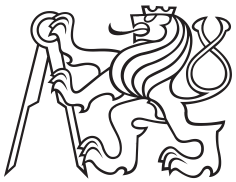
Master's Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

# Train Car Image Hashing Using Deep Networks

Bc. Konstantin Khokhlov

Supervisor: Mgr. Ondřej Pacovský
Field of study: Cybernetics and Robotics
January 2023

# MASTER‘S THESIS ASSIGNMENT

## I. Personal and study details

Student's name: **Khokhlov Konstantin** Personal ID number: **474681**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

Branch of study: **Cybernetics and Robotics**

## II. Master’s thesis details

Master’s thesis title in English:

**Train Car Image Hashing Using Deep Networks**

Master’s thesis title in Czech:

**Hashování obrázků vlakových vozů pomocí hlubokých neuronových sítí**

Guidelines:

The project aims to implement a system representing train car images with semantic hash codes. Extracted codes will be used to identify train profiles by matching images of the same cars captured from different locations.
1. Review current state-of-the-art in deep instance-level hashing with supervised training.
2. Propose an appropriate system architecture.
3. Gather training and validation datasets. Implement tools for collecting image pairs of objects of the same instance for supervised learning.
4. Implement selected architecture and train the system on the collected training dataset.
5. Experimentally evaluate the accuracy and inference speed of the implemented hashing method.

Bibliography / sources:

[1] Liu, Haomiao and Wang, Ruiping and Shan, Shiguang and Chen, Xilin: “Deep Supervised Hashing for Fast Image Retrieval”, CVPR, 2016.
[2] Luo, Xiao, et al. "A survey on deep hashing methods.", TKDD, 2020.
[3] Hoe, Jiun Tian, et al: “One Loss for All: Deep Hashing with a Single Cosine Similarity based Learning Objective”, NeurIPS, 2021.
[4] Yuan, Li, et al. "Central similarity quantization for efficient image and video retrieval." CVPR, 2020.

Name and workplace of master’s thesis supervisor:

**Mgr. Ondřej Pacovský Eyen SE, Prague**

Name and workplace of second master’s thesis supervisor or consultant:

**prof. Ing. Tomáš Svoboda, Ph.D. Vision for Robotics and Autonomous Systems FEE**

Date of master’s thesis assignment: **15.08.2022** Deadline for master's thesis submission: **10.01.2023**

Assignment valid until: **19.02.2024**

| Mgr. Ondřej Pacovský | prof. Ing. Tomáš Svoboda, Ph.D. | prof. Mgr. Petr Páta, Ph.D. |
|---|---|---|
| Supervisor’s signature | Head of department’s signature | Dean’s signature |

## III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student’s signature

# Acknowledgements

I wish to express my sincere thanks to my supervisor Mgr. Ondřej Pacovský for consultation and the guidance he provided me with this thesis.

I want to thank prof. Ing. Tomáš Svoboda, Ph.D., for providing help with a formal part of the work.

I am also deeply grateful to my family, friends, and girlfriend for their unwavering support and encouragement throughout the study and during the completion of this work.


# Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, 10. January 2023

# Abstract

This thesis aims to create a system representing train car images using semantic hash codes. These codes can be used to match images of the same cars captured from different locations and times. The study begins by examining existing algorithms in image representation. The work focuses on the 2-staged CNN-based method, in which a continuous representation of the images is learned first and then transformed into a binary form using a trained hashing layer. The custom datasets for supervised learning were collected without human annotation. Within the testing, the work compares several integrated neural network architectures, hashing methods, and learning settings in terms of their search-based accuracy, memory footprint, and inference speed. It also evaluates the strengths and weaknesses of the proposed method and examines the differences between continuous and binary image representation. The results of this project show that the proposed method achieves high performance on the collected datasets and can be applied in a real industrial system.



**Keywords:** image representation, image hashing, instance-level retrieval, CNN

**Supervisor:** Mgr. Ondřej Pacovský



# Abstrakt

Cílem této práce je vytvořit systém reprezentující obrazy vlakových vozů pomocí sémantických hash kódů. Tyto kódy lze použít k porovnávání snímků stejných vozů zachycených z různých míst a v různé časy. Studie začíná zkoumáním existujících algoritmů v reprezentaci obrazu. Práce se zaměřuje na 2-stupňovou metodu založenou na CNN, ve které se nejprve naučí spojitá reprezentace obrazu a poté se pomocí natrénované hashovací vrstvy převede do binární podoby. Vlastní datové sady pro řízené učení byly shromážděny bez lidské anotace. V rámci testování práce porovnává několik architektur integrovaných neuronových sítí, hašovací metody a nastavení učení z hlediska jejich přesnosti, obsazené paměti a rychlosti. Dále práce hodnotí silné a slabé stránky navrhované metody a zkoumá rozdíly mezi spojitou a binární reprezentací obrazu. Výsledky tohoto projektu ukazují, že navrhovaná metoda dosahuje vysokého výkonu na shromážděných souborech dat a lze ji aplikovat v reálném průmyslovém systému.



**Klíčová slova:** reprezentace obrázku, hashování obrázků, vyhledávání obrázku, CNN

**Překlad názvu:** Hashování obrázků vlakových vozů pomocí hlubokých neuronových sítí

# Contents

# Figures

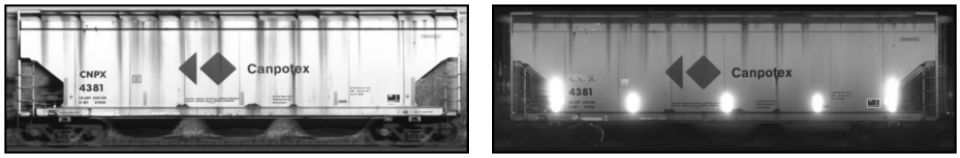
# Tables

# Chapter 1

# Introduction

This work has been carried out in cooperation with Railstate LLC [1]. The company needs a method to match images of the train cars captured from different locations and times with low storage requirements. Hence, the goal of this thesis is to design and implement a system for representing train car images using compact semantic hash codes. These feature codes will serve as a concise and unique representation of the train cars. Like "fingerprints," they can be used to identify and match images of the same car, regardless of when and where the images were taken.

The task has several challenges. Images of different cars may look very similar and be hardly distinguished, even for a human (see fig. 1.1); on the other hand, the images of the same car captured from different locations can be visually dissimilar. The images can be captured in completely




**(a) :** Example of images of the same car captured from different locations during the day (left) and night (right).


**(b) :** Images of different cars that are visually similar to 1.1a. The reader may recognize different identification numbers of the cars.

**Figure 1.1:** Examples of similar train car images.

different lighting (day/night) or weather (snow/rain) conditions. Since

[1] https://www.railstate.com/

images are taken from different locations, the contrast, brightness, and car placement may change significantly. The designed system should be real-time, meaning that it will be able to process images from one train with around 150 cars in terms of several minutes. To keep memory requirements low, the image representation must be compact, with the representation of one train taking up only tens of kilobytes and the representation of one car being less than 1 kilobyte. Note that the task can be resolved with the detection of unique identification numbers. However, those numbers are quite often not readable in case they are obscured, blurred, or corrupted.

To achieve this goal, the research conducted in this thesis first examines and evaluates existing algorithms in the fields of image representation, instance-level retrieval, and image hashing. Based on this review, the study selects a two-stage CNN-based method for generating semantic hash codes. The first stage involves learning a continuous representation, while the second stage involves transforming this representation into a binary form using a trained hashing layer. For the supervised learning algorithms, there was implemented the dataset preparation tool that collects labeled data without human annotation.

The study compares several integrated neural network architectures, hashing methods, and learning settings. The comparison is based on several key metrics, including search-based accuracy, memory footprint, and inference speed. The results of this comparison are then used to assess the strengths and weaknesses of the proposed method, as well as the differences between continuous and binary image representation. Ultimately, the results of this project demonstrate that the proposed method is highly effective on the collected datasets and has the potential to be applied in the real-world industrial system.

# Chapter 2

# Theoretical Background

## 2.1 Problem Description

Many computer vision tasks do not operate with raw images. The high dimensionality of images makes it difficult to process the entire image; moreover, much of the information contained within it is unnecessary for the task. Therefore, rather than using the entire picture, only essential information is extracted. The process of getting a relevant image representation is called feature extraction.

This project aims to create a robust real-time image representation model that will minimize the difference between image hashes of the same train cars and, at the same time, minimize the similarity between images of the different cars. The representation should be invariant to lighting and weather conditions, changes in contrast and brightness, and car placement. The goal is also to minimize the memory footprint of extracted features so that it is possible to store and compare those features on a large scale.

### 2.1.1 Image Retrieval Pipelines

One challenging task that requires good image representation is Content-Based Image Retrieval (CBIR). Therefore, the methods designed to resolve CBIR problem, especially instance-level image retrieval, are studied in this work.

CBIR is a technology that enables users to search for and retrieve images from a large database based on the visual content of the images. The structure of a CBIR pipeline is shown in fig. 2.1. First, features are extracted from all images from the dataset using the descriptor model. Once extracted features are extracted, they are indexed in a database or search engine to enable efficient search and retrieval. This step is usually done offline,

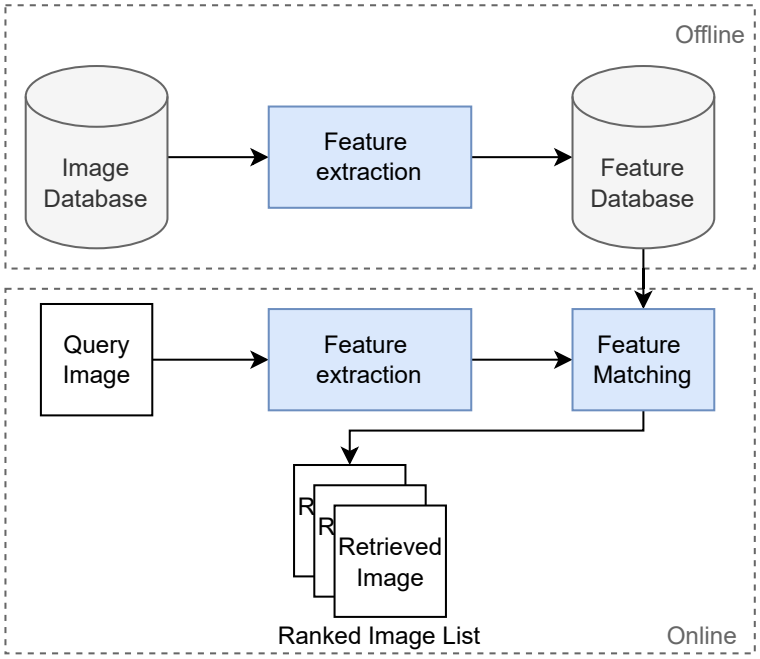



**Figure 2.1:** Schematic diagram of a CBIR pipeline. The features of dataset images are pre-computed offline. After the query image is provided, the list of potentially matching images, ranked by descriptors similarity, is returned.

meaning the descriptor database is already pre-computed when the query is passed to the system. When a user initializes a search, the features are extracted from the query image, and the matching algorithm is applied to find the most similar images. Ideally, the system should retrieve all the matching images or the ranking list, where the images are ranked by feature similarity. In instance-level retrieval, the images are searched to match a specific object in a query image.

## 2.2 Survey on Sate-Of-The-Art

### 2.2.1 Continuous Representation

Image features can typically be classified as either local, which is related to specific, small regions within the image, or global, which describe the entire image as a whole The first significant advances in image representation were achieved by handcrafted local descriptors like SIFT [Low99] with the Bag of visual Words (BoW) [SZ03] method.

After the success of AlexNet architecture [KSH12], the most advanced and accurate methods for image retrieval are based on Convolutional Neural Networks (CNNs) that can learn powerful global feature representations

with multiple levels of abstraction directly from data [CLW+22]. Features extracted by CNN layers were then transformed to compact vectors by fully-connected layers, or a variety of pooling layers (MAC [TSJ15], SpoC [BL15], GeM [RTC18]). The first methods applied CNN-based models pre-trained on the classification datasets like ImageNet [DDS+09]. However, the pre-trained networks are influenced by domain shifts when performing retrieval tasks on new datasets. Thus later works [BSCL14], [RTC18] showed the importance of the fine-tuning of deep networks to the specific domain.

The more recent approaches incorporate both global and local descriptors. A common image retrieval system setup is to search by global features first, then re-rank the top database images using local feature matching. Most of those systems need to use different models and extract each separately, leading to high memory usage and increased latency. Therefore, the authors of [CAS20] presented DELG - unified model to represent both local and global features that achieves state-of-the-art performance on the Revisited Oxford, Revisited Paris [RIT+18] and Google Landmarks v2 datasets [WACS20]. However, storing DELG local features takes an excessive amount of memory, which is unsuitable for this work. Therefore, this project focuses only on global descriptors, mainly on CNN-based models with GeM pooling [RTC18], since it aims at image representation only and not on image retrieval.

### 2.2.2 Binary Representation

A fundamental building block of a large-scale image retrieval system is hashing. The objective of image hashing is to represent features using a binary code to make it yet more compact than continuous representation. As mentioned in [LWW+20] and [HNZ+21], the deep image hashing methods (based on the CNNs) have two main learning objectives: to make the learned binary hash codes discriminative and to minimize a quantization error. Therefore, the main problems are designing the loss function to preserve the similarity structure and optimizing the deep neural network with the discretization problem.

After the first approach to fully use the CNN architecture in CNNH [XPL+14], many different architectures focused on different modifications, such as DSH [LWSC16], ADSH [JL18], HashNet [CLWY17] and many other methods. Later, better results were achieved by methods with pre-defined targets such as DPN [FNJ+20] and CSQ [YWZ+20].

The OrthoHash architecture [HNZ+21] was proposed as the novel deep hashing model. Many state-of-the-art methods employ a large number (>4) of losses, while the OrthoHash focuses only on a single learning objective. Authors claim that maximizing the cosine similarity between the continuous codes and their corresponding binary orthogonal codes can ensure both

hash code discriminativeness and quantization error minimization. Also, most of the deep hashing methods were evaluated on category-level retrieval in classification datasets ImageNet[DDS+09], Cifar-10 [KH+09], MS-COCO [LMB+14]. However, good results on the category-level do not guarantee success on instance-level retrieval. Yet, authors of OrthoHash [HNZ+21] are providing experiments showing that their model outperforms the state-of-the-art hashing models on three large-scale instance retrieval benchmarks GLDv2[WACS20], ROxf and RPar [RIT+18]. In addition, the work of Luo *et al.* [LWW+20] shows that OrthoHash has the lowest training time cost across all the state-of-the-art hashing models since it only leverages one brief objective during optimization. Considering the training simplicity and high performance, the OrthoHash architecture was chosen for experiments in this thesis.

Deep image hashing models can be trained using an end-to-end approach, where both the convolutional backbone and the hashing layers are updated during training. Alternatively, the convolutional backbone can be fixed, so that only the hashing layers are updated during training. Additionally, the convolutional backbone can be pre-trained for image classification or for generating continuous image descriptors before it is used for training.

## 2.3 Selected Method

As discussed in the previous section, there are many different methods for image representation and hashing. This thesis focuses on the 2-staged CNN-based hashing algorithms. The diagram describing the idea of the method is shown in figure 2.2.

First, a CNN-based model is trained on the custom dataset (sec. 3.1) using ranking loss (sec. 2.8.2) to generate continuous codes represented by float numbers. As also mentioned in [Jar21], the CNN global descriptor can be taken from one of the popular architectures for classification tasks (such as VGG [SZ14], ResNet [HZRS15], or Efficient Net [TL19]). The fully connected layers are removed in such architectures, resulting in a fully convolutional backbone. This backbone processes an input image of size $W \times H \times C$, where $W$ and $H$ stand for the width and height of the image, respectively, and C is the number of channels. It produces an output 3-dimensional tensor $\chi$ of size $W' \times H' \times K$, representing the extracted local features, where $K$ is the number of feature maps. The global pooling layer then converts the feature maps $\chi$ into a K-dimensional feature vector $\mathbf{f}$, which is also $L_2$ normalized.

The second step involves learning a hashing layer that takes the K-dimensional real-valued descriptor $\mathbf{f}$ as input and transforms it so that it can be converted into binary hash codes $\mathbf{b} \in \{-1, 1\}^M$ after being processed by the quantization layer (*sign* function). The hashing model must

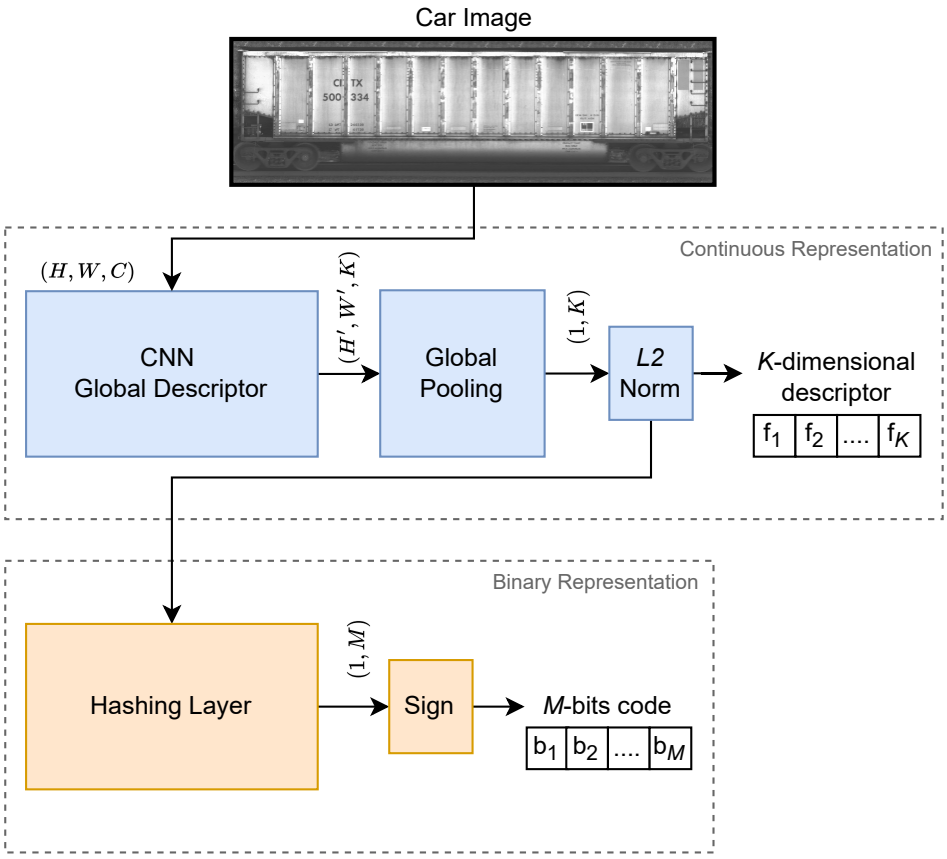



**Figure 2.2:** The idea of the used method: first, obtain continuous descriptor **f** from CNN backbone and then convert it to the binary code **b** using the fine-tuned hashing layer.

aim to satisfy two main objectives: the generated binary codes should be nearly as discriminative as the continuous codes, and the quantization error should be minimized.

The following sections will describe what CNN backbones, global pooling methods, and hashing layers are studied in this work.

## 2.4 Backbones

### 2.4.1 ResNet

ResNet (short for Residual Network) [HZRS15] is a type of convolutional neural network developed by Microsoft Research in 2015. It is well known among researchers for its good performance on image classification tasks and has been used in many image recognition competitions.

The main characteristic of the ResNet architecture is the use of so-called "skip connections," which allow the network to learn residual functions. The network is divided into blocks, each consisting of one or more convolutional layers. The output of each block is then added to the input of the block via a skip connection before being passed on to the next block (see fig. 2.3). This allows the network to learn more efficiently and avoid the vanishing gradient problem that can occur in very deep networks.

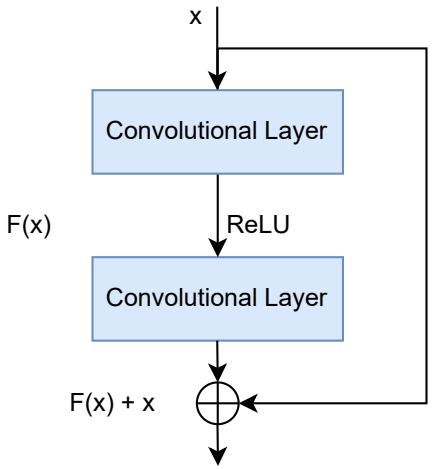



**Figure 2.3:** ResNet skip connection between network blocks.

The number of layers in the ResNet models may vary from 18 to 152; authors in [HZRS15] showed that the deeper architectures are more accurate without the degradation problem being observed; however, the complexity of such networks is increasing significantly. The ResNet-50 was chosen for experiments in this work since it shows promising accuracy and does not require extremely high computational capacity. Moreover, the ResNet-50 was used by RailState for the classification of the car train images; therefore, it is possible to fine-tune the pre-trained model for the image hashing task.

### 2.4.2 EfficientNet

EfficientNet [TL19] is a family of convolutional neural network architectures developed by Google Research in 2019. It is also known for its good performance and efficiency in terms of the number of parameters and computational resources required.

The main characteristic of the EfficientNet architecture is the use of a compound scaling method, which scales the dimensions of the network (e.g., width, depth, and resolution) in a balanced and optimal way. This allows the network to achieve good performance with a smaller number of parameters and fewer computational resources compared to other CNN architectures.

Authors of [TL19] claim that some of the EfficientNet models have a smaller number of parameters and, at the same time, higher accuracy on the ImageNet dataset [DDS+09] than models with ResNet architecture. Starting from EfficientNet-B0, there were obtained EfficientNet-B1 to B7 by scaling up the network dimensions. Table 2.1 compares ResNet and EfficientNet models by top-1 accuracy, the number of parameters, and the number of floating points operation per second (FLOPS), representing the complexity of the models, and also by the number of feature maps of the last CNN layer.

| Model | Top-1 Acc. | #Params | #FLOPs | #Feature Maps |
|---|---|---|---|---|
| ResNet-50 | 76.0% | 26M | 4.1B | 2048 |
| ResNet-152 | 77.8% | 60M | 11B | 2048 |
| EfficientNet-B0 | 77.1% | 5.3M | 0.39B | 1280 |
| EfficientNet-B2 | 80.1% | 9.2M | 1.0B | 1408 |
| EfficientNet-B5 | 83.6% | 30M | 9.9B | 2048 |
| EfficientNet-B7 | 84.3% | 66M | 37B | 2560 |

**Table 2.1:** Selected EfficientNet and ResNet models along with their accuracy on ImageNet [DDS+09], number of network parameters, FLOPS, and the size of the descriptor. Data are provided by [TL19]

In this work, it was decided to use the EfficeinNet-B2 for further study, which is 2.8 times fewer parameters, 4.1 times fewer FLOPS, and 5% higher ImageNet accuracy than ResNet-50. The more complex EfficentNet model was not chosen since the complexity and inference speed of the model matter. In addition, the number of feature maps of the EfficientNet-B2 is lower than ResNet-50 meaning that it requires fewer parameters to represent the feature space, and it may also reduce the complexity of the hashing layer.

## 2.5 Global Pooling Methods

Earlier approaches to applying the CNNs for image description used fully connected layers after the CNN feature maps. However, after the work of Razavian *et al.*[RSMC14], the focus was moved to global pooling methods. Global pooling operation is used to convert CNNs 3D feature maps $\mathcal{X}$ of dimensions $W' \times H' \times K$ into K-dimensional feature vector $\mathbf{f}$ (see fig. 2.4).

There are several popular methods of extracting values from each feature map:

- **Sum Pooled Convolutions (SPoC)** [BL15] is a simple average pooling, where the average value is taken for each feature map $\mathcal{X}_i$:

$$f_i = \frac{1}{\|\mathcal{X}_i\|} \sum_{x \in \mathcal{X}_i} x. \tag{2.1}$$

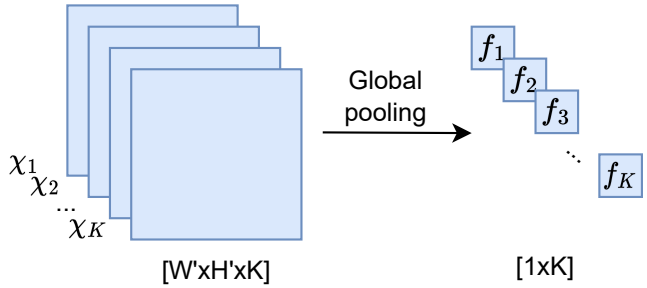



**Figure 2.4:** Global pooling

- **Max Pooled Convolutions (MAC)** [TSJ15] is taking the maximal value all over the dimensions per each feature map $\chi_i$:

$$f_i = \max_{x \in \chi_i} x. \tag{2.2}$$

- **Generalized Mean (GeM)** [RTC18] pooling is proposed by Radenović *et al.* and generalizes MAC and SPoC operations using the trainable parameter. GeM pooling is defined as follows:

$$f_i = \left( \frac{1}{\|\chi_i\|} \sum_{x \in \chi_i} x^{p_i}. \right)^{\frac{1}{p_i}} \tag{2.3}$$

  where $\chi_i$ is feature map and $p_i$ is a parameter. If $p_i$ equals 1, the pooling becomes average pooling or SPoC; however, if the $p_i$ is approaching infinity, it becomes maximum pooling or MAC. Since the parameter $p_i$ is differentiable, it can be back-propagated and learned during the end-to-end training. The value of $p_i$ can be shared across all the feature maps ($p_i = p$). The authors of [RTC18] claimed that learning a shared parameter turns out to be better than learning multiple ones. They also showed that GeM pooling outperforms SPoC and MAC methods. Therefore, this setup is used in this work.

## 2.6 Whitening

The work of Jegou and Chum [JC12] showed that whitening the data representation is very essential for image retrieval tasks. Whitening is a technique that reduces information redundancy in image representation data and is often used to improve feature extraction performance. It standardizes the data by removing correlations and scaling the data. The whitening can be learned end-to-end as a fully-connected layer applied after the pooling as in [GARL16] or as a post-processing procedure learned in a discriminative manner [RTC18].

## 2.7 OthoHash

As mentioned in 2.2.2, OrthoHash [HNZ+21] is a single-loss image hashing model that removes the difficulty of tuning coefficients of various losses. OrthHash generates hash centers or orthogonal targets for each class using the Bernoulli distribution. Then it maximizes the cosine similarity between the $L_2$-normalized continuous codes and binary orthogonal targets. The author shows that their single loss function based on cosine similarity can maximize inter-class Hamming distance and minimize quantization error simultaneously. The formulation of the loss function is written in the following section 2.8.4.

The Orthohash method is also illustrated in Figure 2.5. First, it obtains $K$-dimensional continuous codes $\mathbf{F} = \{\mathbf{f}_n\}_{n=1}^{N} \in R^{N \times K}$ from the backbone network, where $N$ is the number of input samples. It is then passed through a fully-connected (FC) layer and batch normalization (BN) layer to obtain zero-mean $M$-dimensional continuous vectors $\mathbf{B} = \{\tilde{\mathbf{b}}_n\}_{n=1}^{N} \in R^{N \times M}$. Those vectors can be later converted to binary codes $\mathbf{b}_n$ with the *sign* function. Next, the scaled cosine similarity is computed between the processed codes and their binary orthogonal targets $\mathbf{O} = \{\mathbf{o}_i\}_{i=1}^{C} \in \{-1, +1\}^{C \times K}$, where $C$ is the number of classes (or instances in terms of this work). Finally, the scaled cosine similarity will act as a classification output or logits, and it is passed to a cross-entropy loss.

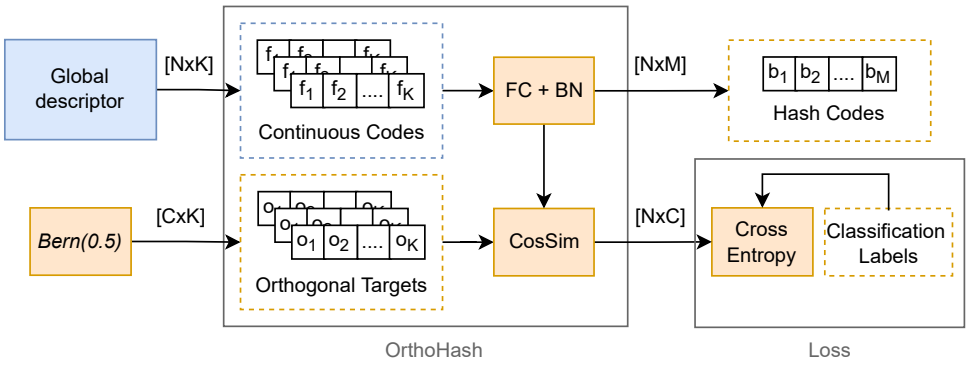



**Figure 2.5:** Illustration of OrthoHash architecture.

## 2.8 Losses

The typical way for learning image representation is to use Siamese architecture and train a network with several branches, where each branch is a clone of the other, meaning that they share the same parameters [HNZ+21][Jar21]. While the training, the network receives a tuple of images (query, positive, and negatives) with assigned labels, representing the samples of the same

and different classes. After each image's feature vectors are extracted, the distances (usually euclidean) are computed between the samples. Those distances are then passed to the ranking loss, which minimizes the distance between similar samples and maximizes the distance between dissimilar. Therefore, the model is optimized to find a representation that will cluster images and isolate samples of one class from another in a feature space.

### 2.8.1 Triplet Loss

The triplet loss is the most commonly used ranking loss used in various tasks: e.g., place recognition [AGT+15], fine-grained image similarity [WsL+14], face recognition [SKP15]. Similar to other ranking losses, it operates with a triplet of samples that are categorized as:

- **query** or anchor sample that is used as the reference for the tuple,
- **positive** sample that is similar to the query from the same class,
- **negative** sample that is dissimilar to the query from another class.

The loss is defined as follows:

$$\mathcal{L} = \max\left(0; \left\|\mathbf{f}(Q) - \mathbf{f}(P)\right\|_2^2 - \left\|\mathbf{f}(Q) - \mathbf{f}(N)\right\|_2^2 + m\right), \tag{2.4}$$

where $m$ is a parameter corresponding to the margin that is enforced between positive and negative pairs, $\mathbf{f}(I)$ is a descriptor producing feature vectors from image $I$, and $Q, P, N$ are query, positive and negative images, respectively.

As mentioned in [SKP15], generating all possible triplets would result in many easily satisfied triplets that do not contribute to the training and result in slower convergence. It is crucial to select so-called hard negative samples that can actively contribute to improving the model. Usually, the negatives with the most similar descriptor to the query are chosen. This work's procedure of generating hard negatives is described in section 3.2.

### 2.8.2 Contrastive Loss

Like the triplet loss, the contrastive loss takes extracted features from the query, positive and negative samples. The definition of loss is the following:

$$\mathcal{L}(i,j) = \begin{cases} \frac{1}{2}\left\|\mathbf{f}(i) - \mathbf{f}(j)\right\|_2^2, & \text{if } Y(i,j) = 1 \\ \frac{1}{2}(\max\{0, m - \left\|\mathbf{f}(i) - \mathbf{f}(j)\right\|_2\})^2, & \text{if } Y(i,j) = 0 \end{cases} \tag{2.5}$$

where $Y(i,j) \in \{0,1\}$ is a set of labels declaring whether the pair of images $(i,j)$ is similar (label 0) or dissimilar (label 1) and other attributes are the same as in 2.4. In this case, the margin parameter $m$ determines whether the non-matching pairs are separated wide enough so that the loss may ignore them in training and focus on more difficult pairs. The illustration of contrastive loss is shown in figure 2.6.

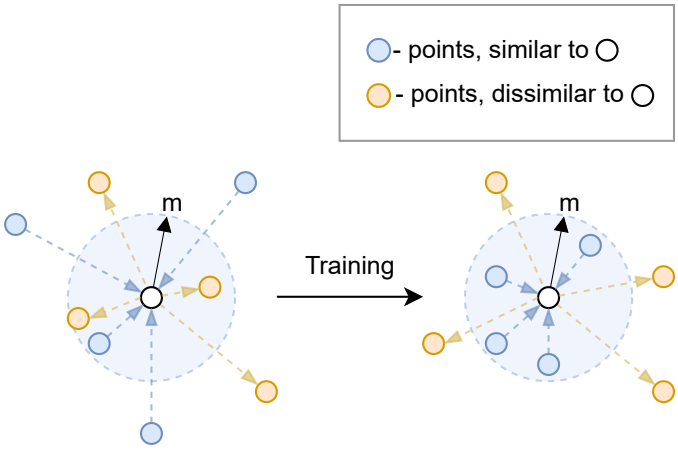



**Figure 2.6:** The concept of a contrastive loss. The loss minimizes the distance of positives and pushes the negatives away from the margin $m$ around the query point.

The authors of [RTC18], with a network architecture similar to the one proposed in this project, reported that contrastive loss generalizes better and converges at higher performance than triplet loss. Therefore this work is mainly focused on contrastive loss.

### 2.8.3 Cross Entropy

Cross entropy loss, also known as the negative log-likelihood loss, is a measure of the difference between two probability distributions [GBC16]. It is defined as:

$$L(p,q) = -\sum_i p_i \log q_i \tag{2.6}$$

where $p$ is the true distribution and $q$ is the predicted distribution.

Cross entropy is typically used for classification tasks, where the softmax function transforms the output of the final linear layer into a probability distribution over the classes. The softmax is defined as follows:

$$\sigma(f)_i = \frac{e^{f_i}}{\sum_j^K e^{f_j}}, \tag{2.7}$$

where $f$ is the input vector and $K$ is the number of classes. The cross-entropy loss is then calculated by comparing the predicted probability distribution

with the true distribution, which is represented as a one-hot vector with a one at the index of the actual class and zeros elsewhere.

### 2.8.4 OrthoCos

As described in section 2.7, OrthoHash loss, also called OrthoCos loss, is maximizing the cosine similarity of the continuous codes $\mathbf{f}_n$ and its corresponding binary orthogonal targets $\mathbf{O} = \{\mathbf{o}_i\}_{i=1}^{C} \in \{-1, +1\}^{C \times K}$ [HNZ+21], where $C$ is a number of classes and $K$ is a number of bits. This is achieved by maximizing the posterior probability of the ground-truth class using softmax and cross-entropy loss:

$$L = -\frac{1}{N} \sum_{n=1}^{N} \log \frac{\exp\left(\mathbf{o}_{y_n}^{\top} \mathbf{f}_n\right)}{\sum_{i=1}^{C} \exp\left(\mathbf{o}_i^{\top} \mathbf{f}_n\right)}, \tag{2.8}$$

where $N$ is a number of samples and $y_n$ is the ground-truth class index. It follows that, the logits $\mathbf{o}_i^{\top} \mathbf{f}_n$ can be transformed as:

$$\mathbf{o}_i^{\top} \mathbf{f}_n = \|\mathbf{o}_i\| \|\mathbf{f}_n\| \cos \theta_{ni}, \tag{2.9}$$

where $\theta_{ni}$ is the angle between continious codes $\mathbf{f}_n$ and the targets $\mathbf{o}_i$. Next, the $L_2$ normalization can be applied on $\mathbf{f}_n$ and $\mathbf{o}_i$, so that $\|\mathbf{f}_n\| = 1$ and $\|\mathbf{o}_i\| = \sqrt{K}$, since it is in binary form. Furthermore, the cosine margin can be leveraged with a parameter $m$, which authors of [HNZ+21] claim to be beneficial. The resulting OrthoCos loss function then looks as follows:

$$L = -\frac{1}{N} \sum_{n=1}^{N} \log \frac{\exp\left(\sqrt{K} \cos \theta_{y_n} - m\right)}{\exp\left(\sqrt{K} \cos \theta_{y_n} - m\right) + \sum_{i=1, i \neq y_n}^{C} \exp\left(\sqrt{K} \cos \theta_{ni} - m\right)} \tag{2.10}$$

## 2.9 Metrics

To measure the performance of an image representation model search accuracy metrics are used - *Precision*, *Recall* and *Mean Average Precision* [LWW+20]. The metrics are computed on the retrieved top-$k$ samples for the query image search.

- **Precision** is defined as a proportion of all returned samples and samples that match with the query, which can be formulated as:

$$precision = \frac{TP}{TP + FP}, \tag{2.11}$$

where TP (True Positives) and FP (False Positives) denote the number of returned samples that are matching and not matching the query, respectively. *precision@k* usually denotes precision for the total number of returned samples is $k$, i.e. $k = TP + FP$.

- **Recall** determines the proportion of returned matching samples across all the matching samples in the database and can be formulated as:

$$recall = \frac{TP}{TP + FN}, \tag{2.12}$$

where FN (False Negatives) denotes the matching samples that were not retrieved. Like for *precision*, *recall@k* denotes recall for the number of retrieved images equal $k$.

- **Mean Average Precision (mAP)** is calculated as a mean of average precision computed for all queries in the dataset. The average precision is defined as follows:

$$AP@n = \frac{1}{F} \sum_{k=1}^{n} precision@k \times \Delta recall@k, \tag{2.13}$$

where $n$ denotes the total number of retrieved samples, and $\Delta recall@k$ denotes the change in recall from item $k-1$ to $k$, and $F$ refers to the sum of $\Delta recall@k$. The core idea of AP is to evaluate a ranked list by averaging the precision at each position. The mAP is then formulated as:

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i, \tag{2.14}$$

where N is the number of queries.

# Chapter 3

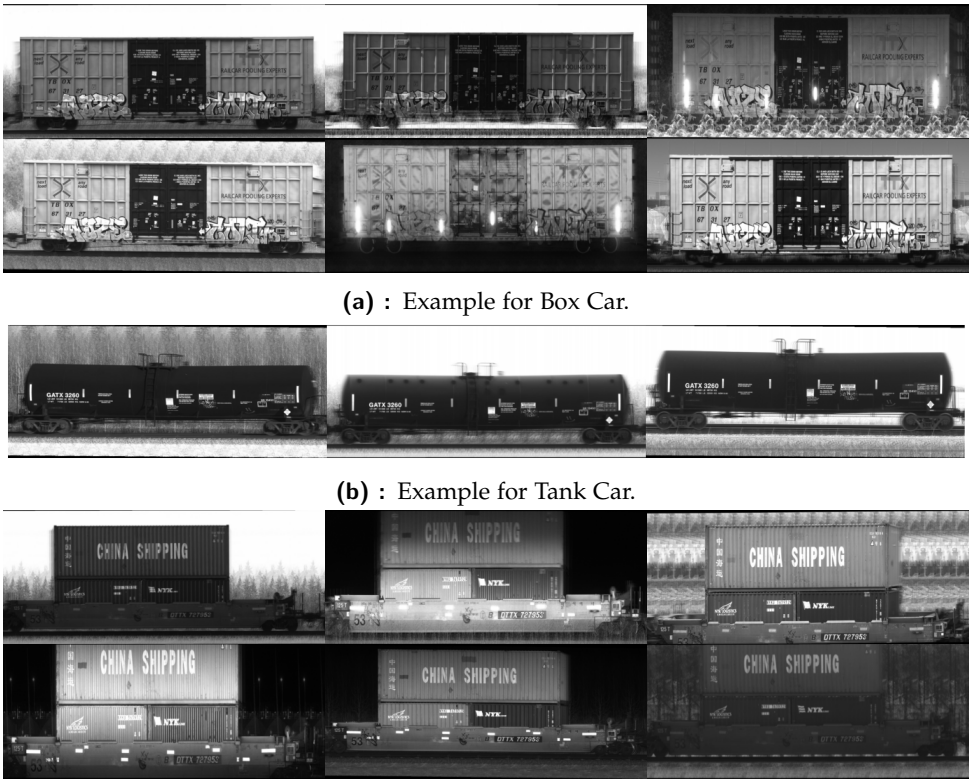
# Datasets

## 3.1 Dataset Creation and Structure

To be able to apply supervised learning algorithms, it is necessary to prepare a dataset with labeled images. All the images of the same instance (train car) should be grouped so it is possible to split them into positive and negative samples. This work implements the procedure to gather datasets without human annotation.

RailState LLC provides all the data used in this work. As stated in introduction 1, there is an existing solution for matching train cars. Using this solution, the train cars captured from different locations and at different times can be aligned together.

The tool for dataset preparation implements the following. First, the list of locations is chosen, so it represents the variety of passing cars, then the time interval is set. The tool retrieves the cars from the selected locations and looks for matching cars from other locations. Those matched car images are then retrieved and stored in groups, representing each car instance. The left and right sides of the train car usually do not look the same due to particular unique patterns (graffiti, scratches, stains, etc.). Therefore images of the same car captured from left or right are considered as different instances in this work. In the end, the car instance is characterized by the car identification number, car type, and the captured side. Each image has also been assigned a label specifying whether the image was taken during the day or night time. The dataset class implemented during this work allows easy separation of the data, so it is possible to split the instances by day or night or other potential characteristics. Examples of the car images from one instance are shown in Fig. 3.2. Note that the appearance of cars can change over time while they are being loaded or unloaded, or for example, if the containers carried by Stack Car were changed. The system is designed for use in short intervals, so those cases are not considered.



**(a) :** Example for Box Car.

**(b) :** Example for Tank Car.

**(c) :** Example for Stack Car.

**Figure 3.2:** Examples of images from one instance for different classes taken from different locations.

The dataset images are grayscale. However, the channel is copied three times to reproduce a 3-channel image. This is the most straightforward way to get input for the CNN networks that were pre-trained on the 3-channel datasets like ImageNet [DDS+09]. The aspect ratio and resolution of dataset images vary. The image's width is usually three or four times larger than the height. Since the images were captured from different locations, data include different lighting and weather conditions, extreme changes in contrast and brightness, varying backgrounds, and car placement. Some of the images are also partially covered by falling snow or raindrops. All this variety makes the task more challenging.

Eventually, three different datasets were collected: for training, validation, and testing. All the datasets were retrieved from different time periods, which ensures they do not include the same images. Table 3.1 shows the distribution of samples across car types, the number of instances, and the total amount of images for each dataset. The occurrence of different car types is usually not balanced. However, the tool allows penalizing retrieval of selected types, which can partially normalize the distribution of car types.

Figure 3.3 shows the distribution of image numbers per one instance

| | Train | | Val | | Test | |
|---|---|---|---|---|---|---|
| Car Type | #Img. | #Inst. | #Img. | #Inst. | #Img. | #Inst. |
| Box Car | 772 | 248 | 202 | 66 | 418 | 151 |
| Flatcar | 806 | 294 | 192 | 66 | 180 | 66 |
| Gondola | 4140 | 1208 | 512 | 162 | 1228 | 348 |
| Hopper | 7294 | 2053 | 729 | 192 | 2187 | 651 |
| Locomotive | 1415 | 413 | 303 | 87 | 400 | 118 |
| Vehicular Flatcar | 1351 | 385 | 399 | 113 | 451 | 131 |
| Stack Car | 4355 | 1070 | 677 | 186 | 2861 | 717 |
| Tank Car | 2582 | 848 | 343 | 108 | 851 | 294 |
| Other | 92 | 26 | 25 | 8 | 31 | 11 |
| **Total** | **22807** | **6544** | **3382** | **987** | **8607** | **2486** |

**Table 3.1:** The total and aggregated by car type number of samples and the number of instances for training, validation, and test datasets.

for each dataset. The previously described procedure of dataset creation does not allow to gather of more than ten samples of one car. The images can be captured from a bigger number of locations. However, they are also separated into left or right car sides. One-third of the instances have only two samples, meaning that there is only one positive pair. This is a very different scenario in comparison with the large-scale image recognition dataset GLDv2 [WACS20] or benchmark datasets *R*Oxford/*R*Paris [RIT+18] that are usually used for evaluation. Therefore, the results of experiments achieved in this work may not follow the results in the works that used those datasets.

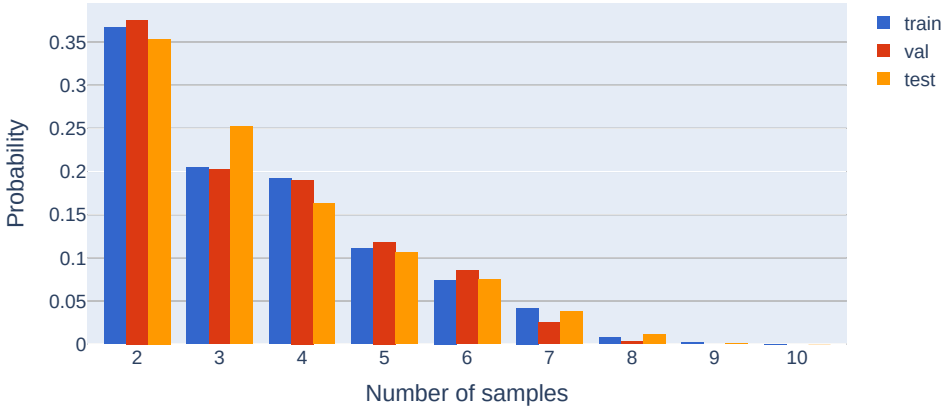



**Figure 3.3:** Histogram showing normalized counts of image numbers per instance for train, validation, and test datasets

The provided system is imperfect; therefore, the car alignment method is based on heuristics and provides a probability of car matching. In this

work, only the images of the cars that are correlated with high probability are considered. However, it is possible that some of the images in the dataset can be mislabeled or misclassified. Some of the images can also be corrupted, covering only part of the car. The procedure described in section 3.3 describes how to detect and remove possible outliers.

## 3.2 Negatives Mining

As mentioned in 2.8.2, learning with ranking losses requires assigning query, positive and negative tuples. This work applies a similar approach as used in [RTC18]. First, the query samples are randomly selected across all the training datasets. The positive samples are then assigned to each query simply as one of the samples from the same instance. The random assignment of negatives may be inefficient and block the loss convergence since some of the samples can already fulfill loss margin requirements, resulting in a zero error. Therefore, the hard negatives are selected: the samples from different instances with the most similar descriptor to the query. This procedure, called hard negatives mining, is applied after each epoch with a more and more optimized model, which provides more complicated cases each time. The hard negatives are only chosen from the selected negative pool that is re-assigned each epoch, which can ensure that different samples will occur in each epoch, even if the model does not change much. The example of training tuples is shown in figure 3.4.

## 3.3 Semi-automatic Dataset Cleanup

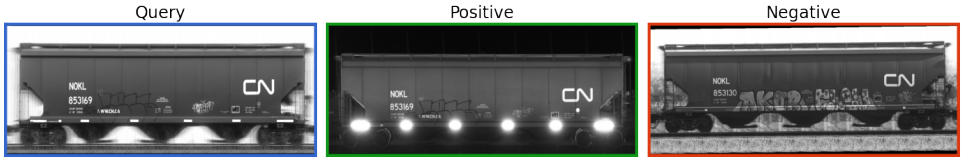
As mentioned in the previous section, miss-labeled images can occur in the datasets, which can affect the training process and validation metrics. Yet, it is possible to detect potentially incorrect car images automatically. The most straightforward way would be to compute the center of mass of all the feature vectors in each class and then find potential outliers by analyzing the distance from that center. However, the center of mass or the mean can be significantly shifted by outliers, and therefore it is a good idea to use the geometric median [FVJ09].

For the given set of points $x_1, x_2, \ldots, x_n \in R^d$, the geometric median (also known as the Fermat point, Weber's L1 median) is defined as follows:

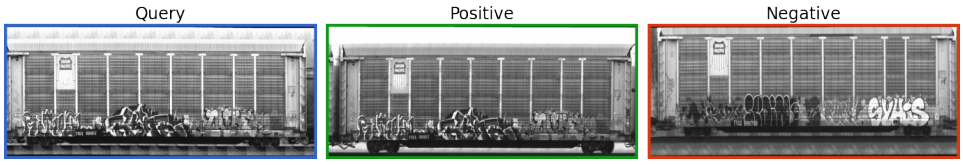
$$x_{GM} = \arg\min_{z \in R^d} \sum_{i=1}^{n} \|z - x_i\|_2 \tag{3.1}$$

Figure 3.5 shows an abstract illustration of the geometric median and mean.


**(a) :** Training tuple example for Covered Hopper.




**(b) :** Training tuple example for Covered Hopper.




**(c) :** Training tuple example for Vehicular Flatcar.

**Figure 3.4:** Examples of good training tuples (query, positive, negative). The negative samples look very similar; they are from the same car type as the query and have similar patterns; however, the car instances are different. Therefore, the ranking loss is high, and the network is forced to learn the features that will help to distinguish such cases.

Then, the modified z-score is applied to detect outliers in the instance group, which is a robust, standardized score that measures outlier strength [OEOI14]. Modified Z-score, in this case, is defined as follows:

$$MZ_i = \frac{\|x_i - x_{GM}\|_2}{MAD} \tag{3.2}$$

where $x_i$ is an $i$-th feature vector, $x_{GM}$ is the geometric median of the feature vectors of the current instance, and $MAD$ is the median absolute deviation. The modified Z-score is then compared with an absolute threshold $t$. If the $MZ_i$ is greater than $t$, the $i$-th sample is labeled as a potential outlier.

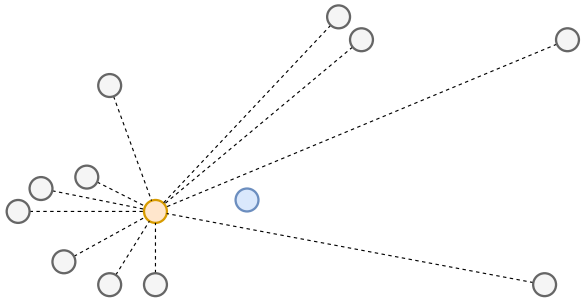


**Figure 3.5:** The idea of the difference between the geometric median (in orange) and the center of mass (in blue) of a series of points.

This procedure was applied to the datasets after training one of the initial

descriptors. The potential outliers were manually checked, and the true outliers were removed from the datasets. The value of the threshold was set to $t = 2$. The procedure automatically detected 493 potential outliers across all datasets, and 177 of them were actual outliers and were removed. Table 3.1 shows the image counts after this cleanup procedure.

# Chapter 4

# Experiments and Analysis

## 4.1 Framework

This work integrates various open-source libraries. As machine learning research progresses rapidly, it is essential to use up-to-date libraries when implementing neural networks.

The training and validation pipeline, data preparation tools, and all the helping utilities were implemented during this work. The implementation was mainly based on *PyTorch* [PGM+19], *NumPy* [HMvdW+20], and *Pandas* [pdt20] libraries. The graphs and visualizations were made using *Plotly* [Inc15] and *Matplotlib* [Hun07]. The experiment tracker *Neptune.ai* [nep22] was used to monitor the training process and store the model weights. The service was beneficial in visualizing and tracking the learning progress and comparing the results of various models.

The architecture of *OrthoHash* and the *OrthoCos* loss were integrated from the official author's implementation [HNZ+21]. *GeM* pooling and *Contrastive Loss* layers were integrated from official Radenović implementation [RTC18] that was based on pure PyTorch.

## 4.2 Training Setup

The training procedure of the task was split into two parts: first, to train a good image representation with continuous (float) codes and then train a hash layer one the same training set that will convert continuous codes into binary.

For a fair comparison, the training setup was fixed except for the studied parameters for all the experiments. The Adam optimizer [KB14] was used

with a step learning rate scheduler. The metric mAP in the following experiments is computed for top-10 retrievals. Euclidean distance was used for continuous codes, and hamming distance for the binary codes. The default input image resolution was set to 224x720, with an aspect ratio corresponding to most images in the datasets.

### 4.2.1 Continuous Representation

As mentioned in 2.8.2 the image representation model with continuous codes was trained with a contrastive loss. The loss margin was set to 0.7. Each input tuple was formed of 1 query image, one positive, and five hard negative samples. The batch size was set to 5. However, the model was fed one tuple at a time, and the loss was accumulated until the whole batch was processed. Due to the fact of training one tuple at a time, the running statistics of batch normalization layers were frozen while training. The queries, positives, and negatives were sampled randomly before each epoch. After each epoch, the hard negatives pool was re-calculated with a procedure described in 3.2. The size of the query pool was set to 2000, and the size of the negative pool was set to 10000. The number of epochs was 50. The starting learning rate was set to $5 \cdot 10^{-4}$ and then reduced by 0.5 each 10th epoch with the weight decay of $5 \cdot 10^{-6}$.

Since the training procedure may be affected by random initialization and positive/negative sample assignments, most of the main experiments were repeated several times. Table 4.1 shows how the metrics may vary for the same experiment with a ResNet50, GeM, and 224x720 input resolution. The metrics in the tables of the following sections are shown for the best runs.

| | Run 1 | Run 2 | Run 3 |
|---|---|---|---|
| Val mAP | 30.2 | 31.7 | 30.5 |
| Test mAP | 21.1 | 22.5 | 22.2 |

**Table 4.1:** mAP of several runs of the same experiment with a ResNet50 + GeM and 224x720 input resolution

### 4.2.2 Binary Hash Codes

The training setup for binary representation was similar to the procedure described in OrthoHash work [HNZ+21]. The starting learning rate was set to 0.0001 and then lowered by a factor of 10 at epochs 12 and 24, with the total number of epochs equal to 30 and a weight decay of 0.0005. The margin loss parameter was set to $m = 0.2$ (see sec. 2.8.4). In the first experiments, the backbone layer producing continuous codes was frozen, so only the hashing layer was trained, as it was used in [HNZ+21]. The batch

size was set to 32 for experiments with a frozen backbone and 8 for full model training due to the limitations of the GPU.

## 4.3 Training Experiments

### 4.3.1 Whitening

The first experiments involved the training of the initial model with ResNet backbone and GeM pooling. In those experiments, the impact of the additional whitening layer was tested (see sec. 2.6). The fully connected layer with $L_2$ normalization was added after ResNet-50 with GeM pooling and trained end-to-end. It was observed that the convergence of the network is much slower in this case, and the final achieved performance is lower than without an additional layer. Moreover, the whitening layer may also slow down the convergence of a hashing layer. Similar results were reported in Radenović [RTC18] work, which shows that the whitening introduced as a post-processing step achieved better results. However, this work did not test this technique due to complex integration with a hashing layer.

Since the additional layer increases the complexity of the model and does not improve performance, it was decided to use GeM pooling without additional layers in future steps.

| Model | Input | Val mAP | Test mAP |
|---|---|---|---|
| ResNet50+GeM | 224×720 | 31.7 | 21.1 |
| ResNet50+GeM+W | 224×720 | 30.2 | 18.9 |
| ResNet50+GeM | 336×1080 | **38.6** | **27.7** |
| ResNet50+GeM+W | 336×1080 | 35.6 | 23.9 |

**Table 4.2:** Performance comparison of ResNet-50 models with/without additional whitening (W) and OrthoHash layer (OH) for different input image sizes. The metrics are computed on validation and test datasets.

### 4.3.2 Backbones

As discussed in 2.4 it was decided to experiment with an Efficient Net architecture as a backbone. Table 4.3 shows a significant performance improvement for the Efficient Net B2 backbone for both continuous and binary representations. The Efficient Net was pre-trained on ImageNet [DDS+09] dataset, while the ResNet-50 was pre-trained on the classification dataset from the same train cars domain.

The change of loss and validation mAP during training is shown in figure 4.1. As you may observe, the EfficientNet model has better accuracy already

| Architecture | Input | Feature Size | Val mAP | Test mAP |
|---|---|---|---|---|
| ResNet50+GeM | 224×720 | 2048 | 30.2 | 21.1 |
| EfficientNet+GeM | 224×720 | 1408 | 60.3 | 52.0 |
| ResNet50+GeM | 336×1080 | 2048 | 38.6 | 27.7 |
| EfficientNet+GeM | 336×1080 | 1408 | **68.4** | **61.4** |

**Table 4.3:** Comparison table of ResNet and EfficientNet B2 backbones and OrthoHash (OH) for different input image sizes. Feature size shows the number of floats in the output feature vector. The metrics are computed on validation and test datasets.

after a few first epochs and converges faster. One of the possible reasons could be that the backbones were pre-trained on a different dataset. Also, EfficientNet has fewer training parameters, which could simplify and speed up the training process.

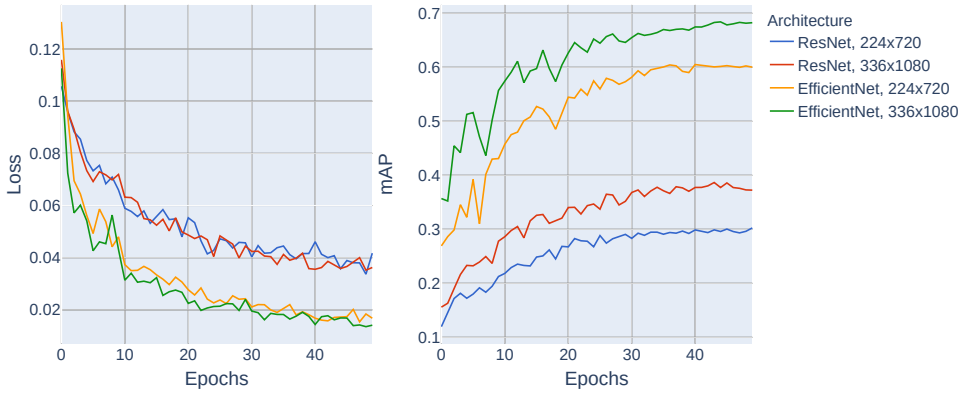


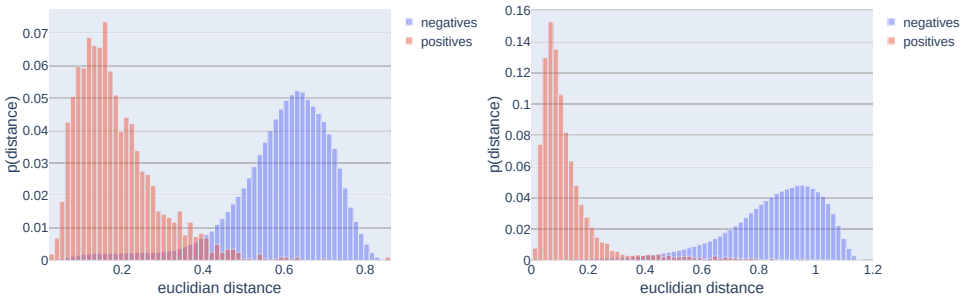

**Figure 4.1:** Graphs of Loss and mAP change during the training for ResNet50 and EfficientNet B2 models with a contrastive loss function.

Figure 4.2 shows the histogram of euclidean distances between positive and negative samples for studied backbones. The difference between distributions of positive and negative distances for EfficientNet is higher, indicating that it has better separability.

### 4.3.3 Hashing Layer

The table 4.4 shows the results of training the hashing layers with different backbones and input image sizes. The table also shows results for the simple Cross-Entropy (CE) model, which denotes the model with a similar architecture to an OrthoHash (OH), which, instead of a cosine similarity module, has the fully connected layer that produces logits for the cross-entropy loss.

It was observed during experiments that backbone freezing leads to an




**(a) :** ResNet+GeM, 336×1080 **(b) :** EfficientNet+GeM, 336×1080

**Figure 4.2:** Histograms of euclidean distances between positives and negatives computed on the test dataset. The frequency is normalized, so the sum of all (64) bins is equal to 1.

| Architecture | Input | Backbone Train | Val mAP | Test mAP |
|---|---|---|---|---|
| ResNet50+OH | 224×720 | ✗ | 24.6 | 18.5 |
| ResNet50+OH | 224×720 | ✓ | 43.9 | 33.9 |
| EfficientNet+OH | 224×720 | ✗ | 41.6 | 32.2 |
| EfficientNet+OH | 224×720 | ✓ | 88.7 | 83.7 |
| ResNet50+OH | 336×1080 | ✗ | 33.5 | 24.0 |
| ResNet50+OH | 336×1080 | ✓ | 53.8 | 45.4 |
| EfficientNet+OH | 336×1080 | ✗ | 51.1 | 43.2 |
| EfficientNet+OH | 336×1080 | ✓ | **91.3** | **86.7** |
| EfficientNet+CE | 336×1080 | ✗ | 53.4 | 45.5 |
| EfficientNet+CE | 336×1080 | ✓ | 82.5 | 76.7 |

**Table 4.4:** Comparison table of ResNet and EfficientNet B2 backbones and OrthoHash (OH) with GeM pooling for different input image sizes. Feature size shows the number of floats in the output feature vector. The metrics are computed on validation and test datasets.

accuracy drop concerning the corresponding continuous descriptor used as a backbone, whereas training the whole network can significantly improve the representation performance. Moreover, the performance of the OrhtoHash is lower than the performance of the baseline CE model for the frozen EfficientNet backbone and 336×1080 input.

The authors of [HNZ+21] reported that the binary representation might slightly outperform the continuous descriptors used as a backbone. However, in the case of this work, the 30%-40% accuracy gain is observed. The specificity of the dataset might explain this significant advantage. Most of the instances in the dataset have only a few positive pairs, which may decrease the efficiency of the ranking loss since it has low variability. On the other hand, the hashing loss is not dependent on the number of positive pairs, and each sample is taken into account separately. The further study in section 4.4.4 shows how feature clustering differs for binary and continuous representations. Training the whole network also significantly increases

training time, which lasts 3-4 times longer.

Figure 4.3 shows the change of loss and validation mAP while training the previously mentioned models. You may see that the models with frozen backbones do not converge well and do not improve much during 30 epochs, unlike the fully trained models.

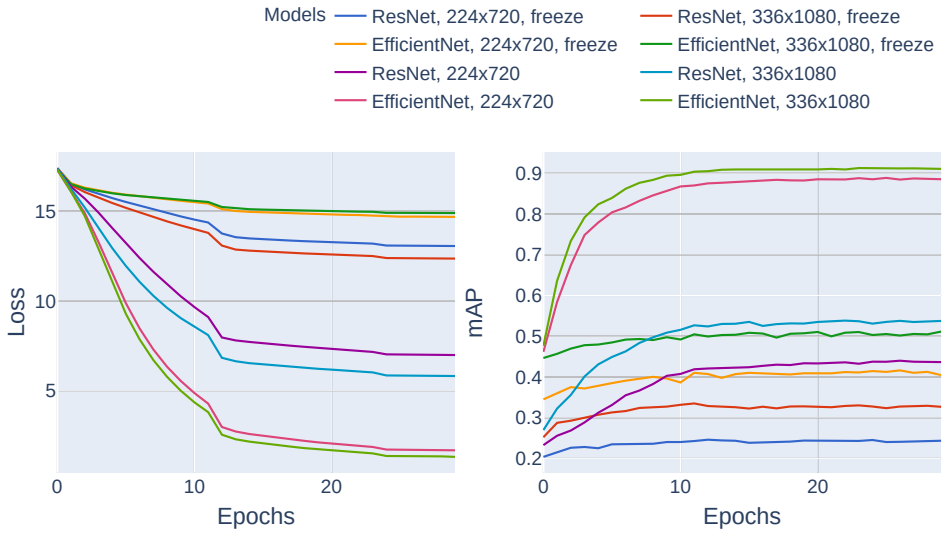



**Figure 4.3:** Graphs of Loss and mAP change during the training for ResNet50 and EfficientNet backbones with OrthoHash layer. Models with frozen backbones during training are marked with a "freeze" label.

### 4.3.4 Hash Size

Table 4.5 shows the memory footprint of the produced features for continuous and binary representation. One may see that quantization with a hashing layer reduces the amount of memory needed to store the representation 32 times for ResNet and 22 times for EfficientNet B2.

| Architecture | Occupied Memory |
|---|---|
| ResNet50+GeM | 8192B |
| EfficientNet+GeM | 5632B |
| ResNet50+GeM+OH | 256B |
| EfficientNet+GeM+OH | 256B |

**Table 4.5:** Comparison table with the amount of memory needed to store the extracted features is shown in bytes, considering one float number is represented by 4 bytes. OrthoHash (OH) layer is considered to produce 2048-bits codes

Nonetheless, it is possible to lower or increase the number of bits for the hashing layer. The graph 4.4 shows how the dimensionality of a hashing layer may affect the total accuracy. The EfficientNet with GeM pooling

and input resolution 336×1080 was used as a backbone. The experiments were done for the fully re-trained model and the model with a freezed backbone, pre-trained for continuous representation. However, due to high computational cost, the re-trained backbone was trained only once for the 2048-bit OrtoHash layer and then was used to fine-tune the hashing layers with different hash sizes. It is expected that the dependencies on hash size would be similar to training the whole model end-to-end. The results of these experiments show the trade-off between accuracy and memory requirements.

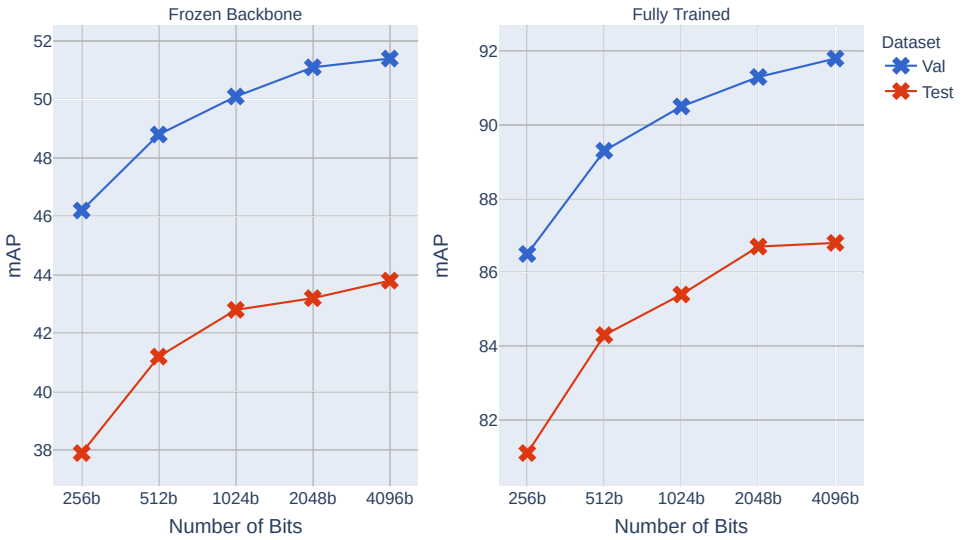



**Figure 4.4:** Performance comparison of EfficientNet backbone (freezed/trained) and OrthoHash layer with different hash sizes for 336×1080 input image.

## 4.4 Analysis

### 4.4.1 Image Resolution

With higher-resolution input images, convolutional backbones can potentially capture more fine-grained patterns. In terms of this work, it may help to distinguish an identification number of a train car or other more minor details. Tables 4.2, 4.3, and 4.4 also compare the performance of the models trained and evaluated with 224×720 and 336×1080 input image resolution. It is clear that higher resolution increases accuracy and improves representativeness. However, the resolution may also significantly affect the inference speed, which is studied in section 4.5.

### 4.4.2 Day/Night Changes

The day and night images of the same car can be very visually dissimilar; therefore, it is challenging for the model to find a good representation invariant to the lighting. The best binary and continuous representation models were evaluated on the separated by the time of day datasets (see Fig. 4.6). Day- and night-only datasets include images captured during the day or night, respectively.

| | Both | | Day Only | | Night Only | |
|---|---|---|---|---|---|---|
| Architecture | Val | Test | Val | Test | Val | Test |
| EfficientNet+GeM | 68.4 | 61.4 | 73.9 | 72.8 | 76.3 | 65.7 |
| EfficientNet+GeM+OH | 91.3 | 86.7 | 94.0 | 92.5 | 93.6 | 87.5 |

**Table 4.6:** Performance on the separated by time of the day datasets for the best continuous and binary descriptors. Both models were trained and evaluated on 336×1080 input image resolution.

It is observed that the models have slightly better metric values on the separated datasets. This could be explained by fewer images in those datasets. However, this also indicates that there is room for improvement for models to be more lighting invariant.

### 4.4.3 Car Types Analysis

Different car types usually have different features that the descriptor can capture. Table 4.7 shows the mAP metrics calculated for each car type separately.

| Model: | EfficientNet+GeM | | EfficientNet+GeM+OH | |
|---|---|---|---|---|
| Car Type | Val mAP | Test mAP | Val mAP | Test mAP |
| Box Car | 80.6 | 76.0 | 95.0 | 96.8 |
| Flatcar | 42.2 | 34.7 | 87.3 | 74.6 |
| Gondola | 66.5 | 52.5 | 92.8 | 84.2 |
| Hopper | 85.3 | 73.1 | 95.6 | 91.2 |
| Locomotive | 23.6 | 21.4 | 64.8 | 60.0 |
| Other | 38.9 | 67.8 | 58.3 | 80.4 |
| Stack Car | 79.6 | 64.0 | 95.0 | 88.2 |
| Tank Car | 71.8 | 56.9 | 92.7 | 82.9 |
| Vehicular Flatcar | 63.5 | 58.3 | 90.9 | 90.7 |

**Table 4.7:** Train and validation metrics across different car types for the best continuous and binary descriptors with EfficientNet backbone. Both networks were trained and evaluated on 336×1080 input image resolution.

The best performance is achieved for the Box Cars, Hoppers, and Stack

Cars. Those types usually have unique patterns on the cars - graffiti, scratches, company names, etc.

The following figures use the continuous descriptor with EfficientNet and GeM trained on the 336×1080 image resolution to show retrieval results. Figure 4.5 shows the cars with various unique drawings. The model achieves high performance on such samples, showing that it can catch and recognize unique patterns quite well. You may also see that most of the retrieved negative samples are visually similar to the query image. This means that the model clusters the images by their similarity.

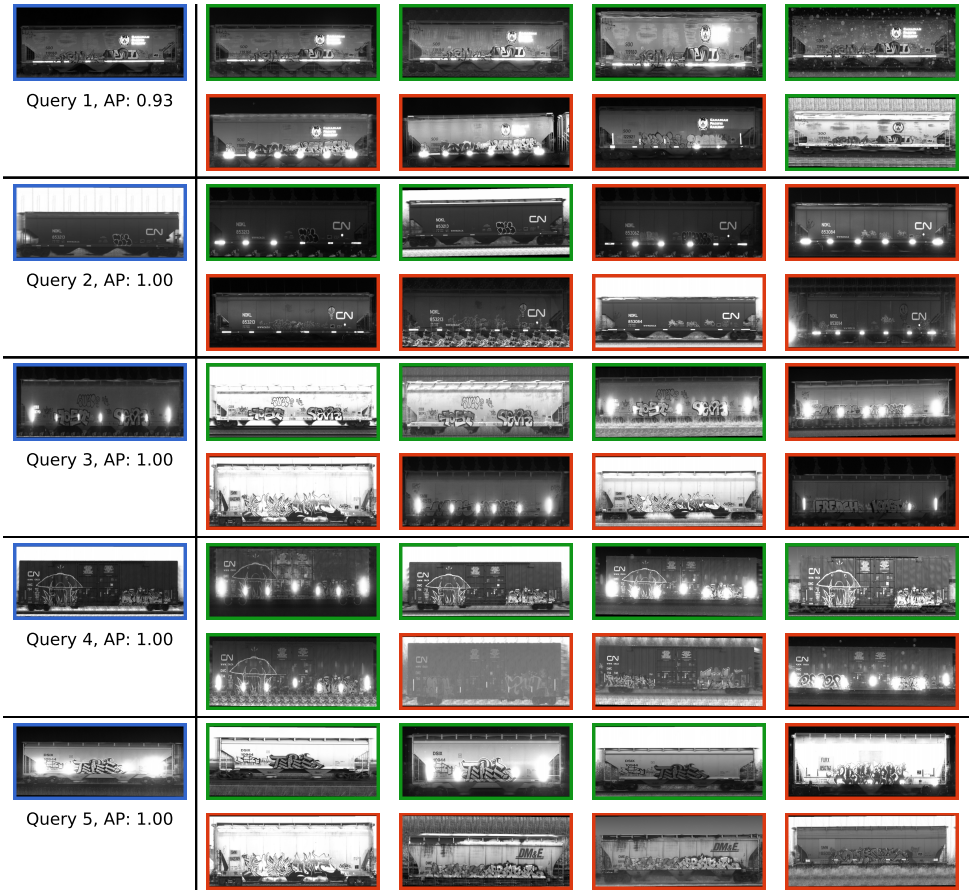



**Figure 4.5:** Example of top-8 retrieval results for query car images with unique drawings and graffiti. The retrieved samples are sorted by the euclidian distance from the query descriptor from top to bottom and left to right.

Figure 4.6 includes samples for the Stack Cars. Stack Cars are usually loaded with shipping containers that often have written shipping company names, which the descriptor can capture.

In figure 4.7 you can see cars with written company names that are not unique. In such cases, it is more difficult for the model to find the cars from the same instance because many samples have this discriminative feature but differ only with an identification number. It was also observed that query

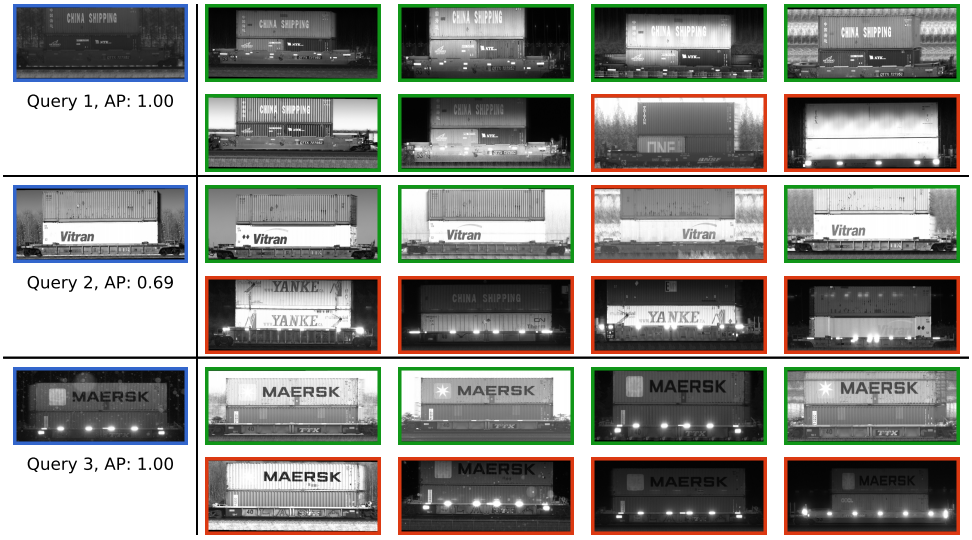



**Figure 4.6:** Example of top-8 retrieval results Stack Cars. The retrieved samples are sorted by the euclidian distance from the query descriptor from top to bottom and left to right.

and top retrieved images are often captured at different daytimes, indicating that the models are not very lighting dependent, as it was mentioned in 4.4.2.

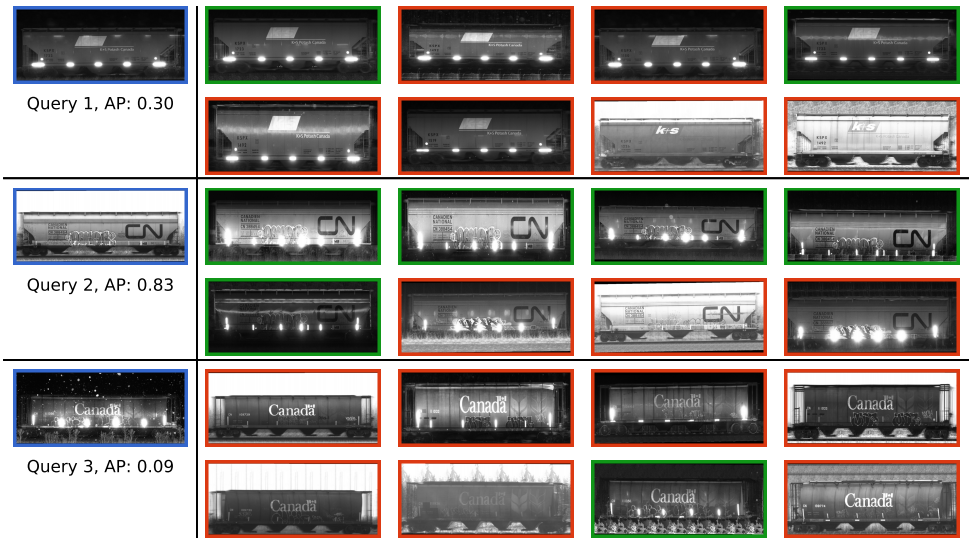



**Figure 4.7:** Example of top-8 retrieval results queries non-unique with company names. The retrieved samples are sorted by the euclidian distance from the query descriptor from top to bottom and left to right.

The worst metrics are observed for Locomotives and Flatcars. Figures 4.8 and 4.9 show the retrieval for those types where the model mostly fails. These samples do not have any easily recognizable patterns. Moreover, those car types are less represented than others, as it was shown in 3.1. Hence, it is very hard for the descriptor to find a unique representation.

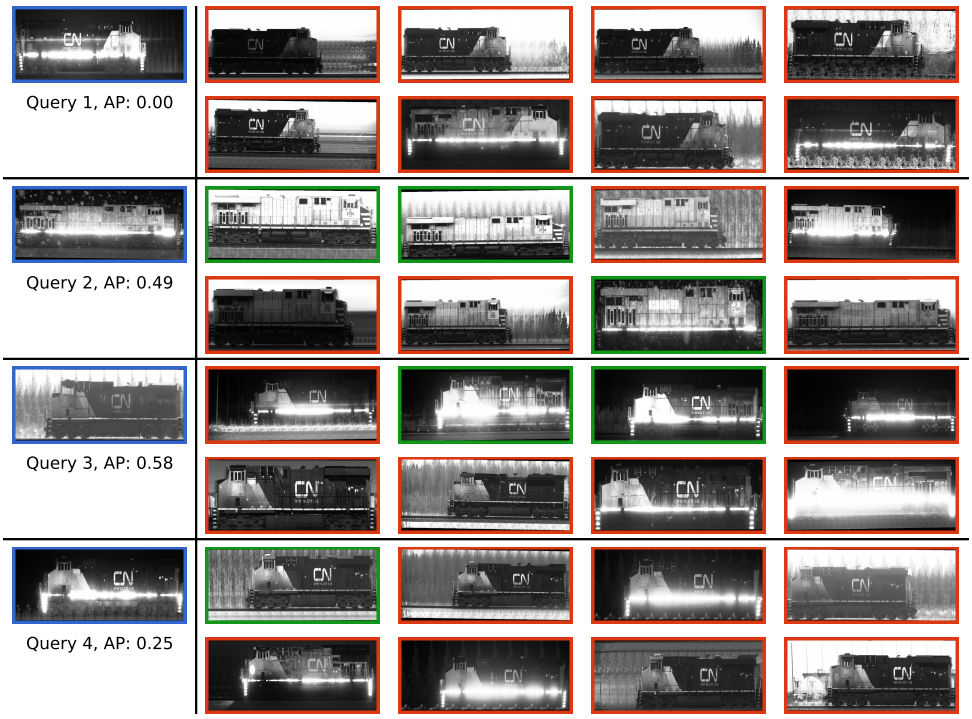



**Figure 4.8:** Example of top-8 retrieval results for Locomotives. The retrieved samples are sorted by the euclidian distance from the query descriptor from top to bottom and left to right.

Figure 4.8 also shows that the model can distinguish train directions well. All the retrieved Locomotives are captured from the same side as the query.

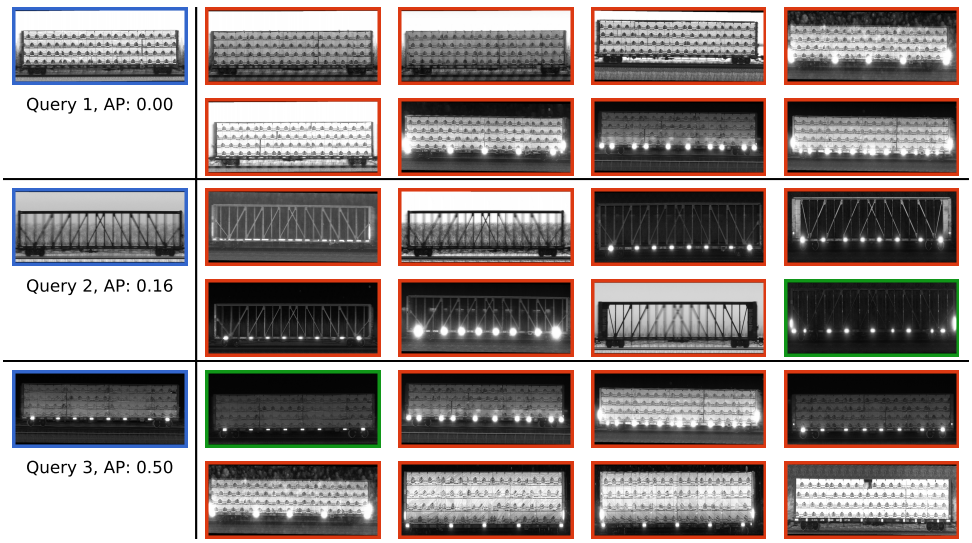



**Figure 4.9:** Example of top-8 retrieval results for Stack Cars. The retrieved samples are sorted by the euclidian distance from the query descriptor from top to bottom and left to right.

### 4.4.4 t-SNE Analysis

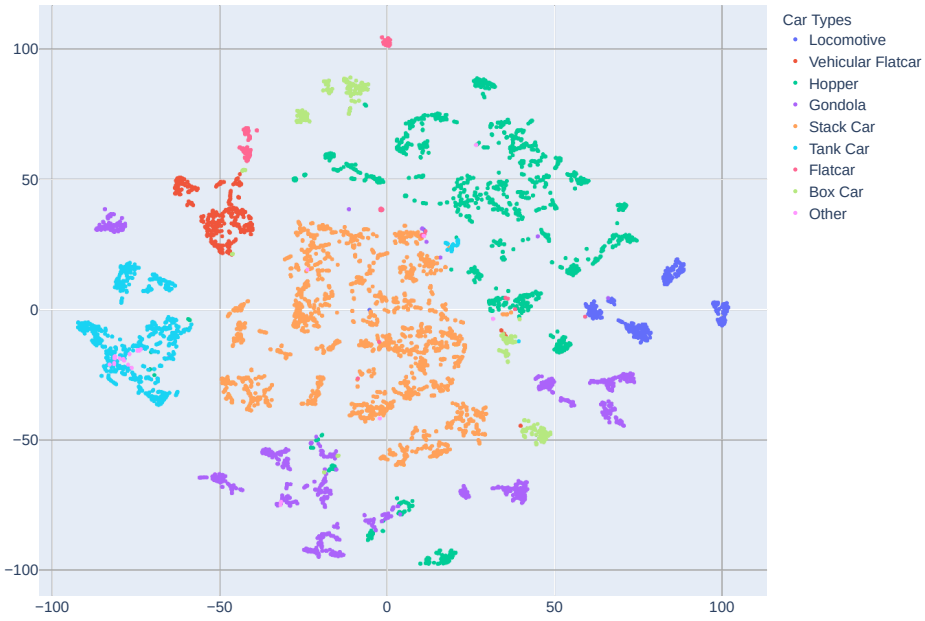
t-Distributed Stochastic Neighbor Embedding (t-SNE) is a technique for dimensionality reduction that is suitable for visualizing high-dimensional datasets [VdMH08]. This work used t-SNE to plot the high-dimensional feature vectors in a 2D plane.

The visualization of continuous and binary descriptors extracted from the test dataset images is shown in figures 4.10a and 4.10b, respectively. Feature points are colored by the corresponding car type. It is clear that both models are clustering images well since all the hash space points from one car type are grouped together. Each cluster has visible subgroups, which may correspond to similar car images across one type, e.g., cars sharing company names, cars carrying similar shipping containers, etc. However, it can be seen that the binary codes extracted with the OrthoHash layer produce a more dense visualization, which indicates that it covers a large part of the feature space. This corresponds to the concept of OrthoCos loss, which produces orthogonal targets for each instance and maximizes the similarity between binary codes and their targets. In the case of codes produced by OrtoHash, the features are clustered well not only class-wise but also instance-wise, meaning that it groups together images from one instance, which is also observed on search-based metrics (tab. 4.4).

## 4.5 Inference Speed

This work also aimed to evaluate the inference speed of the models. For fair benchmarks, models are exported to the ONNX format [BLZ+19]. ONNX is an open format for machine learning models, which allows the interchange of models between various tools and frameworks. The exported models can be further loaded to the NVIDIA Triton inference server [NVIb]. Triton software package includes a performance analyzing tool measuring the inference performance of the models. The application generates inference requests to the model and measures the throughput and latency of those requests. The performance is measured over a time window to get representative results, and then the measurements are repeated until the stable values.

Table 4.8 shows the performance of the models converted to ONNX. Due to issues connected with GeM implementation, the models were converted with a fixed batch and input size. Therefore, the experiments were run with a batch size equal to one. The requests included random data with a fixed input shape. All the experiments were run on the NVIDIA Tesla T4 GPU. While evaluating the inference speed of an OrthoHash model, there is no need to calculate logits with a cosine similarity module used only




**(a) :** Continuous codes produced by EfficientNet+GeM


**(b) :** Binary codes produced by EfficientNet+GeM+OH

**Figure 4.10:** t-SNE visualization of continuous and binary descriptors of test dataset images. Features are classified by car type. Both models used 336×1080 input resolution.

for training the hash codes. Therefore, that module was ignored in the following experiments.

| Architecture | Input | Throughput (infer/sec) | Latency (msec) |
|---|---|---|---|
| ResNet50+GeM | 224×720 | 84.4 | 11.8 |
| ResNet50+GeM+OH | 224×720 | 82.7 | 12.2 |
| EfficientNet+GeM | 224×720 | 82.2 | 12.1 |
| EfficientNet+GeM+OH | 224×720 | 81.1 | 12.3 |
| ResNet50+GeM | 336×1080 | 44.3 | 22.6 |
| ResNet50+GeM+OH | 336×1080 | 44.1 | 22.6 |
| EfficientNet+GeM | 336×1080 | 40.3 | 24.7 |
| EfficientNet+GeM+OH | 336×1080 | 39.6 | 25.2 |

**Table 4.8:** Inference speed performance of the models converted to ONNX format, computed by performance analyzing tool via Triton inference server.

It was observed that the models with the EfficentNet backbone have a slightly worse inference speed than those with a ResNet. This is unexpected behavior since the EfficientNet B2 has a lower number of parameters and FLOPS, as it was mentioned in section 2.4.2. One of the explanations could be an inefficient implementation of the EfficientNet depth-wise convolutions in the PyTorch framework [PGM+19]. The models without hashing layer were also exported to the NVIDIA TensorRT format [NVIa] (see tab. 4.9). Unlike ONNX, TensorRT also performs several types of optimization, which may accelerate the models significantly. It includes hardware-specific optimization that selects the best layers and algorithms based on the target GPU platform. In this case, the EfficientNet-based models achieved better inference performance. However, the speed difference does not correspond to the model's number of parameters and FLOPS. Thus, the issue requires further study.

| Architecture | Input | Throughput (infer/sec) | Latency (msec) |
|---|---|---|---|
| ResNet50+GeM | 224×720 | 106.2 | 9.4 |
| EfficientNet+GeM | 224×720 | 122.1 | 8.1 |
| ResNet50+GeM | 336×1080 | 44.2 | 22.6 |
| EfficientNet+GeM | 336×1080 | 58.3 | 17.1 |

**Table 4.9:** Inference speed performance of the selected models converted to TensorRT format, computed by performance analyzing tool via Triton inference server.

## 4.6 Final Model

After all the experiments, the EfficientNet B2 backbone with GeM pooling and OrthoHash layer with 2048-dimensional binary output and 336×1080 input resolution was chosen as the final model due to its highest performance, reasonable computational cost, and efficient memory use. The experiments also showed that the model can be scaled by the number of hash bits or input resolution according to memory or speed requirements. The lower resolution of 224×720 may increase the inference speed almost 2x by the accuracy cost of around 3%. The lower number of hashing bits, for example, 256, may reduce the feature size 8 times, with an accuracy drop of approximately 5%.

# Chapter 5

# Conclusion

This work built a system for accurate and memory-efficient representation of train car images. The proposed method integrates novel network architectures and techniques. The model was trained and tested on the custom datasets gathered without human annotation with tools provided by Railstate LLC. The study analyzed the effects of different model parameters and techniques to find the most suitable configuration. The final method achieves extremely high accuracy with a mean average precision of 91.3 and 86.7 on the collected validation and test datasets, respectively. Whereas the model only occupies 256 bytes per image. The benchmarks showed that the best model reaches real-time inference speed allowing the processing of more than 30 cars per second on consumer-grade GPU. The results of this work achieved high performance and met all the requirements; thus, the designed method can be integrated into the real system.

## 5.1 Future Work

The model in this work was trained on a limited amount of data. However, it is possible to collect large-scale datasets and re-train the model on a much wider variety of train images. The data samples can be selected for specific car types on which the model has lower performance (see sec. 4.4.3). After the model produced in this work is integrated into the real system, it may improve the matching algorithm and therefore improve the data collection procedure.

Many recent works study visual transformers (ViT) for image representation, which achieve promising performance [DSC21]. A more efficient backbone may improve the accuracy of a hashing pipeline. The framework implemented during this work allows easy integration of a new architecture.

Meanwhile, the ResNet and EfficientNet backbones should be re-trained using the weights pre-trained on the same datasets to better understand

the difference and the superiority of EfficientNet. Additionally, the models for continuous representation could also be re-trained with the triplet loss function (sec. 2.8.1), which may produce different results. Besides, the issue of the lower inference speed of the EfficientNet-based models should also be studied in future work.

The data augmentation process usually plays an important role in convolutional neural network training and helps the model to generalize the data better. This technique may extend the datasets by applying different image-processing algorithms used with some probability. In terms of this work, augmentation may include brightness and contrast changes, Gaussian blur and noise, or slight random shifts or scales. However, the horizontal flip, often used in classification tasks, can not be used since the direction of the train cars matters.

The work of [Alb20] showed that Generative Adversarial Network (GAN) could transform images from day to night and vice versa to extend the dataset. This may help the network to extract features invariant to lighting.

# Appendix A

## Bibliography


[AGT+15] Relja Arandjelović, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic, *Netvlad: Cnn architecture for weakly supervised place recognition*, 2015.

[Alb20] Möhwald Albert, *Data augmentation by image-to-image translation for image retrieval*, 2020.

[BL15] Artem Babenko and Victor Lempitsky, *Aggregating deep convolutional features for image retrieval*, 2015.

[BLZ+19] Junjie Bai, Fang Lu, Ke Zhang, et al., *Onnx: Open neural network exchange*, `https://github.com/onnx/onnx`, 2019.

[BSCL14] Artem Babenko, Anton Slesarev, Alexandr Chigorin, and Victor Lempitsky, *Neural codes for image retrieval*, Computer Vision – ECCV 2014 (Cham) (David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, eds.), Springer International Publishing, 2014, pp. 584–599.

[CAS20] Bingyi Cao, Andre Araujo, and Jack Sim, *Unifying deep local and global features for image search*, European Conference on Computer Vision, Springer, 2020, pp. 726–743.

[CLW+22] Wei Chen, Yu Liu, Weiping Wang, Erwin M Bakker, Theodoros Georgiou, Paul Fieguth, Li Liu, and Michael S Lew, *Deep learning for instance retrieval: A survey*, IEEE Transactions on Pattern Analysis and Machine Intelligence (2022).

[CLWY17] Zhangjie Cao, Mingsheng Long, Jianmin Wang, and Philip S. Yu, *Hashnet: Deep learning to hash by continuation*, 2017.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li, *Imagenet: a large-scale hierarchical image database*, 2009 IEEE conference on computer vision and pattern recognition, 2009, pp. 248–255.

[DSC21] Shiv Ram Dubey, Satish Kumar Singh, and Wei-Ta Chu, *Vision transformer hashing for image retrieval*, 2021.

[FNJ+20] Lixin Fan, Kam Woh Ng, Ce Ju, Tianyu Zhang, and Chee Seng Chan, *Deep polarized network for supervised learning of accurate binary hashing codes*, Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20 (Christian Bessiere, ed.), International Joint Conferences on Artificial Intelligence Organization, 7 2020, Main track, pp. 825–831.

[FVJ09] P. Thomas Fletcher, Suresh Venkatasubramanian, and Sarang Joshi, *The geometric median on riemannian manifolds with application to robust atlas estimation*, NeuroImage **45** (2009), no. 1, Supplement 1, S143–S152, Mathematics in Brain Imaging.

[GARL16] Albert Gordo, Jon Almazan, Jerome Revaud, and Diane Larlus, *Deep image retrieval: Learning global representations for image search*, 2016.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, MIT press, 2016.

[HMvdW+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant, *Array programming with NumPy*, Nature **585** (2020), no. 7825, 357–362.

[HNZ+21] Jiun Tian Hoe, Kam Woh Ng, Tianyu Zhang, Chee Seng Chan, Yi-Zhe Song, and Tao Xiang, *One loss for all: Deep hashing with a single cosine similarity based learning objective*, Advances in Neural Information Processing Systems (NeurIPS), 2021.

[Hun07] John D Hunter, *Matplotlib: a 2d graphics environment*, Computing in science & engineering **9** (2007), no. 3, 90–95.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Deep residual learning for image recognition*, 2015.

[Inc15] Plotly Technologies Inc., *Collaborative data science*, `https://plot.ly`, 2015.

[Jar21] Jekatěrina Jaroslavceva, *Image retrieval via cnns in tensorflow2*, 2021.

[JC12] Hervé Jégou and Ondrej Chum, *Negative evidences and cooccurrences in image retrieval: the benefit of PCA and whitening*, ECCV - European Conference on Computer Vision (Firenze, Italy), October 2012.

[JL18] Qing-Yuan Jiang and Wu-Jun Li, *Asymmetric deep supervised hashing*, Proceedings of the AAAI conference on artificial intelligence, vol. 32, 2018.

[KB14] Diederik P. Kingma and Jimmy Ba, *Adam: A method for stochastic optimization*, 2014.

[KH+09] Alex Krizhevsky, Geoffrey Hinton, et al., *Learning multiple layers of features from tiny images*.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, *Imagenet classification with deep convolutional neural networks*, Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (Red Hook, NY, USA), NIPS'12, Curran Associates Inc., 2012, p. 1097–1105.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár, *Microsoft coco: Common objects in context*, 2014.

[Low99] D.G. Lowe, *Object recognition from local scale-invariant features*, Proceedings of the Seventh IEEE International Conference on Computer Vision, vol. 2, 1999, pp. 1150–1157 vol.2.

[LWSC16] Haomiao Liu, Ruiping Wang, Shiguang Shan, and Xilin Chen, *Deep supervised hashing for fast image retrieval*, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 2064–2072.

[LWW+20] Xiao Luo, Haixin Wang, Daqing Wu, Chong Chen, Minghua Deng, Jianqiang Huang, and Xian-Sheng Hua, *A survey on deep hashing methods*, ACM Transactions on Knowledge Discovery from Data (TKDD) (2020).

[nep22] neptune.ai, *Neptune: experiment tracking and model registry*, `https://neptune.ai`, 2022.

[NVIa] NVIDIA, *Tensorrt*, `https://developer.nvidia.com/tensorrt`.

[NVIb] _____, *Triton inference server*, `https://developer.nvidia.com/nvidia-triton-inference-server`.

[OEOI14] Adaku Obikee, Godday Ebuh, and Happiness Obiora-Ilouno, *Comparison of outlier techniques based on simulated data*, Open Journal of Statistics **04** (2014), 536–561.

[pdt20] The pandas development team, *pandas-dev/pandas: Pandas*, https://doi.org/10.5281/zenodo.3509134, February 2020.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala, *Pytorch: An imperative style, high-performance deep learning library*, http://arxiv.org/abs/1912.01703, 2019, cite arxiv:1912.01703Comment: 12 pages, 3 figures, NeurIPS 2019.

[RIT+18] Filip Radenović, Ahmet Iscen, Giorgos Tolias, Yannis Avrithis, and Ondřej Chum, *Revisiting oxford and paris: Large-scale image retrieval benchmarking*, 2018.

[RSMC14] Ali Razavian, Josephine Sullivan, Atsuto Maki, and Stefan Carlsson, *A baseline for visual instance retrieval with deep convolutional networks*, ITE Transactions on Media Technology and Applications **4** (2014).

[RTC18] Filip Radenović, Giorgos Tolias, and Ondřej Chum, *Fine-tuning CNN image retrieval with no human annotation*, TPAMI (2018).

[SKP15] Florian Schroff, Dmitry Kalenichenko, and James Philbin, *FaceNet: A unified embedding for face recognition and clustering*, 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, jun 2015.

[SZ03] Sivic and Zisserman, *Video google: a text retrieval approach to object matching in videos*, Proceedings Ninth IEEE International Conference on Computer Vision, 2003, pp. 1470–1477 vol.2.

[SZ14] Karen Simonyan and Andrew Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2014.

[TL19] Mingxing Tan and Quoc Le, *EfficientNet: Rethinking model scaling for convolutional neural networks*, Proceedings of the 36th International Conference on Machine Learning (Kamalika Chaudhuri and Ruslan Salakhutdinov, eds.), Proceedings of Machine Learning Research, vol. 97, PMLR, 09–15 Jun 2019, pp. 6105–6114.

[TSJ15] Giorgos Tolias, Ronan Sicre, and Hervé Jégou, *Particular object retrieval with integral max-pooling of cnn activations*, 2015.

[VdMH08] Laurens Van der Maaten and Geoffrey Hinton, *Visualizing data using t-sne.*, Journal of machine learning research **9** (2008), no. 11.

[WACS20] Tobias Weyand, Andre Araujo, Bingyi Cao, and Jack Sim, *Google landmarks dataset v2 – a large-scale benchmark for instance-level recognition and retrieval*, 2020.

[WsL+14] Jiang Wang, Yang song, Thomas Leung, Chuck Rosenberg, Jinbin Wang, James Philbin, Bo Chen, and Ying Wu, *Learning fine-grained image similarity with deep ranking*, 2014.

[XPL+14] Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan, *Supervised hashing for image retrieval via image representation learning*, Proceedings of the AAAI Conference on Artificial Intelligence **28** (2014), no. 1.

[YWZ+20] Li Yuan, Tao Wang, Xiaopeng Zhang, Francis EH Tay, Zequn Jie, Wei Liu, and Jiashi Feng, *Central similarity quantization for efficient image and video retrieval*, Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 3083–3092.