no ground truth available.

## Vehicle re-identification

In this sub-section we will elaborate on the experimental settings that enable re-identification in the pipeline. As mentioned in section [2.3](#_Vehicle_re-identification￼) we have chosen the baseline model by Zheng *et al*. as our re-identification model and will further work to integrate this model into our pipeline.

### Testing and data annotation

First and foremost, we should establish tests and create test data, that will let us evaluate the experiments. We will use a combination of both publicly available datasets and recorded test data from our network cameras:

* **Benchmark datasets.** By using benchmark datasets, such as Veri-776, we can access reliable test data, standardize our test metrics and evaluate the re-identification model itself.
* **CityFlow test track video.** By accessing the videos included in the CityFlow test tracks, we can test the whole re-identification part of the pipeline and simulate a real-world scenario where we are re-identifying vehicles in an intersection. 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. 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.

### Going over hyper-parameters

As mentioned in [3.4.4](#_Going_over_model) we will train the model with variant values for a group of hyper-parameters.

We can choose to evaluate the following variants of:

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).

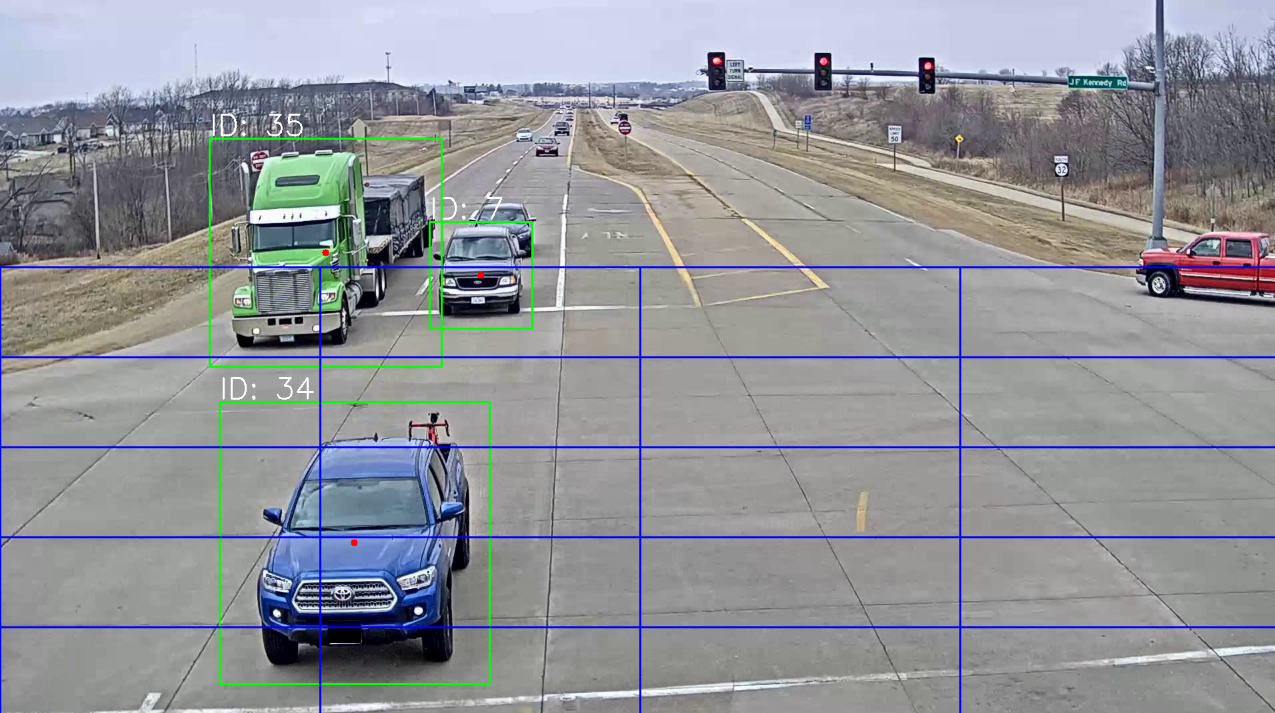
### Saving the feature extractions

We will experiment with four methods (See table 4.1) 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 4.1 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 and approaches we have described earlier in the paper, such as: Finding the data that generalizes best for our custom data and vehicles, Training the model on different datasets, supplementing the training with synthetic data, changing the prediction head during inference, going over hyperparameters, testing the performance of the best model on the data from our network cameras and testing the whole re-identification part of the pipeline on the CityFlow video test track.

## Performance metrics used

* **Rank-1, Rank-X Accuracy**: Measures how often the correct match appears as the top-ranked result (Rank-1) or within the top X results (Rank-X) in a retrieval task.
* **mAP (Mean Average Precision)**: The mean of the average precision scores for all queries, reflecting how well the model ranks true positives across the entire dataset.
* **Micro Precision and Recall**: Calculated by aggregating true positives, false positives, and false negatives across all classes to provide a single precision and recall score.
* **Macro Precision and Recall**: Precision and recall calculated separately for each class and then averaged, giving equal weight to all classes regardless of their size.
* **Validation Loss**: A measure of how well a model is learning during validation step of training, typically the difference between the predicted and actual values, minimized during the training process.

## Finding the dataset that generalizes best on our custom data

As stated in [3.4.3](#_Training_the_ReID) we aim to find the dataset, that makes the model generalize best on our own recorded data. For this we will use the Rank-1 accuracy, that has been averaged from 3 tests on our custom dataset.

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 |

We can see that simply training our baseline model on the Veri dataset for 19. Epochs has brought the best result, when applying the model on our custom dataset. Even though the difference between VeRi and Cityflow datasets may be called unsignificant, visually the vehicles seen in the Veri dataset seem to resemble our collected data more.

## Comparing baseline model to VeRi-776 benchmark

Now that we know that the VeRi dataset is most like our real-world data, we should choose the VeRi-776 benchmark to compare our findings.

Even though we use the VehicleNet Paper [14] reported results as a benchmark, we do not directly compare them with the baseline model by Zheng *et. al.* used by us. As clarified previously - the baseline model by Zheng *et. al.* is a general object re-identification baseline, that is open source and customizable, however, in [14] Zheng *et. al.* makes further modifications to this concept, creating a model that is specific to vehicle-re-identification and raises accuracy percentages by creating their custom dataset and employing multiple tiers of training and specialization for the benchmark.

Table 5.2 VeRi Trained Model comparisons on the VeRi-776 benchmark

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Rank-1** | **Rank-5** | **Rank-10** | **mAP** |
| VehicleNet reported results | *96.78* | - | - | *83.41* |
| Baseline model (Trained on Veri, 19th Epoch) | 93.80 | **97.61** | 98.74 | 68.96 |
| Baseline model (Trained on Veri, 49th Epoch) | **94.04** | 97.49 | **98.80** | **69.4** |
| Baseline model (Trained on Veri + Vehicle-ID, 19th Epoch) | 93.44 | 96.90 | 98.68 | 69.29 |
|  |  |  |  |  |

We can see that our chosen model performs best on the benchmark, when trained on the Veri dataset for 49 epochs. Even though there is great similarity between VeRi and Vehicle-ID datasets, their combined dataset does not yield greater results than the model trained simply on VeRi.

## Supplementing dataset with VehicleX synthetic data

We further test if model performance on the benchmark can be improved by adding the VehicleX synthetic data to the VeRi dataset.

Table 5.3 Model performance with and without VehicleX synthetic data

|  |  |  |
| --- | --- | --- |
| **Metric** | **Training on VeRi** | **Training on VeRi + VehicleX** |
| Rank1, % | 94.04 | **94.15** |
| Rank5, % | 97.49 | **97.85** |
| Rank10, % | **98.80** | 98.74 |
| mAP, % | 69.4 | **71.64** |

Training with the synthetic data has improved the model's performance, most notably the mean average precision – by 2.24%.

## Going over hyper-parameters

After discussing the hyper-parameters in [4.2.3](#_Going_over_hyper-parameters) we have trained the model on all combinations of the parameters of interest over 15. epochs and recorded their loss values. Training the model for 36 times in total, consisting of 540 epochs. The 10 entries with the lowest validation loss value we showcase in table 5.5.

Table 5.5 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 the model generalises best to validation data with a Stride of 1, Number of linear layers of 256 and Color Jitter enabled.

Table 5.6 Verifying improvement from updated training parameters on the VeRi-776 benchmark

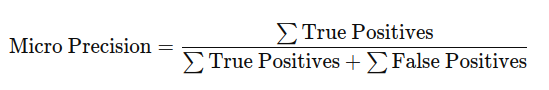
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Rank-1** | **Rank-5** | **Rank-10** | **mAP** |
| Baseline model trained on default parameters (19. Epoch) | 93.80 | 97.61 | **98.74** | 68.96 |
| Baseline model trained on updated parameters (19. Epoch) | **95.47** | **97.62** | 98.51 | **71.70** |
| Baseline model trained with synthetic data on default parameters (39. Epoch) | 94.15 | 97.85 | 98.74 | 71.64 |
| Baseline model trained with synthetic data on updated parameters (39. Epoch) | **95.83** | **98.09** | **98.87** | **73.64** |

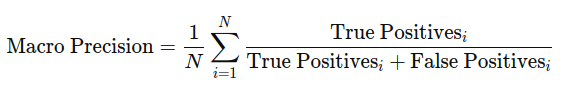
As seen in table 5.6 the updated parameters demonstrate improvement on the benchmark, both for the model trained on VeRi dataset for 19. epochs and the model trained with the VehicleX augmented VeRi dataset (our best trained model so far) for 39. Epochs.

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

Now that we’ve established that the Baseline model trained with synthetic data on updated parameters is offering the best results, we can test the model performance on the test data gathered from our network cameras.

### Precision

We create table 5.7 with the Micro and Macro precision values, when re-identifying from a query camera *x* (rows) to gallery camera *y* (columns).(5.1)

(5.2)

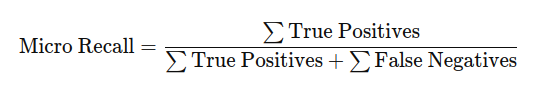
Where *N* is the total number of classes

Table 5.7 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 |  |  |

### Recall

We create table 5.7 with the Micro and Macro recall values, when re-identifying from a query camera *x* (rows) to gallery camera *y* (columns).

(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 5.7 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 |  |  |

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

* The model generalizes reasonably well to our custom testing data
* The best re-identification occurs between camera pairs 2. and 3. and the worst – between 1. and 2.
* The worst re-identification scenario was camera 2. images queried from camera 1. gallery and the best – camera 2. images queried in camera 3. gallery.
* Images from camera 3. had the overall best results when querying results and images from camera 3. were also best at being the gallery for queries.
* The highest difference between Macro and Micro values were in camera 1. gallery images, which suggests that there were vehicle identities with a smaller class size, that the model failed to re-identify or that well re-identified identities were overrepresented numerically.
* In the case of camera 2 to camera 1 re-identification, the precision values (micro: 66.13, macro: 57.58) show that the model performs reasonably well at identifying correct matches (precision), but the drop in macro precision suggests that some classes (vehicle identities) are underrepresented or harder to recognize.
* Furthermore, for camera 2 to camera 1 re-identification the recall values (micro: 66.13, macro: 46.22) indicate the model failing to retrieve many true matches, particularly for underrepresented vehicle identities. Specifically, the model may be overfitting to certain well-represented vehicle identities in camera 2 or reflects the model's difficulty in recognizing rare vehicles.
* 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 5.8 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 |

Table 5.8 shows that we have been able to even out the results meaning that the 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 will take the best trained model we have and test it 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 simply 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. Here it’s practically the same measure as Micro Precision and Recall.

As mentioned in section [4.2.4](#_Saving_the_feature) , we test the 4 approaches of capturing the feature embeddings and saving them into a database.

Table 5.9 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** |

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

# Future research

Although the paper proposes a complete set of methods to construct an object re-identification pipeline on the edge, there are still multiple areas for future research to be done, such as: optimizing the pipeline for a specific edge device such as Nvidia Jetson, finetuning models trained on the public datasets with our in-house datasets and a deeper analysis on the content of feature vectors.

Future research should focus on optimizing the re-identification pipeline for specific edge devices, such as the Nvidia Jetson. This could involve acquiring the device, profiling the pipeline's performance, and optimizing resource-heavy components like model inference and data handling through techniques like model pruning, quantization, or using TensorRT for accelerated inference.

Another area for future work involves fine-tuning the model, initially trained on the VeRi public dataset, with our in-house dataset captured from network cameras. This fine-tuning would reduce the distribution shift between the public dataset and real-world deployment data, enabling the model to better adapt to practical application settings.

Lastly, a more in-depth analysis of feature vectors is necessary to identify which components are relevant for re-identification and which are redundant. By understanding the structure and content of these vectors, we can refine the feature extraction process and potentially enhance the accuracy of the pipeline.

# 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. ——, “Person re-identification benchmarks,” <https://paperswithcode.com/task/person-re-identification>, accessed: September 24, 2024.
19. C. Sharma, S. R. Kapil, and D. Chapman, “Person re-identification with a locally aware transformer,” arXiv preprint arXiv:2106.03720, 2021.
20. M. Wieczorek, B. Rychalska, and J. Dabrowski, “On the unreasonable effectiveness of centroids in image retrieval,” in Neural Information Processing: 28th International Conference, ICONIP 2021, Sanur, Bali, Indonesia, December 8–12, 2021, Proceedings, Part IV 28. Springer, 2021, pp. 212–223.
21. L. Wen, D. Du, Z. Cai, Z. Lei, M.-C. Chang, H. Qi, J. Lim, M.-H. Yang, and S. Lyu, “Ua-detrac: A new benchmark and protocol for multi-object detection and tracking,” Computer Vision and Image Understanding, vol. 193, p. 102907, 2020.
22. 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.
23. 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.
24. 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.
25. AI City Challenge, “2020 data and evaluation,” https://www.aicitychallenge.org/2020-data-and-evaluation/, 2020, accessed: September 24, 2024.
26. 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.
27. layumi, “Pytorch reid github repository,” https://github.com/layumi/Person reID baseline pytorch/, accessed: September 24, 2024.
28. 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.
29. T. Taipalus, “Vector database management systems: Fundamental concepts, use-cases, and current challenges,” Cognitive Systems Research, vol. 85, p. 101216, 2024.
30. 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.
31. AXIS communications, “Product support for axis p1427-le network camera,” <https://www.axis.com/products/axis-p1427-le/support>, accessed: September 26, 2024.
32. 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>.
33. 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).