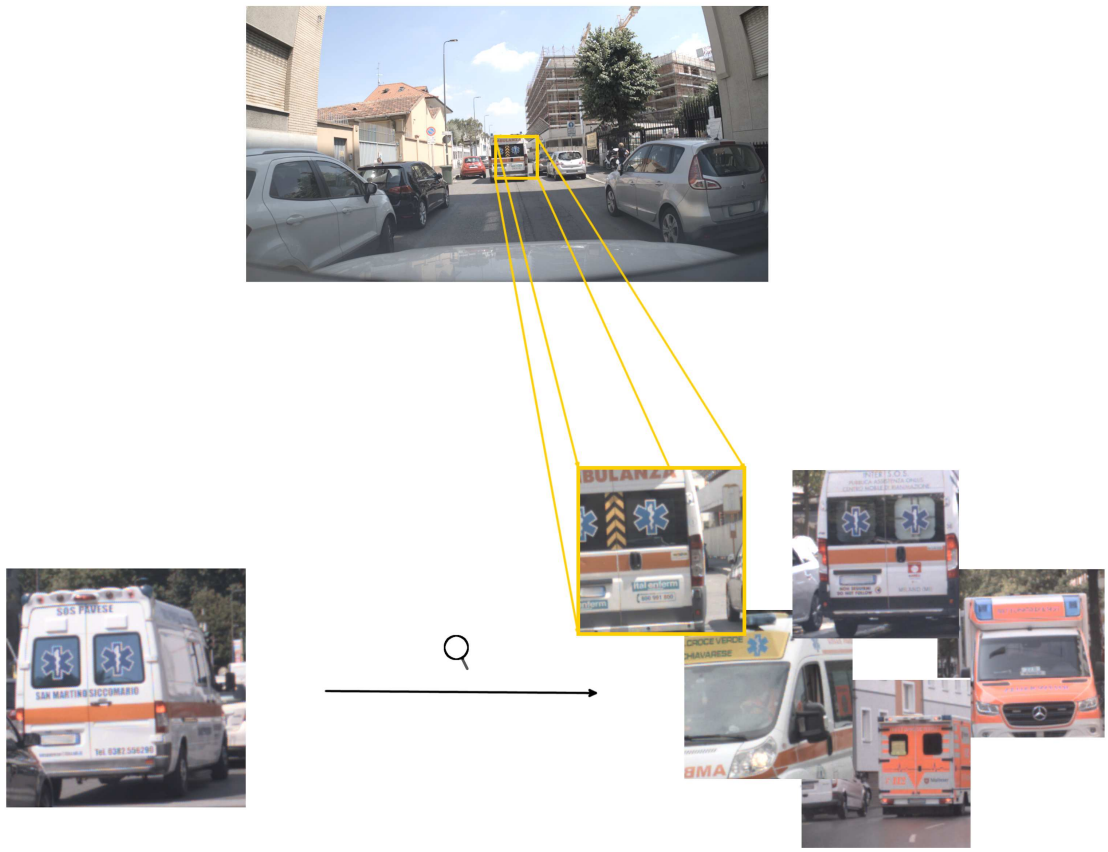


Patch-wise image similarity search

Searching for small regions in collections of large images

Master's thesis in Complex Adaptive Systems

Christoffer Arvidsson
Ebba Davidsson

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Patch-wise image similarity search

Searching for small regions in collections of large images

CHRISTOFFER ARVIDSSON
EBBA DAVIDSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2022

Patch-wise image similarity search
Searching for small regions in collections of large images
Christoffer Arvidsson
Ebba Davidsson

© Christoffer Arvidsson, Ebba Davidsson, 2022.

Supervisors: Erik Werner & Niklas Gustafsson, Zenseact
Examiner: Bernhard Mehlig, Department of Physics, Gothenburg University

Master's Thesis 2022
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Example of found patches when searching with a patch of an ambulance.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Patch-wise image similarity search
Searching for small regions in collections of large images
Christoffer Arvidsson
Ebba Davidsson
Department of Physics
Chalmers University of Technology

Abstract

In autonomous driving it is important that a neural network performs well even for examples that do not occur often in the dataset. One method to improve performance is to find and add examples similar to those the network struggles with, thereby increasing the training data available. Similarity search is an automatic method for searching large datasets for the most similar examples to some target. This thesis describes a similarity search algorithm for locating different sized objects in a large dataset of high-resolution images. The algorithms uses patches, which are small regions in an image, in order to enable precise searches for small objects. Each patch is embedded to 512 dimensional vectors with CLIP[1] that captures the semantic meaning of the content in the patch. The main contribution of this thesis is a method to reduce the potential number of patches resulting from each image, by selecting the most visually interesting patches in each image. We evaluate the combined patch selection and similarity search on three classes of objects relevant for autonomous driving: ambulances, animals and a specific traffic sign marking an upcoming road narrowing, to measure the fraction of relevant patches retrieved. Further, we show that searching with the average vector representation of several images of the same object improves the result, while searching with a text string gives varying results depending on the object class.

Keywords: thesis, nearest-neighbor search, deep learning, image retrieval.

Acknowledgements

We would like to thank Zenseact for enabling us to write this thesis by both supporting us during the project and supplying us with data and resources. Our industrial supervisors Erik Werner and Niklas Gustafsson at Zenseact have been a great source of ideas and feedback, along with the rest of Team Valdag. We would also like to thank Bernhard Mehlig for his continuous support during the thesis project.

Christoffer Arvidsson, Ebba Davidsson, Gothenburg, June 2022

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

ANN	Approximate Nearest-Neighbor
CNN	Convolutional Neural Network
CLIP	Contrastive Language-Image Pre-training
k-NN	k-Nearest-Neighbor
PQ	Product quantization
OPQ	Optimized Product Quantization

Contents

List of Figures

List of Tables

1
Introduction

With the growth of large scale data collection, deep neural networks have become popular in autonomous driving systems for their efficiency in perception tasks such as object detection [2]. To train these networks one first collects a large amount of data, then selects a subset of this large dataset as a training set. This training set should be representative of the whole dataset, and usually requires additional annotation and data processing before training the network.

It is important for the perception model to perform well on examples that rarely occur, for example rare objects or objects in odd environments. To improve the models performance, it is appropriate to extend the training set with examples similar to these rare cases. However this introduces a problem, namely how to quickly find similar examples. It is not feasible to manually search for these examples due to the size of the large dataset and the scarcity of these examples. For these reasons, other methods are required to perform these searches.

1.1 Background

Supervised learning can be used to learn the mapping between predictor-target pairs. These pairs are called training examples and make up a training set that should be large and representative enough to allow the model to generalize to unseen examples.

There can be multiple reasons for why a model does not generalize, for example due to the complexity and architecture of the model [3], the method used to train the model, or the training set used. If it is presumed that the training set is the issue, then active learning can be used to detect and extend the training data with examples the model is performing badly at [4].

In autonomous driving these examples include unexpected objects on the road, animals that blend into the background, rare objects such as a highly specific traffic sign, or very small objects that may be difficult to detect even for a human. To improve performance in such scenarios a good starting point is to add more of them to the training set, thereby increasing coverage of the model.

Given an that we have found an example the model is performing badly at, one might be able to find additional cases by searching for similar cases in the collected data. It is unfeasible to manually search a large dataset for many matching cases, so a human may use metadata such as the time of day and location a photo was taken to narrow down the search space [5]. However this metadata is often not descriptive

enough to enable precise searches such as for specific objects in certain conditions.

For this reason, other methods exist that do not require any metadata. One such method is similarity search which uses a similarity metric to compare pairs of images, creating a score for how similar the two images are [6]. This is known as content-based image retrieval, where the content of images are compared during the search rather than metadata. Although it is possible to compare images in this way, it is more efficient to compare feature vectors extracted from each image as the representation is more compact [7, 8].

1.2 Problem statement

This thesis describes a method for performing content-based image retrieval on a large dataset consisting of unlabeled high-resolution images. The system may be used by humans to search this dataset for similar regions, either via text or region targets. The images searched among are taken with a front-facing camera on a vehicle in traffic and include a variety of environments, weather conditions and time of day. The resolution of each image is 3848×2168 pixels.

Because these are high-resolution images they contain a lot of information and would therefore not be specific enough to search with. Instead it may be possible to search with smaller regions, called patches. This should allow the search to find for example a moose on the road, which occupies only a tiny fraction of the full image. We distinguish between a *query patch* as the patch we would like to find similar patches to, and patches that we search among.

Considering all the possible sizes and locations of a patch, there are an immense number of possible patches in just a single image. Similarity search requires storing a feature vector for every single patch, which will be prohibitively costly both in terms of memory and time. To mitigate the number of patches, we wish to only keep patches with interesting content. For autonomous driving we are for example interested in regions with objects, pedestrians, traffic signs and rare vehicles. Less useful regions are those with trees, clouds and parts of buildings. By finding these interesting regions, it should be possible to reduce the total number of patches.

We foresee a few challenges in developing an image retrieval system that works on a patch level

Patch selection Extracting interesting patches from an image while ignoring regions with little information,

Semantic embedding creation Extracting features from patches that allow pairwise vector comparison for patch similarity,

Similarity search Searching for patches by vector similarity and returning the images each found patch originally came from.

In addition to these challenges, memory usage is an issue due to the number of images in the large dataset of collected images, and subsequently the number of patches that will be selected. Part of the problem under consideration is to reduce this memory usage so that the method can be applied to very large datasets.

1.2.1 Aim

The aim of this thesis project is to develop an image retrieval system that achieves high-quality searches on a large dataset of unlabelled images. A high-quality search is defined as the system retrieving a large fraction of images relevant to the search query. Additionally, the system should make use of patches in the images to further increase precision for small object searches.

While the system will be used on large unlabelled data, it is convenient to evaluate the system on images with corresponding object annotations, as this allows evaluating whether a result actually included the object searched for. The system will not make use of annotations unless it is for evaluation purposes.

With the addition of many patches per image, we also hope that the memory usage remains feasibly low for large collections of images. We hypothesis that quantization and filtering methods can be used to significantly reduce the memory footprint without large loss of search accuracy.

1.2.2 Scope and limitations

We chose to focus on searches for objects that would be relevant for autonomous driving, such as finding certain types of vehicles or certain types of traffic signs. The system can certainly be used for other kinds of data, but this is not the focus of the project.

We also limit the project to use existing similarity search libraries, since these have shown high query performance and have more efficient implementations than what could be produced in the time span of the project.

2

Theory

This chapter provides a concise overview of the ideas and concepts necessary to understand the methodology and results of this thesis. We begin by describing image retrieval systems, followed saliency prediction and text-image feature extraction. Finally we describe approximate nearest neighbor search.

2.1 Image retrieval systems

An image retrieval system is a type of information retrieval system that aims to retrieve images given a search query [5]. The purpose of these systems is to recommend a set of relevant images to a user, where relevancy depends on the query used. The query could for example be a specific attribute value, a text string or another image [9]. Depending on the complexity of the task, the retrieval system could involve anything from a simple checkup among the attributes of each image, or be based on features extracted from an image.

One method to find the most relevant images is to use a similarity score. This score measures how similar two elements are and can then directly be used to find the most similar elements. One measure for this is the L_2-distance defined as

$$L_2\text{-distance}(\mathbf{q}, \mathbf{e}) = \sqrt{\sum_{i=1}^{n}(q_i - e_i)^2}, \tag{2.1}$$

which is the length of the difference vector between the two vectors $\mathbf{q}$ and $\mathbf{e}$. More similar vectors have a smaller L_2-distance.

Another alternative is the *cosine similarity*. It measures similarity between two vectors as the cosine of the angle θ between the vectors. It is defined as

$$\text{cosine-similarity}(\mathbf{q}, \mathbf{e}) = \cos\theta = \frac{\mathbf{q} \cdot \mathbf{e}}{\|\mathbf{q}\|\|\mathbf{e}\|} = \frac{\sum_{i=0}^{n} q_i e_i}{\sqrt{\sum_{i=0}^{n} q_i^2}\sqrt{\sum_{i=0}^{n} e_i^2}} \tag{2.2}$$

and takes on values in the interval $[-1, 1]$, where -1 represents opposite vectors, 0 for two orthogonal vectors, and 1 for two aligned vectors. If $\mathbf{q}$ and $\mathbf{e}$ are normalized, then the dot product is equal to the cosine similarity. For similarity search, a larger cosine similarity is better.

2.1.1 Precision metric

Image retrieval systems can be evaluated using image metadata such as objects contained inside the image or categories [5]. A query might regard a certain type of object, and relevant images would then include that type of object.

An important metric is the percentage of retrieved images relevant to the query (precision). *Precision at k* (precision@k) describes the fraction of relevant images among the k retrieved. It is computed as

$$\text{precision@}k = \frac{|\text{relevant images} \cap \text{retrieved images}|}{k}. \tag{2.3}$$

2.2 Saliency prediction

To select only a few patches in an image while still covering the interesting objects, one needs to select the patches strategically. This strategy refers to aligning the patches with regions which are likely to include interesting objects. To do this, one needs a metric for distinguishing the more interesting parts of the image from uninteresting parts. One way of measuring this is by using saliency prediction.

Saliency prediction refers to predicting the areas of visual attention in a scene. The prediction is usually in the form of a black and white saliency map, with a greater pixel value in regions of greater visual interest. Estimating regions of visual attention has long been an interesting area. The earliest models mostly consisted of handcrafted feature models, where the estimations were made based on contrast, color and texture [10]. While these implementations were useful in some cases, the accuracy was limited. As more advanced techniques have emerged, the usage of deep learning has proven especially applicable for this task.

The deep learning models are commonly supervised, meaning they depend on some sort of ground truth data, where areas are labeled as more or less salient. For larger datasets, this ground truth data is often object based and mark the placements of prominent objects as salient. Another alternative is to record the mouse movements over images as humans view them online. This has proven to be highly correlated to visual attention and has thereby enabled the enlargement of relevant ground truth datasets. Some other datasets are collected from recording the eye movements of humans looking at the specific images [11]. This type of ground truth data is the most accurate for saliency networks, though it is costly to collect and such datasets therefore tend to be small.

2.2.1 SimpleNet

One recent implementation of a saliency prediction network is SimpleNet. It consists of a fully convolutional **encoder-decoder** UNet architecture [12]. This network was chosen for this study based on its high performance and accessibility.

It was implemented along with the pretrained weights from the original implementation. These weights were trained on the SALICON dataset, consisting of 10,000 training, 5000 validation and 5000 test pairs of images and ground truth saliency

predictions. The original images came from the MS COCO dataset [13] and the ground truth saliency maps are based on the mouse movements of viewers. By using this instead of human eye tracking data, the size of the dataset can grow from hundreds to thousands [14]. This makes the SALICON dataset the largest saliency dataset available.

An illustration of the network is shown in Figure 2.1.

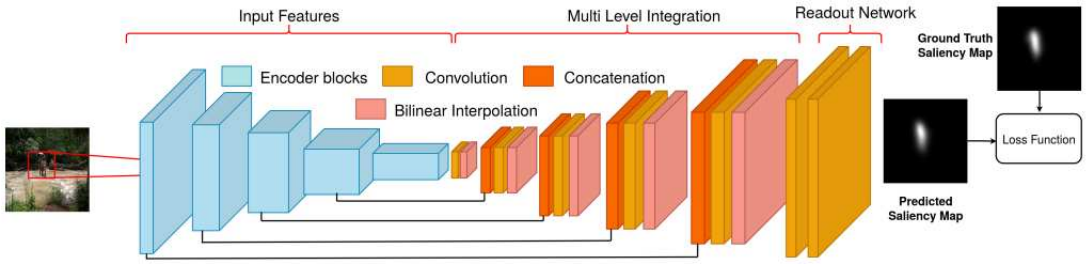
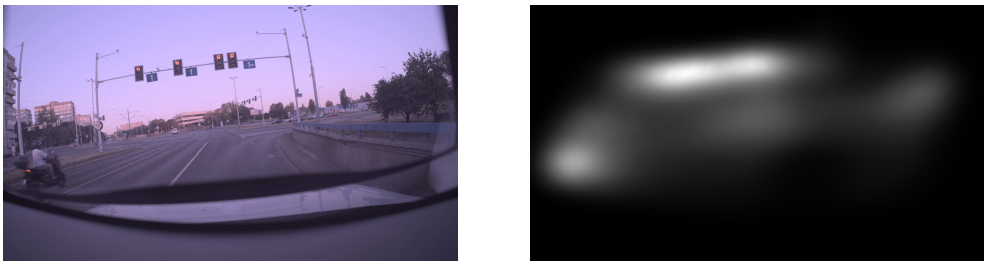


Figure 2.1: Illustration of the SimpleNet architecture for saliency. Uses a UNet architecture to encode and decode an image into a saliency map. Image source [15].

The model takes a 256x256 image as input and returns a saliency prediction of the same size. Figure 2.2 shows the re-scaled saliency prediction of an image.



(a) original image **(b)** saliency prediction

Figure 2.2: Illustration of the saliency prediction of an image. The regions with traffic lights and motorcycle have high predicted saliency, while sky and the road have less predicted saliency.

2.3 Contrastive Language-Image Pretraining

Feature extraction is the process of transforming data into descriptive numerical values. These numerical values form a feature vector that captures the information in the data required to perform some predefined task [16]. In our case, the data are pixel values of patches, and the vectors describe the contents of the patch. These vectors will then be used for similarity comparisons during a search.

The feature vectors need to e locality sensitive such that similar vectors are closely placed in the embedding space. Although there are classical methods for extracting

features from both text and images, they are often outperformed by newer deep learning approaches[17].

Contrastive Language-Image Pretraining (CLIP) [1] is a recent architecture trained to connect images and natural language. As an example, a picture of a dog could be described by the text "dog" or a class label as is common in many classification datasets. A better description might include what the dog is doing, such as "a photo of a dog jumping in a field of grass". If the model is capable of connecting the image with this text, then it shows an understanding of how language relates to visual concepts. What then exists inside the model is a good representation for describing the image or text.

To create this representation, CLIP receives minibatches of N image-text pairs. There are N^2 possible pairings, N of which are true and $N^2 - N$ of which are false. Each image and text is embedded to a vector in shared 512-dimensional space via an image and text encoder respectively. The model is trained to minimize a contrastive loss function called the InfoNCE loss [18, 19]. Contrastive means it simultaneously pull together the true pairings while pushing apart the false pairings. The training loss involves two loss functions

$$H_i^{(I\to T)} = -\log \frac{\exp\left(\left[\mathbf{z}_i^{(I)} \cdot \mathbf{z}_i^{(T)}\right]/\tau\right)}{\sum_{j=1}^{N} \exp\left(\left[\mathbf{z}_i^{(I)} \cdot \mathbf{z}_j^{(T)}\right]/\tau\right)} \tag{2.4}$$

as an image-to-text contrastive loss function and

$$H_i^{(T\to I)} = -\log \frac{\exp\left(\left[\mathbf{z}_i^{(T)} \cdot \mathbf{z}_i^{(I)}\right]/\tau\right)}{\sum_{j=1}^{N} \exp\left(\left[\mathbf{z}_i^{(T)} \cdot \mathbf{z}_j^{(I)}\right]/\tau\right)} \tag{2.5}$$

for text-to-image for pair i. In these equations $\mathbf{z}_i$ is the normalized feature vector extracted with the text/image encoder for pair i and τ is a temperature parameter that controls the range of the logits and is learned during training. A superscript (I) or (T) signifies image and text respectively.

The final training loss is computed as the mean of these two loss functions, averaged across N pairs.

$$H = \frac{1}{2N}\sum_{i=1}^{N}(H_i^{(I\to T)} + H_i^{(T\to I)}). \tag{2.6}$$

After minimizing Eq. 2.6 via stochastic gradient descent and backpropagation, the image and text decoder can be used to create embeddings that are applicable to many different tasks, including nearest-neighbor search[20].

2.3.1 Transformers and self-attention

Transformers [21] were developed as an alternative to the then dominant sequence learning models that made use of recurrent neural networks (RNN), long short-term memory [22], or gated recurrent neural networks [23]. They are encoder-decoder architectures, i.e. the method involves first encoding the input into an efficient representation, then decoding that representation into something else dependent on

the task. One such task might be machine translation, where the text is translated to a different language. CLIP uses these as part of its text and vision encoders.

As an alternative to convolutional networks, transformers use attention weights to "attend" to different parts of the input. This is called *self-attention* because an attention weight is computed for every pair of positions in the sequence. Because the transformer can attend to anywhere in the input sequence, it can use both local and global information. The scaled dot-product attention [21] is computed according to

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2.7}$$

where d is the dimension of the representation, Q is the query vector, K is the key vector and V is the value vector.

The naming of these vectors come from retrieval systems; the user provides a query, the system maps the query against many available keys in the database, then presents the best matching value. These vectors are obtained by matrix product of the embedding and a corresponding weight matrix for each vector. The weight matrices are learnt during training.

The matrix multiplication QK^T computes a score for how related each element in the output is to every element in the sequence. Scaling by $\sqrt{d_k}$ is done to mitigate potentially large dot product results that would destabilize the gradients during backpropagation. Softmax turns the scores into fractions that sum to 1, and the matrix product with V selects parts of the input sequence proportional to the scores, creating a weighted average.

Instead of learning one generic attention weight matrix, it can be useful to learn multiple weight matrices. There are called attention heads, and they focus on different concepts by projecting the input vector into different subspaces. Multi-head attention involves computing Eq. 2.7 for multiple attention heads, then concatenating the resulting vectors. This larger vector is reduced in size by another learnt weight matrix.

2.3.2 CLIP encoders

In the version of CLIP used in this thesis, both the image and text encoders use the encoder part of the transformer architecture. Figure 2.4 show these encoders. Both of these make use of residual blocks, referred to as ResBlocks. The structure of these is shown in Figure 2.3. The residual connections help mitigate the vanishing gradient problem, where the loss signal vanishes in deep networks.

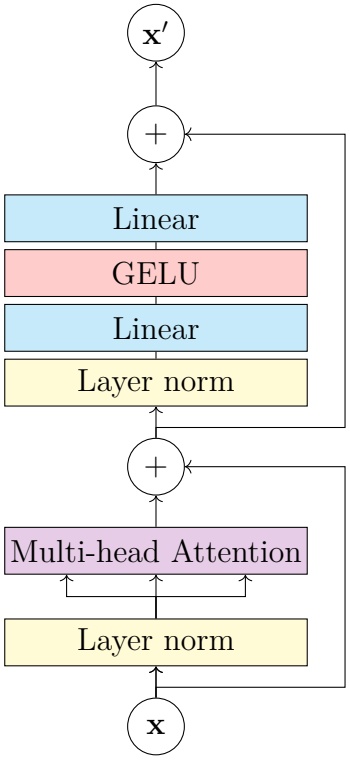
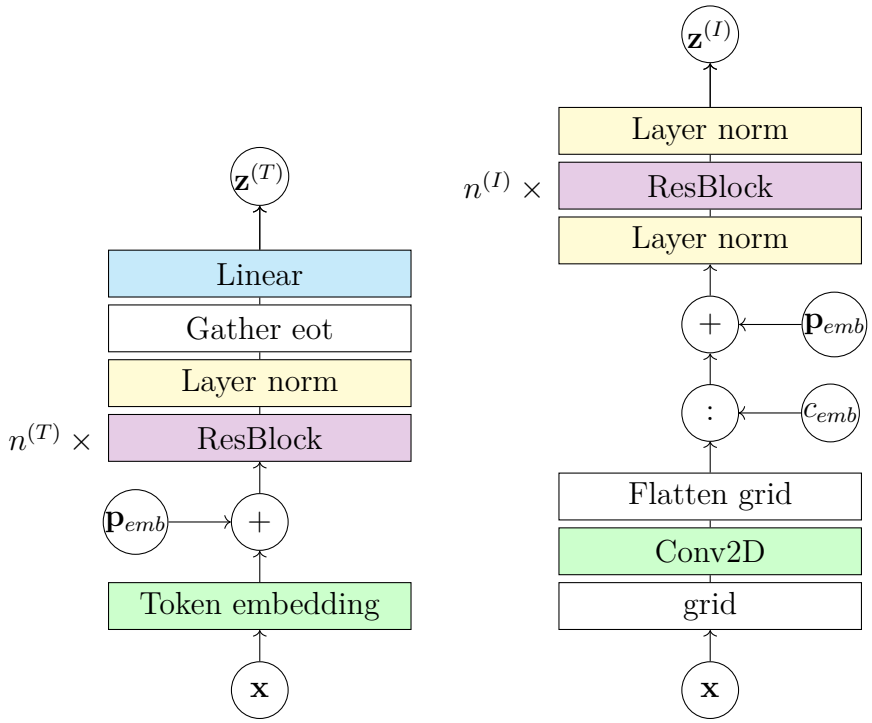


Figure 2.3: Structure of a residual attention block used in both the transformer and vision transformer in CLIP. *Layer norm* normalizes the input vector by subtracting mean and dividing by standard deviation of the batch [24] and *GELU* is the Gaussian Error Linear Unit [25].

Figure 2.4a shows the structure of the text encoder used in CLIP. An initial processing step for text is to perform Byte-pair encoding (BPE), where the most common pairs of consecutive bytes are replaced by a byte not found in the text. This compresses the input into a sequence of discrete tokens that can be embedded using a token embedding layer. The text embedding is extracted by gathering the *end-of-sentence* tokens, which act as general representations for the input text.

Figure 2.4b is a vision-transformer encoder [26]. The difference to the transformer used for text is that images are not typically treated as sequences of elements as one could do for words in a text. To create this representation, the image is split into a grid of 16×16 sized patches, then convolved individually to form embeddings. In the same way end-of-sentence tokens were used as the general representation of the text, a class token is used in vision transformers. This token is appended to the sequence of grid patch embeddings.

In contrast to RNNs or LSTMs where a sequence is processed in order, attention processes sequences all at once. Positional embeddings are added to preserve the sequence position of each token.



(a) text encoder **(b)** vision encoder

Figure 2.4: Structure of the two encoders used in CLIP. x is the input (either text tokens or an image) and z is the output embedding. The : symbol denotes concatenation and + is the add operation. *Gather eot* gathers the end-of-text tokens from each example in the minibatch. *grid* splits the image into a grid with 16×16 sized patches. The *Conv2D* layer is a 2D convolution layer, see original paper for parameter details. *Layer norm* normalizes the input vector by subtracting mean and dividing by standard deviation of the batch. Finally ResBlock takes the structure shown in Figure 2.3 and has n such blocks stacked.

2.4 Approximate nearest-neighbor search

Consider a dataset of images and a single text description of an image. The goal is to find images that match that text description. These matches should specifically be the best matching images or close to the best. Algorithms that solve this problem are called *nearest-neighbor search algorithms*.

Nearest-neighbor search algorithms find the closest vectors to a query vector $\mathbf{q}$ among a collection of vectors based on a similarity metric. The query vector is what was earlier referred to as a a text description, but it can be any vector as long as it describes the content we want to search for.

One naive method for finding the nearest-neighbors is to compute the pairwise similarity between every vector and the query, then collect the most similar k vectors as the result of the search. However this scales poorly for large datasets since $\mathbf{q}$ has to be compared to every other vector. Optimizations of this involve clever use of structuring data, avoiding many of these checks. We refer to the data structure as a

search index, which often stores more information than just the vectors in order to speed up the search. One such optimization is to use graph-based nearest-neighbor search algorithms [27, 28]. These algorithms utilize graphs to guide the search in the direction towards the query.

2.4.1 Nearest-neighbor graphs

A common graph used in nearest-neighbor search is a k-nearest-neighbor graph (k-NN graph). A k-NN graph is a directed graph connecting nodes with their k nearest neighboring nodes [29]. Figure 2.5 shows an example of a k-NN graph in two dimensions. In this graph an edge between nodes p and q exists if a given similarity measure, for example Eq. 2.2, between p and q is greater than the maximum similarity of the k closest neighbors to p.

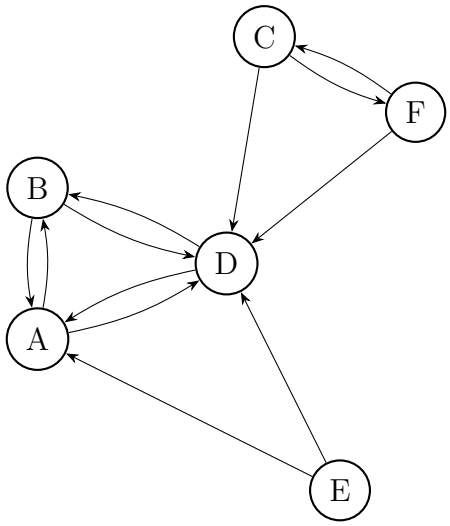


Figure 2.5: Example of a k-nearest neighbor graph where $n = 6, k = 2$. Each node has exactly $k = 2$ outgoing edges pointing to the nearest neighbors of the node.

Constructing an exact k-NN graph is computationally expensive and often not required. Instead an approximate k-NN graph can be computed by iteratively refining either a completely random graph, or a graph that is already close to a k-NN, but faster to create [7]. Refinement of this initial graph makes use of the notion that a node's neighbors' neighbor are often close to the original node. Returning to Figure 2.5, node C has the neighbor D which has the neighbor B. Even though B is not a neighbor of C, it would be a good candidate to consider as it is fairly close to C.

2.4.2 Graph-based nearest-neighbor search

Consider a k-nearest-neighbor graph such as the one shown in Figure 2.6. Given a query point q outside of the graph, and a random start node E^* in the graph, the approximate nearest-neighbors of q can be found by iteratively taking steps towards the query from the start node. In each iteration, the decision of which edge to traverse depends on what the distance to the query would be as a result from

traversing that edge. Effectively this means choosing the edge that minimizes the distance to the query.

Figure 2.6 also shows the resulting path this algorithm would find if it started at node E^*. For such a small graph, most nodes are visited to get to G. However for larger graphs, this straight line to target approach ignores the majority of nodes and is therefore very efficient.

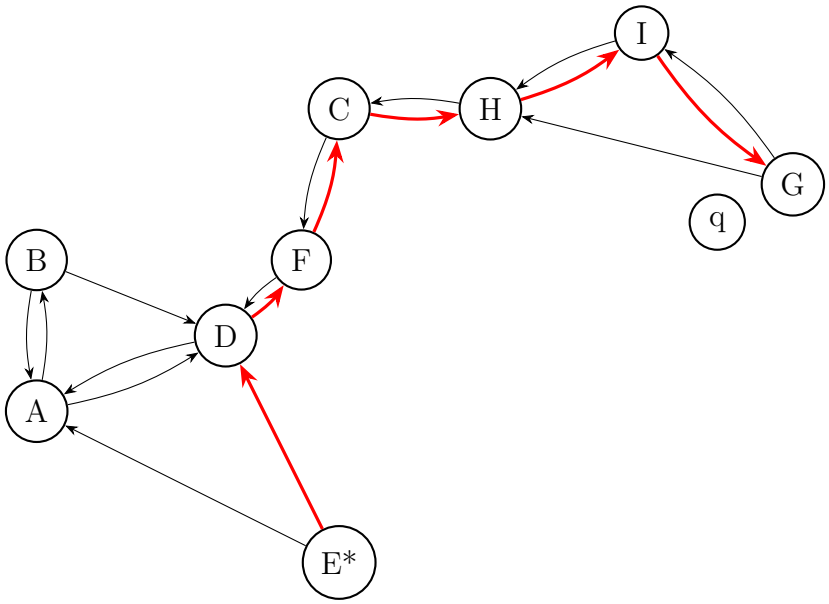


Figure 2.6: Path found by graph traversal in a k-NN graph where $n = 9, k = 2$. The * signifies the random start point in the search, q is the query point outside of the graph, and the red path (bold) is the path found where at each iteration, the edge that minimizes the distance to the query point is selected.

One issue is that this kind of greedy algorithm can easily get stuck in local minima. Consider the case where C had been slightly further to the left in Figure 2.6. Then F would be a local minimum because the distance from C to q would be greater than the distance from F to q. To alleviate such cases, one can run the algorithm many times with different starting nodes.

The full algorithm is shown in Algorithm 1 [28]. The search is run R times with different starting nodes sampled uniformly from nodes in G to help avoid local minima. Each search iterates a fixed number of steps T to guarantee the search ends even if trapped in a local minimum. $N(Y, E, G)$ returns the first E neighbors of node Y in graph G. Here E is an integer smaller than k and is used to further improve performance by checking fewer edges than available. $\rho(X, Y)$ is a distance function that computes the distance from X to Y. If cosine similarity is used, the *argmin* should be swapped to an *argmax* to maximize the similarity instead. Finally note that here K is the number of sought after closest neighbors, while k is the parameter used to construct the k-NN graph.

Various improvements exist to make this algorithm even faster, notably edge pruning to mitigate nodes with a large amount of outgoing edges, which are slow to process. Another improvement is the use of hierarchical navigable small world graphs

Algorithm 1 graph-nearest-neighbor-search(G, q, K, E, R, T) [28]

Require: graph k-nearest neighbor graph G, query point q, K number of neighbors to find, E max edges per node, R number or restarts, T iterations per restart
Ensure: K nearest neighbors of q

```
S ← {}
U ← {}
for r=1 ... R do
    Y_0 ← random node in G
    for t=1 ... T do
        Y_t ← argmin_{Y∈N(Y_{t-1},E,G)} ρ(Y, q)
        S ← S ∪ N(Y_{t-1}, E, G)
        U ← U ∪ {ρ(Y, q) : Y ∈ N(Y_{t-1}, E, G)}
    end for
end for
sort S based on U
return first K nodes in S
```

(HNSW) [30], which adds layers of simpler graphs, allowing the algorithms to take larger leaps initially. Figure 2.7 shows an example of the search performed on a small graph with the HNSW graphs. The time to execute a search is improved for very large graphs at the cost of storing more edges from using multiple graphs.

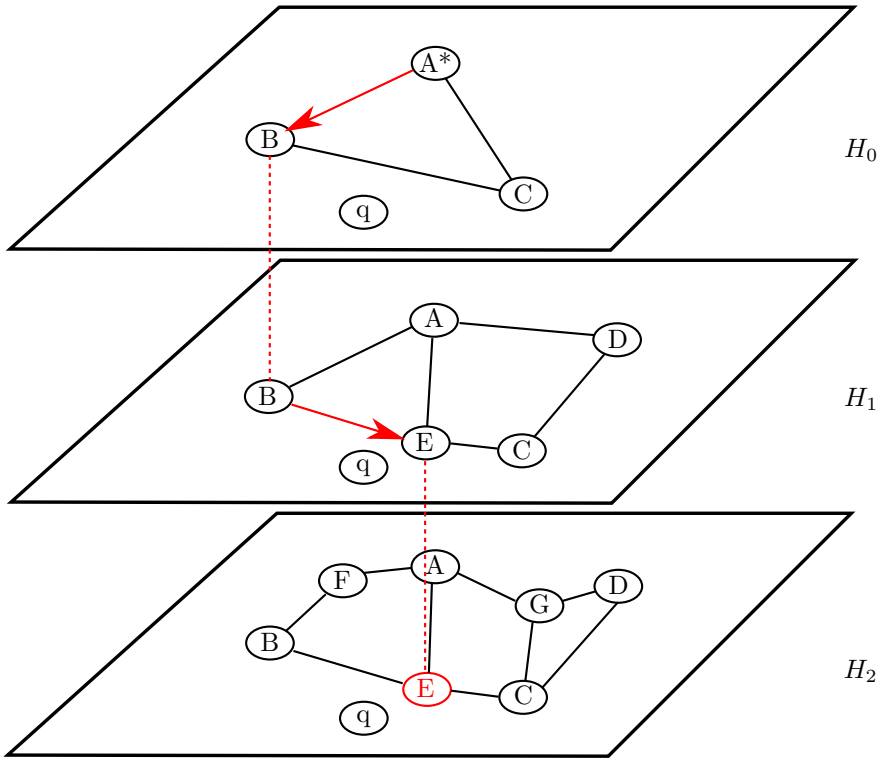


Figure 2.7: Example of a hierarchical structure used by HNSW. Point q is the query point and the start node is denoted A^*. Each level H_i contains significantly fewer nodes thus allowing each step to traverse a larger distance.

2.4.3 Vector quantization

Nearest neighbor search is often performed on billions of vectors. On this scale exhaustive search is not feasible, requiring other search indices that can take up a considerable amount of memory along with the memory required to store each vector for similarity comparisons. To get around this problem it is appropriate to compress each vector to a smaller size. This compression is called quantization, and has to also preserve the property that compressed vectors can still be compared for similarity.

Formally, quantization is the process of reducing the size of a set of continuous values by mapping them to a smaller set of discrete values [31]. The end goal is to preserve as much of the information as possible with this map. In this thesis we use product-quantization, which is commonly used in conjunction with approximate nearest-neighbor search to reduce memory usage [32].

Product quantization uses k-means clustering to find k optimal clusters and their centroids. The idea is that for any vector, it is more memory efficient to store which cluster that vector belongs, than to store the full vector. Given a set of vectors, first split each vector into even sized chunks, called subvectors. This yields multiple sets of subvectors, each subvector originating from a different vector. Next run k-means clustering separately on each set of subvectors. The cluster centroids make up a codebook that can be used to map any subvector to the closest centroid..

Quantization of a vector can now be done by splitting the vector into subvectors, finding which centroid each subvector belongs to, and storing the unique ID of that centroid as the compressed subvector. Further, the vector can be decompressed with the use of the codebook by mapping the centroid IDs back to the subvectors of the centroids, though with some error from the lossy compression.

An optimization introduced for product quantization (OPQ) is to include additional free parameters that rotate the full vector space. This rotation makes the vectors more amenable to PQ coding by relaxing the constraints of PQ, reducing the distortion from compressing the vectors [33].

3

Methods

This chapter will describe the details of how the algorithm was constructed. The main part consists of the patch selection method, where the saliency map of the image is used to select relevant patches and a hierarchical approach is employed to also capture smaller objects in the patches. The patch selection will then be followed by a description of the embedding of the patches, as well as the search among those embeddings.

3.1 Patch selection

The images used in this study are large and require large feature vectors to fully describe the content. The goal of patch selection is to instead enable search on a patch-level with a limited number of patches per image. A motivation for searching with patches is that feature extraction models such as CLIP resizes the input image to 224×224 pixels respectively. Thus small details are immediately lost if the whole image is used as a feature vector.

An alternative to using whole images is to select regions within the image. This makes the eventual embeddings created more specific, allowing search for smaller objects. Still, objects occur at different scales, so there may not exist a perfect patch size to use for general object search. It would then be appropriate to use multiple sizes to capture objects at different scales.

A naive approach to patch selection is to divide the image into a grid and use each cell as a patch. There are numerous issues with this method, such as potentially cutting an object in half if it falls on the boundary of a cell. Allowing grid cells to overlap by employing a strided square method solves this issue. The greatest problem with this solution is that the space required to store all selected patches quickly grows beyond available memory. This approach is implemented as a baseline and referred to as *baseline-unfiltered*.

We can handle these issues in two ways: by using larger patches with a larger stride, or by reducing the number of patches. The first suggestion will reduce the performance for small objects as discussed previously. Because of this, we opt for the second suggestion. The approach used in this thesis is to filter out uninteresting patches by scoring each patch based on saliency. The next few sections describe this approach in detail, and the full algorithm of patch selection is given in Algorithm 2 in Section A.2.

3.1.1 Saliency prediction and scoring

We create an initial set of patches for each image via a strided square approach. Here, stride refers to the distance between the center points of two adjacent squares. This resembles dividing an image into a grid, but using a stride smaller than the patch size allows overlapping patches. This is important to avoid the cases where an object might be cut in half from just using a simple grid.

The saliency map of the complete image (see Figure 2.2) is predicted using SimpleNet. We extract each candidate patch from this saliency map to get a set of candidate saliency patches. These patches are scored with two different scoring functions. The first method measures the *average pixel value* in the patch and is computed as

$$S_{\text{apv}}(p) = \frac{1}{255}\left(\frac{1}{A}\sum_{p\in P} p\right) \tag{3.1}$$

where 255 is the maximum pixel value, P is the set of pixels in the patch and A is the patch area.

The second score is based on the difference between the average pixel value of the inner and outer part of the patch. Since a perfect saliency map gives a light blob where an object is, a perfectly selected patch should have a large light blob in the center and darker corners. This score is called *inner difference score* and is computed according to

$$S_{\text{id}}(p) = \frac{1}{255}\left(\frac{1}{A_{in}}\sum_{p_{in}\in P_{in}} p_{in} - \frac{1}{A_{out}}\sum_{p_{out}\in P_{out}} p_{out}\right) \tag{3.2}$$

where P_{in} and P_{out} are the sets of pixels in the inner and outer part of the patch and A_{in} and A_{out} are the areas of the inner and outer parts. The inner part was represented by a smaller centered rectangle in the patch with a margin space of 10% of the patch side length in each direction. This score was then used to determine which patches were likely capture interesting objects in a good way.

3.1.2 Selecting the k most interesting patches

As an initial and simple filtering, all patches with an average pixel score using Eq. 3.1 under a specified threshold are excluded. This is done to decrease the risk of selecting completely empty patches which should have a low average pixel value score.

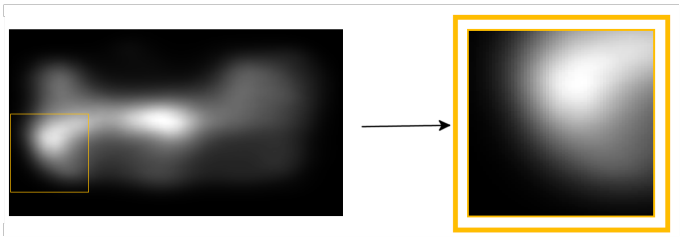
Next we compute the inner square difference score according to Eq. 3.2 and sort patches from high to low score. Although we allow overlap in initial patches, there is still a need for reducing the number of overlapping patches as these essentially capture the same content. To filter overlapping patches, we first add the highest scoring patch to the set of selected patches. Then the next best patch is added only if the largest fraction of shared area to every other selected patches was below a maximum overlap parameter. Since patches can have different sizes, the overlap is calculated as the overlapping area relative to the area of the larger patch. The patches were added consecutively in this manner until k patches were selected.

Adding this filtering step with saliency scoring to the baseline-unfiltered model reduces the number of patches considerably. We refer to this models as *baseline-filtered*. To capture objects of different sizes we can use multiple patch sizes, and still select the top scoring patches. The model that uses different patch sizes with saliency scoring is referred to as *multsize-filtered*.

3.1.3 Using hierarchy for different object scales

Since SimpleNet resizes the image to 256×256, smaller objects are likely to be missed and excluded from the saliency map. An example is shown in Figure 3.1, where the saliency map of the whole image and the saliency map of a cropped area are compared. To capture these objects, the saliency map needs to be predicted on zoomed in parts of the image which are likely to include objects. This can be done through an initial step of selecting larger patches in the image. Smaller patches can then be selected based on the saliency map of these initial patches. This gives rise to a hierarchical patch selection structure which we refer to as *hierarchical-filtered*. Although it requires many forward passes through SimpleNet, it does improve the quality of patches for small objects.

(a) saliency precision on cropped area



(b) saliency prediction on full image

Figure 3.1: An example of the difference between (a) predicting the saliency map on a cropped area and (b) predicting the saliency map on the full image.

The hierarchical structure could be represented by a tree. Each branch connects a sub region with its parent region and the root of the tree is the original image. To create the next level in the tree, patches are selected from each of the regions at the current level. The patch selection process is done as described in the previous sections. Figure 3.2 shows some patches selected with the hierarchical approach.

In each level of the hierarchy, we limit the total number of patches. By using several different patch sizes in one level of the hierarchy and allowing competition against each other in the patch selection process, the probability of selecting the best patch

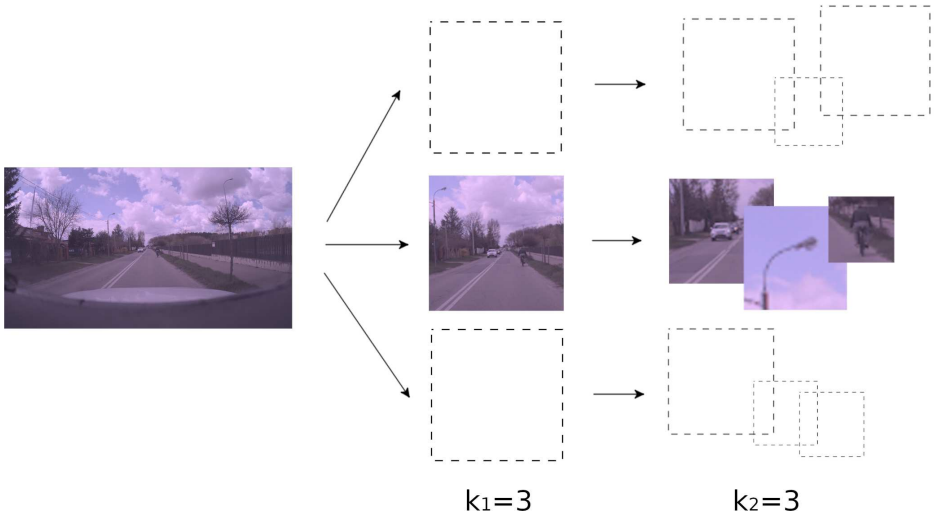
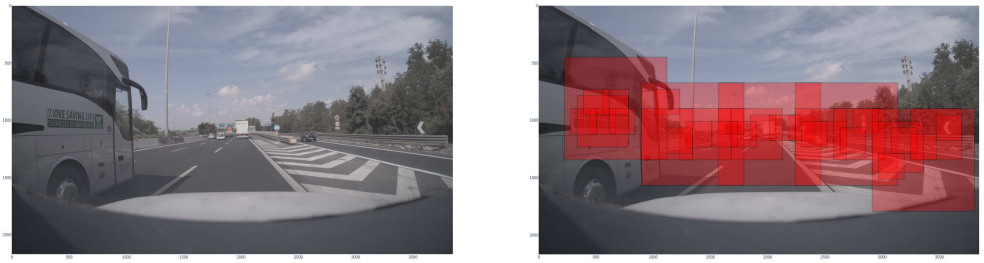


Figure 3.2: Illustration of the hierarchical patch selection. k_i describes the number of top-k scoring to select in each level. Note the different sizes of patches in the last layer of the tree.

size for a certain object increases and fewer objects are missed. This is because the patch selection otherwise tend to focus on only a few very clear objects while other less clear and perhaps smaller objects are missed.

Since the saliency map is trained on images which mostly include at least one clear salient object, it tends to give false positive areas for images without any salient objects like an image of only sky or road pavement. Considering this, as well as the method being approximate and imperfect, a single zoom in step using a few larger patches is sufficient. Especially considering that the saliency prediction is a costly procedure. The selected patches for an image are shown in figure 3.3.



(a) original image **(b)** hierarchically selected patches

Figure 3.3: An example of the patches selected with hierarchical patch selection in an image where $k_1 = 5$ and $k_2 = 6$.

3.2 Creating semantic embeddings

We now have a selection of patches from one or multiple different images. To enable search with both text and images, a model that could handle extracting features to a shared embedding space is required. CLIP suits this role and has been shown to produce a detailed embedding space and was therefore used for the feature extraction of both text and images.

We use the 16×16 vision transformer version of CLIP, named *ViT-B/16*. We begin by resizing every patch to 224×224, the resolution required by CLIP, followed by standardization of pixel values. Because of the use of square patches, this step does not warp the content of the patch but instead allows batch processing many patches, speeding up the embedding step. It should also be noted that the minimum patch size returned by the patch selection is 224×224 to avoid up-scaling a patch.

Since CLIP comes with a text encoder, it will be straight forward to also test the search on text queries. These queries are first tokenized using CLIPs' supplied tokenizer, then embedded using the text encoding model.

3.3 Nearest-neighbor search

The process of finding similar images is now formulated as the problem of finding the k nearest neighbor patch embeddings to a query patch embedding, then mapping the resulting patches back to their source images. *FAISS*[8] is a similarity search library developed by Facebook AI. It implements a variety of search algorithms and quantization methods, such as those introduced in Section 2,and was designed with GPU acceleration in mind.

3.3.1 Search process

Figure 3.4 shows the process of performing a patch-level search. We begin by supplying either a text or patch query which is embedded with the image/text encoder in CLIP to 512 dimensional feature vector that is then normalized using L2 normalization.

The algorithms used for finding the k nearest neighbors to the query depends on what kind of a search index is used. The indices used in this thesis all maximize the cosine similarity (Eq. 2.2) by maximizing the dot product between normalized vectors. The search retrieves the "patch ids" of the k approximate nearest-neighbors. The "patch ids" are used as keys in the patch database to retrieve the source "image ids" the patches came from and the patch bounding boxes.

One problem is that a search can return multiple patches from the same image. Because of the strided window patch selection method, these patches can even overlap the same object. If such patches do not overlap, then these are valid results that could describe the same type of traffic sign occurring multiple times in an image. If they do overlap, then it is likely that they overlap the same object, meaning it is unnecessary to return both patches. Thus we prioritize the highest cosine similarity

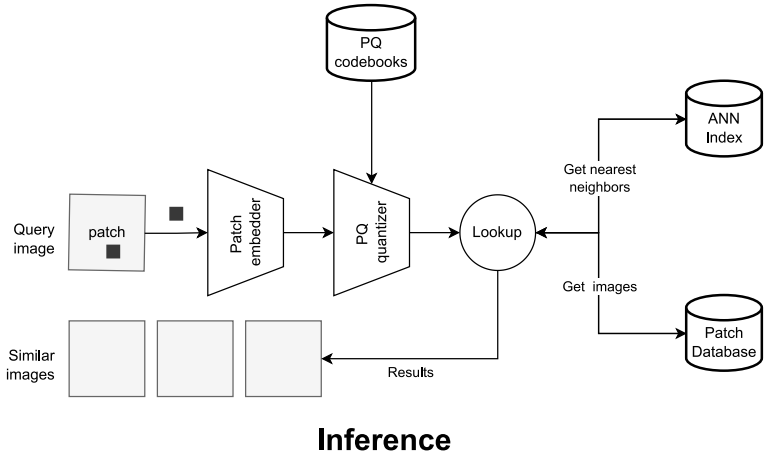


Figure 3.4: An overview of the search process. A query patch from an image is embedded into a 512 dimensional space and quantized with the PQ codebooks. The search then involves retrieving the top k patches from the ANN index based on the cosine similarity. The source images of these patches are retrieved from the patch database before presenting the results to the user.

patches found from the search, and filter away patches that overlap at all.

3.3.2 Search index construction

There are two parts to the search index: the actual index used for finding the nearest neighbor patches of a query patch, and the patch database that hold the mapping between patch and source image. Constructing these two parts was done in a end-to-end system that accepts images and the "image ids" as inputs. The patch database and search index is continuously added to as new images are processed. An overview of the construction process is shown in Figure 3.5.

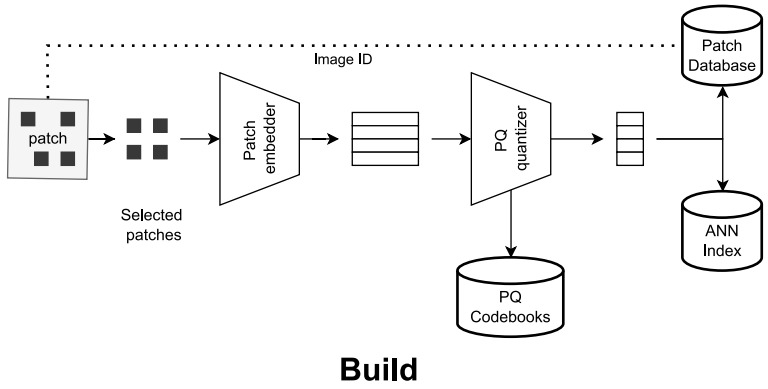


Figure 3.5: An overview of the process for building a search index and the patch database. Patches are selected from each image, then embedded and potentially quantized before being added to the search index. The source image id along with patch id is stored in the patch database for later retrieving the correct image given a patch.

To speed up this process, images were batched prior to patch selection to allow

processing more regions in parallel in the patch embedder. First patches are extracted as described in Section 3.1. Then each patch is embedded according to Section 3.2. Every embedding is normalized with by dividing by the L2 norm.

If quantization is used to further reduce the size of the index, the codebooks have to be available prior to adding any embeddings to the search index. These codebooks do not change during the course of the index, so the initial set of patches used for product quantization has to be large and varied enough to produce representative centroids. Given a dataset of 50 000 images we build the codebooks with 300 000 patches, which is a taken from roughly 8 200 images, or 35 patches per image. This number is determined by $8 \cdot \sqrt{N}$ where $N = 35 \cdot 50\,000$ is the total number of patches. Then every embedding added to the search index is first quantized using these codebooks.

4

Experiments

This chapter describes the evaluation and results of both the patch selection and image retriever components. To evaluate the image retrieval system, we make use of existing object annotations in the form of bounding boxes as well as some text queries. We differentiate between searching with one query, or the average embedding across multiple queries.

4.1 Data

To test the model, annotated data was made available to us by Zenseact. The annotations include the bounding box of the object, what class the object has, and other factors such as the occlusion of the object, i.e. to what extent the object is obscured by another object. The images come from a larger dataset of images shot in ordinary driving situations such as urban or suburban environments, and in different weather conditions. Each image has a resolution of 3848×2168 pixels and is at minimum separated by one second between different images shot with the same camera.

4.1.1 Evaluation dataset

A random subset of 50,000 annotated images was chosen for evaluating the patch selection and the image retrieval. After patch selection this dataset yields roughly 1.5M patches to search. It was important to test the system on rare annotated object types of varying size and prominence. The target classes of objects are therefore chosen to be ambulances, animals, and a certain type of traffic sign denoting a road narrowing. The reason for this specific set of classes is that each is relatively rare and important to detect for autonomous driving. Ambulances are large objects designed to stand out in an environment. Contrast this with animals, which are often small and easily blend into its surroundings. Simply searching for any traffic sign could yield high results, so we chose a somewhat rare sign as an example of a highly specific small object, designed to stand out.

The dataset consists of many objects, some of which are too small or occluded to be identified without zooming in. Since these objects often lie very far away, they are of less interest to the autonomous driving algorithms which focus on the direct surrounding of the vehicle. It would also be difficult for the patch selection algorithm to capture these objects in a good way, since they would appear small even in the

smaller patch sizes. Also, occluded objects will be hard to identify by the saliency prediction as well as the search algorithm. If only objects with an area of at least 3000 pixels and low occlusion are considered, the evaluation dataset includes a total of 290 843 objects. Among these, there are 96 ambulances, 106 animals and 99 road narrowing traffic signs where they have an average area of approximately 94 500 pixels, 23 300 pixels and 15 700 pixels respectively.

4.1.2 Queries

To test the image retrieval system, we construct two query datasets of image and text queries respectively, each including the three targeted classes.

For each class, 20 query patches were chosen separate from the evaluation dataset. The patch queries were hand picked to include a clear representation of the targeted object. A random subset of these queries are shown in Figure 4.1.

Figure 4.1: A random subset of the set of 20 query images used during evaluation for each class.

Since CLIP is trained to embed texts and images in the same semantic space, we can also use text queries to search for image patches. One potential benefit of this is that the text can focus on the object that we are interested in, whereas an image query also includes surrounding context. Table 4.1 shows the text queries used for each class.

4.2 Patch selection evaluation

We begin by evaluating the patch selection separately from the retrieval system. The performance of the patch selection is critical, since regions not included in patch selection are invisible to the retrieval system. We also note the compromise between the memory required to store many patches, and the search precision.

To measure the performance of the patch selection algorithm, the number of objects that are well represented in a least one patch is counted. This was done for each of the targeted classes using the annotations. For an object to be considered well represented in a patch, at least 50% of the object must be covered by the patch, and

Table 4.1: Table of some text queries used in evaluation.

Type	Text query
Ambulance	"An ambulance" "An emergency vehicle" "A medical vehicle" "A hospital wagon"
Animal	"An animal" "A dog" "A horse" "A cow"
Traffic sign	"A warning road narrowing traffic sign" "A road narrows sign" "A triangle sign with two lines" "Traffic sign lane narrows"

the object itself must occupy at least 5% of the patch. An example is illustrated in Figure 4.2.

To compare the performance of different types of patch selection models, the models described in Section 3.1 were all tested. These models can be sumarised as:

baseline-unfiltered Grid variant with no filtering done. This model simply selects all possible patches with a fixed patch size and stride (Section 3.1),

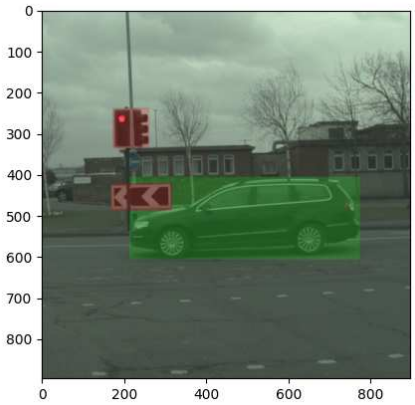
baseline-filtered Grid variant with score filtered patches. Same as baseline-unfiltered, but uses saliency to score every patch (Section 3.1.2),

multsize-filtered A variant where multiple patch sizes are used. All patches of all sizes are scored and compared to each other (Section 3.1.2),

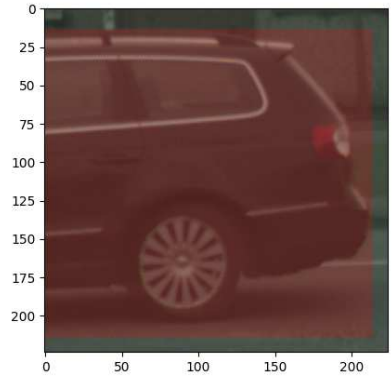
hierarchical-filtered Hierarchical variant with two levels, meaning the saliency map is recreated (Section 3.1.3).

A full parameter list for these models can be found in Section A.1.

The performance of the different patch selection models on the target classes is shows in Table 4.2. We note that *baseline-unfiltered* excludes many objects even with the large number of patches. Intuitively, this means *baseline-filtered* performs at best as well as *baseline-unfiltered* if no patches with target objects are filtered out. By introducing different sized patches in *multsize-filtered*, both larger object like ambulances and smaller objects like traffic signs are captured much more often. Finally we see a clear improvement with recreating the saliency map in *hierarchical-filtered*, notably for the animal object class. Interestingly, this is not true for the road narrowing signs, which are also small objects. Here the difference between these classes lies in how salient they are, and recreating the saliency map allows the model to detect less salient objects. Thus *hierarchical-filtered* is the best performing model while selecting few patches.



(a) A larger patch. Only the car is large enough to cover at least 5% of the patch.



(b) A smaller patch. The patch does not cover at least 50% of the car.

Figure 4.2: An example of how the evaluation handles objects in patches. Each full image is the patch, the green objects are considered to be captured by the patch, while the red are not.

Table 4.2: Patch selection evaluation for the different object classes. The metric is measured as the proportion of objects which are captured by a patch in a good way.

Model	**Ambulance**	**Animal**	**Traffic sign**
baseline-unfiltered (128 patches/image)	0.684	0.333	0.474
baseline-filtered (35 patches/image)	0.615	0.274	0.303
multsize-filtered (35 patches/image)	0.854	0.264	0.657
hierarchical-filtered (35 patches/image)	0.895	0.611	0.747

4.3 Image retrieval evaluation

We now show qualitative examples of what kind of patches the nearest-neighbor search returns. In these tests, we use the *hierarchical-filtered* patch selection since it demonstrated good performance in the previous Section. Figures 4.3, 4.4 and 4.5 show qualitative examples of the patches retrieved for a few different queries for each target class.

Figure 4.3: Patches retrieved for some ambulance query images with $k = 16$. (a): the search seems to focus on the text "ambulance" rather than the object it self in the returned patches numbered 5 and 15. (b): the search returns ambulances of different types and nationalities and has therefore generalised beyond matching text in the query with text in the patches. (c): an example where the search fails. The resolution of the query is low, which seems to affect the search result badly.

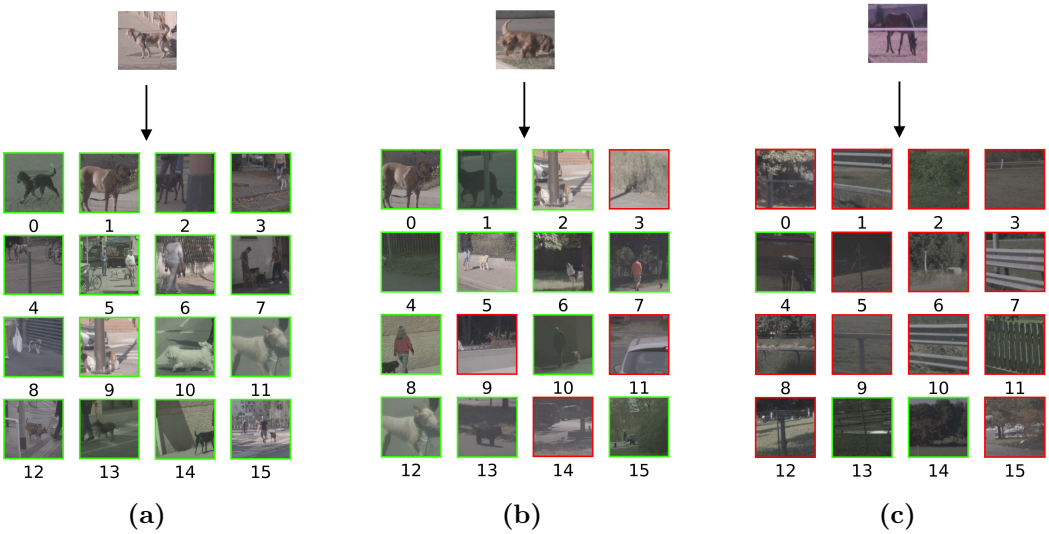


Figure 4.4: Patches retrieved for the animal query images with $k = 16$. (a): an example of where the model performs well by returning a variety of dogs despite the presence of many other objects in the patches. (b): despite there being no human present in the query image, the search returns patches where a human is the main object. (c): an example where the search fails by focusing on the fence rather than the horse.

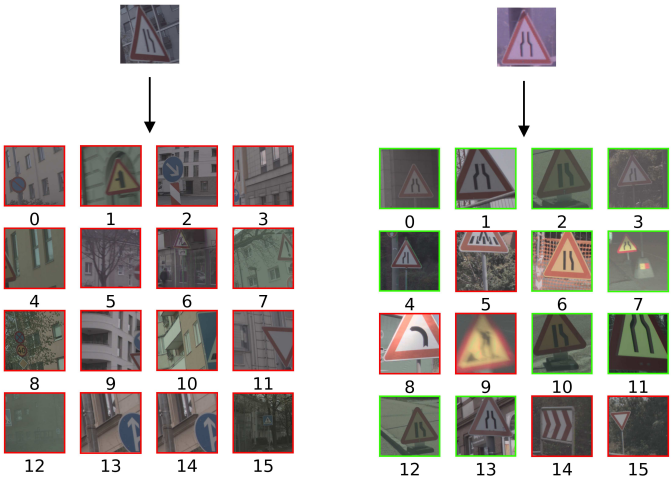


Figure 4.5: Patches retrieved for the road narrowing sign query images with $k = 16$. (a): an example of where the search fails by focusing on the wall rather than the sign. (b): an example of when the search correctly finds multiple signs of the correct type.

We now evaluate the image retrieval on two levels: a whole image level, and a patch level. This is done to measure the impact on searching among an patch embeddings instead of a image embeddings. When seaching among whole images, a square center crop of the image is used to avoid distorting the content when later downscaling for CLIP. The precision@16 is computed for the 20 different image queries and the text queries in Table4.1. We average across queries within the same target class and provide the mean and standard deviations in Table4.3. On a patch level we see similar performance for both image and text queries when searching for ambulances. For the other target classes, there is a significant difference in precision, notably for the traffic signs. This is likely because of the difficulty in describing very specific objects with text, especially those that would see many captions in the CLIP dataset.

Table 4.3: The mean and standard deviation (parenthesis) of precision@16 from searching with 20 image or 4 text queries, evaluated for two exhaustive search indices. Level "whole" uses a center crop of whole images to store a single embedding. Level "patches" uses the described patch selection to store more patches per image.

Level	Query type	Ambulance	Animal	Traffic Sign
whole	image	0.13(0.09)	0.04(0.06)	0.00(0.01)
	text	0.23(0.09)	0.09(0.03)	0.00(0.00)
patches	images	0.72(0.29)	0.56(0.39)	0.36(0.22)
	text	0.70(0.19)	0.30(0.27)	0.00(0.00)

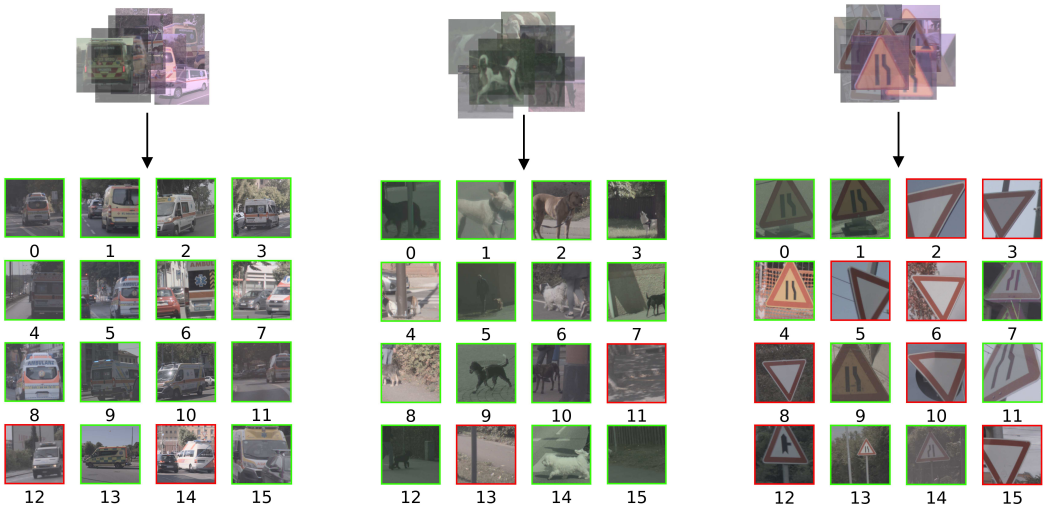
From Figure 4.5 (a) it is clear that it rather focuses on the background than the traffic sign. In an attempt to remedy this, we can try using the average query embedding

for a certain target class as query

$$\bar{\mathbf{q}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{z}_i^{(I)} \tag{4.1}$$

where $\mathbf{z}^{(I)}$ is the normalized query embedding resulting from the patch embedding model, and n is the number of queries. The hypothesis is that the average will preserve the part that describes the object class, while averaging out the differences such as background.

Figure 4.6 shows the search results from searching with the average embedding for each target class. It is clear that these averaged embeddings perform better than the mean performance of the individual queries shown in Table 4.3.



(a) Ambulance **(b)** Animal **(c)** Traffic sign query

Figure 4.6: Results from searching with the average embedding for each target class. (a): it performs well in returning different kinds of ambulances (precision: 0.875). (b): many dogs in different conditions, even though queries include other animals as well (precision: 0.875). (c): all signs have the same form, but the background and conditions varies (precision: 0.5).

We now show some results for searches not necessarily part of the target classes. Figure 4.7 shows a few different patch searches. Figure 4.7d demonstrates that even some scenarios such as a truck on the side of the road can be searched for. Figure 4.7b shows that matching precise text on traffic signs is difficult, and that the cosine similarity is high even for patches with very different meaning, such as the distance signs.

(a) bus **(b)** speed limit

(c) flowers **(d)** side of road truck

Figure 4.7: A few examples of the $k = 9$ nearest neighbor results for a four different patch queries. The top patch is in each figure is the query. The number below each patch is the cosine similarity to the query patch embedding.

A selection of the top 9 patches for four different text queries is shown in Figure 4.8. The cosine similarity for text queries is different to those from searching with patches, likely because the cross-modality of the search. Figure 4.8b is an example of a specific scenario query. Because of the inclusion of "red traffic light", smaller patches focusing

on this object dominate the search results. For this reason scenario search may be difficult to search for because of the number of small patches compared to large ones. Figure 4.8c is an example of the search returning a common street crossing sign. While the results technically depict a person crossing the street, the query related more to a photo of actual people crossing the street.

Figure 4.8: A few examples of the $k = 9$ nearest neighbor results for a four different text queries. The number below each patch is the cosine similarity to the query text embedding.

4.4 Quantization and approximate search

Next we measure the impact of approximate nearest-neighbor search and product quantization compared to a flat index that uses pair-wise comparisons of the full vectors to perform an exhaustive search. We are interested in measuring the effect of precision, index size in memory and time to perform searches. The three indices used are

exhaustive Exhaustive search index where pair-wise similarity is computed for every pair of vectors,

HNSW32 Hierarchical navigable small-world graph index where each node (vector) has 32 edges, and each vector is stored without any compression (Section 2.4.2),

PQ64 Exhaustive search index with product quantization. Vectors are first compressed to 128 dimensions with a linear transformation to make them easier to compress with PQ, then to 64 dimensions with product quantization. Each component of the vector is stored using 8-bit values. The compression takes 512 32-bit floats to 64 8-bit values, i.e. a compression factor of 32, which is close to the extra number of patches stored per image (Section 2.4.3).

Because building an index is time intensive, we only evaluate these three. Other parameter values and types of indices are available that varies the trade-off in accuracy, memory and time. Table 4.4 shows the difference in these metrics for each query class. The HNSW32 index clearly fulfills it purpose of being faster to search in than the exhaustive index, while remaining a similar performance. The PQ64 index aggressively compresses vectors at the expense of performance for all object classes, with small objects such as animals and traffic signs performing the worst.

Table 4.4: The mean and standard deviation (parenthesis) of the precision@16 metric across 20 image and 4 text queries, for different search indices. The flat index works as a baseline index utilizing an exhaustive search and not quantization. "HNSWXX" denotes an HNSW index with $k = \text{XX}$. "PQXX" indicates product quantization from 512 to XX dimensions.
*Same as the evaluation for images in Table 4.3

Model	Ambulance	Animal	Traffic Sign
exhaustive* (3.10GB, 0.75429 s/q)	0.72(0.29)	0.56(0.39)	0.36(0.22)
HNSW32 (3.78GB, 0.00036 s/q)	0.70(0.30)	0.50(0.43)	0.33(0.21)
PQ64 (0.11GB)	0.16(0.16)	0.00(0.00)	0.06(0.05)

To understand the poor performance observed for the PQ64 index, we show three patch searches for the target classes in Figure 4.9. From these figures, it appears that small details are lost in compressing the embeddings. For example some results for ambulances instead include vans, and the embeddings for the traffic sign is compressed into the general idea of a traffic sign, rather than describing the specific sign searched for.

(a) ambulance **(b)** animal **(c)** traffic sign

Figure 4.9: A few examples of the $k = 9$ nearest neighbor results for the target class queries using the product quantized search index. The top patch in each figure is the query used, and the number below each patch is the cosine similarity to the query patch embedding.

5

Discussion

This chapter discusses the results from the experiments detailed in Chapter 4. We then discuss some of the advantages and disadvantages of the method used. Finally we discuss some potential improvements and future work of the project.

5.1 Patch selection evaluation

The main goal of patch selection is to only select patches from areas which are visually interesting. The method used to accomplish this is a saliency map. We found that saliency detection is able to capture a majority of ground truth objects in the patch selection stage, as was shown by the patch selection evaluation in Section 4.2. The discussion in this section is based on Table 4.2.

There is a clear advantage to using a hierarchical method. The hierarchical-filtered model performs the best for all classes used. Even with the same number of patches with the same allowed patch sizes, hierarchical-filtered is strictly better than multsize-filtered. The difference between these models is that saliency is recomputed, and smaller patches are limited to already selected larger patches. Thus we attribute the improved performance to recomputing the saliency map.

The reason for why recomputing the saliency map has a large effect is that small details are lost in the initial image downscaling step of SimpleNet. This explains the poor performance for small details and therefore small objects. Interestingly, this is not a large issue for traffic signs compared to animals, as is clear by comparing multsize-filtered with hierarchical-filtered.

Across all evaluated models, it is clear that the most difficult class to capture is animals. Intuitively this is because animals are less salient than ambulances and traffic signs, which are designed to be visible. Even though the criterion is based on saliency, a majority of animals are found in selected patches. Because the number of patches per image is limited, salient objects have a priority over non-salient ones. This would explain the worse performance for the animal class.

The salient objects are ambulances and traffic signs. Traffic signs have a worse performance compared to ambulances in all models. This can be explained by traffic signs in general being smaller than ambulances and are thus more difficult to detect. In the baseline-filtered and multsize-filtered results, we see a large difference for the salient object classes. This difference is attributed to using many patch sizes, which makes the model flexible enough to capture objects of different sizes.

5.2 Similarity Search evaluation

From the search results in Table 4.3, it is clear that the model performs well for image queries. Further, there is a large increase in precision from searching among patches instead of whole images. This is because of the amount of information discarded by CLIP when downscaling large images. However the standard deviation is quite high, suggesting that the result largely depends on the quality of the query used.

One important aspect of this model is that CLIP is trained on matching images with captions. Thus, the model is able to consider two images similar if they can be *described* in a similar way, even though they might look visually different. This is exemplified in Figure 4.3b, where different looking ambulances are returned. In many ways this is a good thing, as it means objects can be embedded to the same embedding regardless of visual details such as object orientation or details in color. However, this is a problem if for example the goal is to search for a specific type of ambulance rather than any type of ambulance. If this is desired, another type of embedder trained on visual resemblance could be used.

For the object attributes to be represented in the embedding of the patch in the best possible way, the object should occupy a large portion of the patch as well as be the only object visible. For the classes including smaller objects, i.e. animals and traffic signs, this is less likely to happen which might be the reason for why the search performance is lower for these classes. As the object becomes smaller in a patch, the embedding will describe the surroundings of the object rather than the object it self. This problem was minimized through the usage of different sized patches in the patch selection algorithm, though there are likely still many such occurrences.

Even if the image queries were cropped in a good way to only include the relevant object, in some cases the search returns images without the query object, but which are similar in some other sense, e.g. matching the background of the object. Such examples are shown in Figure 4.4c and Figure 4.5a. This is a downside to searching with images rather than text. An attempt to create image query embeddings with minimal focus on context was done by searching once with the average of all query embeddings of a class. The results shown in Figure 4.6 indicate that this could be a possible solution to the problem.

The results of the searches in Table 4.3 show that there are benefits to searching with an image of an object rather than trying to search with a text description. To understand why, it is important to note the way in which CLIP is trained. Images are matched with their corresponding captions on the web, and the performance of the model therefore depend on how a human would describe an image containing the object. A reason for why the performance is low for the road narrowing traffic sign, while the corresponding performance in the patch selection is high, could be that a human simply describes it as a “traffic sign” rather than the specific type. This is exemplified in Figure 4.8d, where all signs have similar embeddings, but none depict the searched for sign. This specific type could also be unique to certain countries. Since CLIP is trained on English captions, traffic signs from non-English speaking countries could be harder to identify.

Because CLIP is trained with natural language, it gives large weight to text in images. This means that the search often improves for ambulances which include the text "Ambulance" in some form, as seen in Figure 4.3a. This is not necessarily bad, but it does mean that finding objects without text is more challenging. However, in Figure 4.7b it does not seem to matter what the text is, as long as the object has somewhat similar text (numbers on a sign). There is also the argument that text can be used trick CLIP into incorrectly embedding an image if it has a mismatched text [34], however the only examples of this in our dataset are billboards and logotypes.

5.3 Time and memory optimizations

In the results shown in Section 4.4, we found that it is possible to drastically speed up the search without significant loss of precision by using HNSW graphs. Although we expect the number of queries per second to be relatively small when using this system, this shows that if the search time is a problem for large datasets, then HNSW is a good alternative with a slight increase in memory usage.

For the quantized vector search indices we found a large reduction in performance across all target classes. We believe this is due to the aggressive compression factor of 32, which may not be the optimal choice. It is also possible that too few vectors were used to train the quantizer (8 200 images, 287 000 vectors). From the qualitative examples in Figure 4.9, it appears that the embeddings of specific objects such as the ambulance and the road narrowing traffic sign become less specific, i.e. vans and any traffic sign.

5.4 Bias and searching for private information

It is possible that the retrieval system described in this thesis could be used to find private information, especially because of the focus on rare and small regions. The kind of queries could involve searching for faces, people and text such as registration signs.

The authors of CLIP discuss bias they observed, particularly for ethnicity and gender [1]. Any such bias would also exist in the system described in this thesis. There is an argument to be made that enabling search for certain groups *could* be used to find training examples for other models, thereby improving performance for those groups. For example, if a model is performing badly for one particular gender, then searching for this gender will yield patches with the typical features associated with that gender. However non-conforming people will be less represented in this particular search, hence there is a bias in any such searches, introduced by CLIP.

Searching for specific registrations signs provides poor results similar to those from searching for specific traffic signs. If the goal is to find *any* registration sign regardless of text, then the search achieves a high precision. Also of note is that patches retrieved are often perfectly centered, zoomed in images of the registration sign. This is a side effect of using saliency which tends to focus on text, especially in small images.

With a larger dataset there may be a possibility of finding the same registration sign in different scenarios, thereby enabling tracking vehicles, but this is not something we observed in our dataset.

5.5 Future work

For future work, we stress the importance of reducing the number of patches per image. The patch selection algorithm described in this thesis evaluates many candidate patches that will never be selected anyway due to low score. We expect there exist more efficient methods than scoring every possible candidate patch, which would likely reduce the number of patches and the time required to select patches from images.

Another idea for reducing the number of patches involves filtering after creating embeddings. We expect this can be done by embedding more patches, but only adding those that are unique compared to the existing embeddings in the search index. We hope this process would mitigate the large amount of uninteresting patches such regions of sky or trees, since these are the types that are filtered out by patch selection process.

It may also be beneficial to search for quantization parameters that reduce the memory without significant precision loss. This was out of scope for this thesis as it is quite dependent on what one wants to search for, but more experimentation in this aspect would be a simple first thing to try. Product quantization may be more complicated than required. A simpler method to compress vectors is to transform the 32-bit floats to 8-bit integers with 8-bit integer quantization (compression factor of 4), as is popular in deep learning [35]. This may be desirable as the method does not require any training, in contrast to the $8,200$ images worth of patch embeddings used to train product quantization.

In terms of patch sizes, this thesis did not study how different sized patches change the result. For objects, our patch selection parameters work quite well as they match typical sizes of objects found in our dataset. Instead of setting a predetermined size of patches, an interesting idea is optimize this parameter based on objects that exist in the dataset. Here it is also interesting to study how providing more context in the form of a margin around the object, changes the search precision.

The results presented in Chapter 4 build on existing annotations for object bounding boxes. While creating a system that can find objects is interesting, objects are not the only thing one might want to search for. For example, it could be relevant to search for more complex scenarios such as a person crossing the road during a red light. However this would be more difficult to evaluate because it is not clear what the threshold for a correct result is. Using object annotations does provide some insight into how the model performs both for how many of the ground truth objects are included by patch selection, and how many are found during search.

In terms of patch sizes, this thesis did not study how different sized patches change the result. For objects, our patch selection parameters work quite well as they match typical sizes of objects found in our dataset. Instead of setting a predetermined size

of patches, an interesting idea is to optimize these parameters based on objects that exist in the dataset. Here it is also interesting to study how providing more context in the form of a margin around the object, changes the search precision.

Patch size and context also influences what is captured by a patch. This thesis places a large focus on finding objects, but objects are not the only thing one might want to search for. For example, it could be relevant to search for more complex scenarios such as a person crossing the road during a red light. How the system performs in this regard is difficult to evaluate because of the subjectivity of when a scenarios matches another scenario. This problem is also present in our object matching evaluation, since a patch of an object also includes some of the background and is therefore more specific than just describing the object label. Other methods to evaluate the search for this purpose could be interesting for future research.

6

Conclusion

In this work we studied the problem of how to search for different sized objects in high-resolution images in a large dataset. The method implemented uses three components: a patch selection algorithm for patches with interesting content; An embedding model that uses CLIP to extract feature vectors from patches and text descriptions; and a retrieval system that uses approximate nearest-neighbor search to find the vectors that are most similar to a query vector. Together, this system enables searching for small objects on a patch-level by comparing the semantic meaning of patches or text.

To evaluate this system, we use known object bounding boxes in the images. The metric reported is the precision of a search, which counts the number of objects that match the search query found in patches retrieved by the search. These metrics along with qualitative examples show that this system is effective at finding objects such as ambulances, small animals and certain traffic signs, even if these objects are rare. By comparing to a search index that does not use patches, we find searching among patches provides a large improvement in precision across all sizes of object. Therefore we conclude that searching on a patch-level is an effective strategy for finding rare objects in a large dataset of images.

The second problem under consideration concerned methods for scaling the system for large datasets. This problem arises because the number of vectors per image depends on how many is selected with the patch selection component. To study this part, we evaluated three different methods for searching and compressing the vectors. For time to execute searches, graph based searches were accurate although requiring more memory to store. Quantization had a large negative impact on the search result, likely due to the aggressive compression factor of 32. Because we only evaluated one set of parameters for each search index type, we can not conclude that any of these methods solve the second problem. Instead more parameter tuning is required to be sure.

Future work involves finding methods for improving the quality of patches selected and thereby reducing the number of patches required. It would also be interesting to study the effect the width and height of patches have both for capturing as much content as possible in patch selection, but also how different sizes affect search precision. Finally, instead of filtering patches, one can attempt filtering the feature vectors based on how many have already been added to the search index.

Bibliography

[1] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," 2021.

[2] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE access*, vol. 8, pp. 58443–58469, 2020.

[3] R. Novak, Y. Bahri, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein, "Sensitivity and generalization in neural networks: an empirical study," *arXiv preprint arXiv:1802.08760*, 2018.

[4] B. Settles, "Active learning literature survey," 2009.

[5] R. Datta, D. Joshi, J. Li, and J. Z. Wang, "Image retrieval: Ideas, influences, and trends of the new age," *ACM Computing Surveys (Csur)*, vol. 40, no. 2, pp. 1–60, 2008.

[6] P. Zezula, G. Amato, V. Dohnal, and M. Batko, *Similarity search: the metric space approach*, vol. 32. Springer Science & Business Media, 2006.

[7] W. Dong, C. Moses, and K. Li, "Efficient k-nearest neighbor graph construction for generic similarity measures," in *Proceedings of the 20th international conference on World wide web*, pp. 577–586, 2011.

[8] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2019.

[9] Y. Liu, D. Zhang, G. Lu, and W.-Y. Ma, "A survey of content-based image retrieval with high-level semantics," *Pattern recognition*, vol. 40, no. 1, pp. 262–282, 2007.

[10] F. Yan, C. Chen, P. Xiao, S. Qi, Z. Wang, and R. Xiao, "Review of visual saliency prediction: Development process from neurobiological basis to deep models," *Applied Sciences*, vol. 12, no. 1, 2022.

[11] M. Jiang, S. Huang, J. Duan, and Q. Zhao, "Salicon: Saliency in context," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[12] N. Reddy, S. Jain, P. Yarlagadda, and V. Gandhi, "Tidying deep saliency prediction architectures," 2020.

[13] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," *arXiv preprint arXiv: Arxiv-1405.0312*, 2014.

[14] M. Jiang, S. Huang, J. Duan, and Q. Zhao, "Salicon: Saliency in context," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1072–1080, 2015.

[15] S. Jain, "Tidying deep saliency prediction architectures." `https://github.com/samyak0210/saliency`, 2020.

[16] I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh, *Feature extraction: foundations and applications*, vol. 207. Springer, 2008.

[17] S. Ding, H. Zhu, W. Jia, and C. Su, "A survey on feature extraction for pattern recognition," *Artificial Intelligence Review*, vol. 37, no. 3, pp. 169–180, 2012.

[18] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[19] Y. Zhang, H. Jiang, Y. Miura, C. D. Manning, and C. P. Langlotz, "Contrastive learning of medical visual representations from paired images and text," *arXiv preprint arXiv: Arxiv-2010.00747*, 2020.

[20] N. Hezel, K. Schall, K. Jung, and K. U. Barthel, "Efficient search and browsing of large-scale video collections with vibro," in *International Conference on Multimedia Modeling*, pp. 487–492, Springer, 2022.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[23] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[24] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016.

[25] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[26] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[27] C. Fu, C. Wang, and D. Cai, "Fast approximate nearest neighbor search with navigating spreading-out graphs," *CoRR*, vol. abs/1707.00143, 2017.

[28] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang, "Fast approximate nearest-neighbor search with k-nearest neighbor graph," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[29] F. P. Preparata and M. I. Shamos, *Computational geometry: an introduction*. Springer Science & Business Media, 2012.

[30] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *CoRR*, vol. abs/1603.09320, 2016.

[31] Y. Matsui, Y. Uchida, H. Jégou, and S. Satoh, "A survey of product quantization," *ITE Transactions on Media Technology and Applications*, vol. 6, no. 1, pp. 2–10, 2018.

[32] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 1, pp. 117–128, 2010.

[33] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized product quantization for approximate nearest neighbor search," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[34] G. Goh, N. C. †, C. V. †, S. Carter, M. Petrov, L. Schubert, A. Radford, and C. Olah, "Multimodal neurons in artificial neural networks," *Distill*, 2021. https://distill.pub/2021/multimodal-neurons.

[35] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

A

Appendix 1

A.1 Patch selection settings

Table A.1: Parameters for baseline-unfiltered patch selection.

Parameters	**Values**
patch size	448px
stride	224px

Table A.2: Parameters for baseline-filtered patch selection

Parameters	**Values**
patch size	448px
stride	224px
mean pixel value threshols	0.05
k	35
overlap threshold	25%

Table A.3: Parameters for multsize-filtered patch selection

Parameters	**Values**
patch size	896px, 448px, 224px
stride	224px, 112px, 56px
mean pixel value threshold	0.05
k	35
overlap threshold	25%

Table A.4: Parameters for hierarchical-filtered patch selection

Parameters	**Values**	
	level 1	level 2
patch size	896px	448px, 224px
stride	224px	112px, 56px
mean pixel value threshold	0.05	0.3
k	5	6
overlap threshold	25%	25%

A.2 Patch selection algorithm

Algorithm 2 Selecting k unique patches$(I, S(), PC(), lt, ot\ )$

Require: graph I input image, $S()$ saliency prediction method, $PC()$ patch construction method, lt average pixel value threshold, ot max allowed overlap threshold
Ensure: top k unique patches
  $top\_k \leftarrow \{\}$
  $SI \leftarrow S(I)$
  $P \leftarrow PC(SI)$
  $P \leftarrow \{p : p \in P \text{ and } S_{light}(p) > lt\}$
  while $len(top\_k) < k$ **do**
    $p_{max} = p : p \in P \text{ and } S_{ids}(p) >= S_{ids}(p_{other}) \forall p_{other} \in P/top\_k$
    if $\{\text{overlap}(p_{max}, p_{selected}) < ot \forall p_{selected} \in top\_k\}$ **then**
      $top\_k \leftarrow top\_k \cup p_{max}$
    end if
  end while

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY