University of Münster

Department of Information Systems

Comparison of Remote Sensing Image Feature Extraction Methods for Nearest Neighbor Search

Master Thesis

submitted by

Lennart Seeger

Chair of Data Science:
Machine Learning and Data Engineering

Principal Supervisor	Professor Dr. Fabian Gieseke
Supervisor	Christian Lülf, M.Sc.
	Chair for Data Science:
	Machine Learning and
	Data Engineering
Matriculation Number	450010
Field of Study	Information Systems
Contact Details	lennart.seeger@uni-muenster.de
Submission Date	24.03.2023

Contents

1 Introduction

Today, finding trees in a desert or beautiful unknown beaches is only one fingertip away. Thanks to modern technologies, humanity has achieved to send satellites into space, providing image data of every square meter of the globe with an increasingly high resolution. This offers incredible new advantages for analyzing, for example, deforestation or finding specific things and places that are hard or expensive to reach.

However, processing this vast amount of data is a challenging task. Thanks to the advances of machine learning in recent years, this task can be increasingly well-accomplished by training classifiers. These can be helpful in various situations but have the downside that a new classifier has to be trained with labeled training data for every object that is supposed to be discovered. This raises the question of whether a more generalized approach can be created.

An early generalized Object Detection approach that only works for cities can be found on the GIS Lounge website[1]. Another more recent example for entire countries is described in Keisler et al. [Kei+19]. In the latter, the authors created a website[2] where a user can select a cutout on a map and gets similar cutouts in return.

The two examples are in line with the core idea of this work, but in recent years, research in the field of neural networks, computer vision, and automatic processing of remote sensing data has significantly improved. Thus, using a model pre-trained on ResNet50 as seen in Keisler et al. [Kei+19] might no longer be the best option available [Wan+22]. Due to the rapid advances in the area, findings that are several years old are likely to be outdated. This makes it impossible to rely on older sources and underlines the necessity to examine which modern methods are most suitable for generalized image representation to conduct a nearest neighbor search.

The present work provides a thorough overview of remote sensing image processing methods, especially the recent approaches applicable to create generalized image representations. Tests of the approaches are conducted with an unlabeled dataset of the country of Denmark. It is found that several methods like Transfer Learning, NNCLR, MAE, and SimCLR can be used for the task at hand. Of those methods, SimCLR performed best, and the reasons for this are analyzed in chapter 5. Finally, some particularities and fine-tuning opportunities of the methods are shown under the limitation of using only one GPU. All insights are combined in one final trained model and a conclusion is drawn giving an outlook for the future.

[1] https://www.gislounge.com/terrapattern-search-engine-satellite-imagery/
[2] https://search.descarteslabs.com

2 Problem Description

2.1 Real World Applications

In real-world application scenarios, finding objects is the primary interest when working with remote sensing data. It can not only help to create scientific value, but it can also have economic advantages as shown by a project of the french government.[3] In France, the property tax rate is dependent on certain luxury factors such as the possession of a swimming pool. Therefore, french citizens are obligated by law to officially register their pools. Nevertheless, many people did not do this, which led to several million dollars in tax evasion. With machine learning methods, those unregistered pools were found, and the owners had to pay tax arrears.

In another example, machine learning methods were used to find ships.[4] This helped to detect the position of superyachts of Russian oligarchs.[5] They have a very distinct appearance and can thereby be found in remote sensing data. With the information about the yachts' locations, governments can easily seek them out and seize them if sanctions are imposed against their owners.

The online retailer Amazon saw the need to find arbitrary specific objects and created the tool *Rekognition*[6], which is a flexible framework for Object Detection. The user can provide multiple samples of an object and then gets similar ones returned. Thereby, a custom model is trained for the specific application of the user with the provided labeled training data.

The examples show the diverse needs for the detection of objects. As seen in the Amazon example, a new classifier needs to be built for every object. A general neighbor search for objects could solve the above mentioned searches without the need for additional data and training for every use case, but just one sample of the searched instance.

2.2 Problem Description and Research Questions

The idea derived from the real-world applications is to create an approach that finds arbitrary objects similar to a given sample. With this, various tasks could be accomplished.

[3] https://www.theverge.com/2022/8/30/23328442/france-ai-swimming-pool-tax-aerial-photos
[4] https://medium.com/earthcube-stories/how-hard-it-is-for-an-ai-to-detect-ships-on-satellite-images-7265e34aadf0
[5] https://www.washingtonpost.com/technology/2022/03/10/russian-oligarch-yacht-tracking/
[6] https://aws.amazon.com/de/blogs/industries/remote-sensing-and-object-detection/

To get similar objects, the semantic similarity, meaning the similarity perceived by a human, is supposed to be recreated. Therefore, a method is needed that can provide a similarity score between two images based on their semantic similarity and not just on their, for example, colors.

It is difficult to get a similarity score for objects not yet known. A possible approach is to split the dataset into small cutouts and then try to find similar cutouts. This approach consists of two parts. (1) The part of extracting features from images that can be compared and have a high semantic meaning. (2) The task of comparing two representations to get a similarity score for them.

Derived from the first part of this previously explained goal of finding similar neighbors, the first research question for this work is:

- Which promising methods for feature extraction of (satellite) images can be found in recent literature?

The idea is to identify methods that can indeed extract the semantic content of images and represent them in some numerical way to be later used for comparisons.

The potential methods should be compared in an exact scenario. A dataset without any indexed objects or labels can be used for this. The research question for it reads:

- Which methods identified in the first research question work best for nearest neighbor detection of satellite data in a practical application setting?

To make the research feasible, this work focuses on the most promising methods for neighbor search using the given dataset of Denmark. The dataset consists of values for the color bands RGB. Only these are regarded during research and experiments. Additionally, to identify relative improvements, the focus is laid on recent advantages in the domain that were mainly published after the reference paper by Keisler et al. [Kei+19] in 2019.

These aspects lead to the formulation of the following two assumptions and exemptions:

- The analysis is reduced to semantics based on visual aspects. Data other than the RGB colorbands is not regarded.
- The analysis focuses on technologies published in the last years mainly after the publication of Keisler et al. [Kei+19] to identify the recent progress and improvement in the field.

To identify neighbors with the numerical representations conveying the semantics of the images, a method for comparison is needed. Since the methods for feature extraction can extract features for the complete dataset in advance of a neighbor search, the method that compares the vectors is crucial for the final runtime of a search. Therefore, it should ideally be working in real-time, so in less than a second to not let the user wait for the results. This can get incredibly challenging with remote sensing data if larger areas or countries, like Denmark in the example, are analyzed. The Denmark dataset alone consists of millions of cutouts which would all need to be compared to the query image to find the most similar cutouts in the intended short amount of time.

The aforementioned speed is not a primary research goal. Nevertheless, it is important if the finally detected best-performing method is provided to users in real time.

2.3 Research Approach

Before answering the research questions, a thorough foundation is laid by explaining definitions of neighbors, critical aspects of computer vision as well as the different learning types in machine learning. This is done to provide the reader with a detailed overview of the specific task at hand and possible solutions and also with valuable insights necessary to understand interconnections. The goal of the work is to assess the approaches in the remote sensing domain regarding their generalizability.

The foundations are followed by the presentation of literature findings in the domain of computer vision and image representation. The aim is to identify both methods that have already been applied to remote sensing data for image representation and those that have only been used with normal image data. Both approaches are considered because it is expected that newer methods perform better, but not all of the methods applied to normal images have been already tested with remote sensing data yet.

To explore the most promising methods, the focus is laid on those with good results on evaluation datasets. Additionally, the credibility regarding the number of citations, the journal they were published in and the referring authors and organizations are used for selection. Other papers not matching those requirements will be mentioned for further research, but those methods will not be regarded for the second research question and will not be explained equally thorough.

In the next step, methods are chosen that have libraries and implementations of them available and differ in terms of their used techniques and technologies. Choosing different approaches makes it possible to derive research outcomes that are as general

as possible. The methods are explained in more detail before they are applied in the second and practical part of the work, where the second research question is addressed. It should be kept in mind that many of the methods have probably not been applied to remote sensing data before. Due to the differences between normal image data and remote sensing image data, the successful transfer can not be taken for granted. For this, differences and expected problems are addressed.

To be able to replicate the experiments, all of the relevant steps of the implementation are explained in detail in section 4.2, and the code will be provided in the IFEfNNS GitHub repository[7].

[7] https://github.com/lennartseeger/IFEfNNS

3 Theoretical Research Background

This chapter describes the research on feature extraction from images and the theoretics of nearest neighbor search. After an overview of computer vision, things necessary to regard in the remote sensing domain are elaborated. This is followed by the presentation of concepts and possible methods for feature extraction. The chapter concludes with possibilities for neighbor comparisons that regard the performance constraint described in the problem description.

3.1 Foundations of Computer Vision

The task of understanding images by computers is called computer vision. In the 2000 released book Computer Vision by Shapiro and Stockman [SS00], the following definition can be found: "The goal of computer vision is to make useful decisions about real physical objects and scenes based on sensed images" [SS00, p.13].

Another more specific definition is from Datta et al. [Dat+08], calling the problem content-based image retrieval describing it as: "any technology that in principle helps organize digital picture archives by their visual content" [Dat+08, p.2].

The two definitions contain the aspects of organizing and decision-making based on images. This is an interesting distinction regarding the methodological approaches over the decades. The oldest approach is to use basic statistical descriptors. One of the most common ones is color-based, for example, the use of histograms [Dat+08]. Those approaches help with the organizing part of the definitions but not with the actual decision-making. Images could be sorted regarding some fixed parameters, but the color is not alone responsible for objects inside an image.

The color-based approaches were used mainly until the early 2000s. Then machine learning methods became dominant [Dat+08]. Those methods, and especially the in recent years emerged neural networks, provided a giant performance leap in the domain [Vou+18]. It also improved the fulfilment of the first part of the definitions to "make useful decisions" [SS00, p.13].

Why this distinction is so important and why more advanced methods like neural networks became necessary can best be realized with the example seen in figure 1. Both images mainly display water and thereby consist in most of the image of the same color. Nevertheless, the interesting part in the left image is the windmill. It is easy for a human to distinguish this image from 1000 others of water. To do this as a computer, the more sophisticated machine learning methods are necessary [Vou+18].

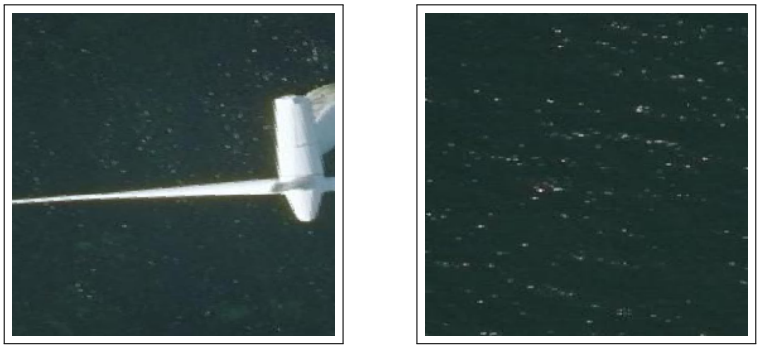


Figure 1 Images with High Similarity in Color and Low in Semantics

3.2 Overview of Existing Methods in Literature

Remote sensing images and their applications are not equal to normal ones. Therefore, the different application scenarios in the remote sensing domain and used images are explained first. This knowledge, in combination with the afterwards presented machine learning types, can then lead to an overview of useful concepts for the processing of images regarding the specific requirements. Then, the most promising methods are identified and explained more sophisticatedly. Finally, datasets usable for the quality assessment of methods are presented.

3.2.1 Application Scenario Overview

Several practical application scenarios were identified with the increasing amount of available high-resolution satellite imagery. The following subsection provides an overview of those and explains which insights can be gained with modern technologies from remote sensing imagery. The four most prominent application scenarios focused on by recent literature are LULC Classification, Object Detection, Scene Recognition, and Change Detection [Ma+19].

LULC Classification

LULC stands for Land Use and Land Cover. LULC Classification aims to assign one class for every pixel in a selected area. Those labels can be, for example, evergreen, water, surface or residential. Knowing the labels and the type of the land is very helpful for planning purposes and assessments of, for example, intact rain forests in a particular area [Ngu+20].

An example of such a classification is visualized in figure 2 where the region of Dak Nong in Vietnam was classified with a support vector machine [Ngu+20]. If such a task is approached, it is either tried to classify every pixel on itself taking the

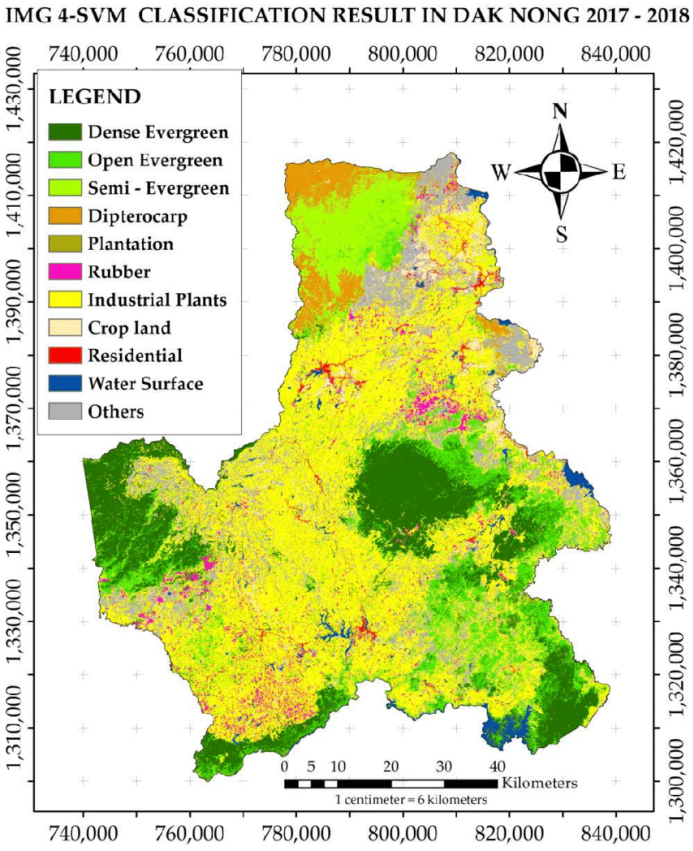


Figure 2 Exemplary LULC Classification [Ngu+20]

surroundings into account or to find objects and structures and classify them together to get the labels for every pixel in the images [AG21].

Object Detection

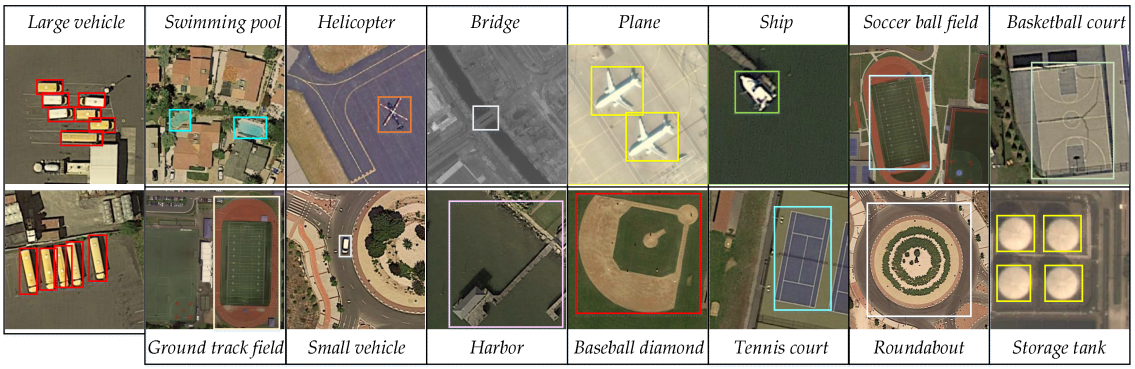


Figure 3 Exemplary Object Detection [Xia+17]

Object Detection is the second block of application scenarios in the remote sensing domain. It aims to find specific objects in the data relevant to a user. Thereby, one approach is to classify whether an object is in a particular area, not regarding how many times it is in it. Another approach is to find the objects and identify bounding boxes around them. This can be seen in the example in figure 3 [Xia+17].

Additionally, the approaches can differ regarding the generalizability of objects. On the one side, some approaches search for very specific objects like a particular yacht [CH16]. On the other side, approaches can also try to find more generic objects like in the paper of Brandt et al. [Bra+20] where the authors detected any type of tree in the West African Sahara and Sahel.

Scene Recognition

Figure 4 Exemplary Scene Recognition [Qi+20]

Scene Recognition also called Scene Classification, is another application scenario building basically on top of the previous two approaches, Object Classification and LULC Classification, by extending it with the identification of a more general label for an area. A harbor could be classified by LULC Classification as water and artificial objects. By the Object Classification, the boats are detected, and then the overall Scene Classification could label the entire thing as harbor [Che+20c].

The exemplary image in figure 4 is from the MLRSNET dataset, which contains such labeled images that have an overall topic like dense residential area and then also the labels of detected objects within the image like buildings or cars [Qi+20].

Change Detection

The fourth application for remote sensing data is Change Detection, which identifies alterations in images. Therefore, an area is monitored over time to get multi-temporal

images. With those different images, it can be checked whether there are changes in the images and how the monitored area changed over time [YCZ20].

This is especially interesting in urban areas, water bodies or the development of forests. It allows researchers from other domains to analyze even very remote areas and monitor huge areas simultaneously [YCZ20].

The two images in figure 5 show an extreme example. They compare the city of Cancun in 1985 and 2019 and thereby depict how Cancun became a major resort in those couple of decades [Pat20].

(a) Cancun in March 1985 (b) Cancun in April 2019

Figure 5 Comparison of Cancun Before and After the Transformation to a Major Resort[8]

3.2.2 Learning Types

The idea of machine learning is to create a model or an algorithm that enables the computer to learn something. This learning happens by processing data through the model and thereby enhancing the model in a way that enables it to create meaningful results. The underlying data used for training can thereby be either labeled or unlabeled. Meaning whether the desired outcome is known for the data because it was provided by a human or whether it is not known. This differentiation is called learning taxonomy. It can be extended by additional approaches that can partially overcome the situation of insufficient labels being available. The five main concepts for learning are distinguished subsequently [Ayo10].

Supervised Learning

Supervised learning is the most intuitive learning taxonomy. In supervised learning, the data for training and related labels, that a model should predict, are available. A

[8] https://earthobservatory.nasa.gov/images/146194/how-cancun-grew-into-a-major-resort

common task in this area is classification, and an example dataset is MNIST[9]. This dataset consists of images of handwritten numbers and the referring numbers that were supposed to be written. The computer can train a model with the given data and has the referring labels to assess the quality of the model in every step [Ayo10].

Unsupervised Learning

Unsupervised learning is the opposite of supervised learning. In this case, no labels at all are available. This means the computer has to learn something without getting the expected result. Therefore, it is hard to assess whether something is right or wrong. To conduct unsupervised learning, a user needs to define a function that is supposed to be optimized. This can be, for example, a clustering task where the computer should create groups of datapoints with low distances within clusters and high ones in between clusters [Ayo10].

Semi-Supervised Learning

In the real world, completely labeled datasets are rare because it can be costly to label all the datapoints [Che21]. Nevertheless, usually, a fraction of data can be labeled. For this case, techniques exist that can facilitate the remaining unlabeled datapoints, and it is not necessary to only use the few labeled datapoints [EH20].

The way to do this is semi-supervised learning. In this two-step approach, one option is to first label the remaining unlabeled datapoints with, for example, a classifier trained on the labeled datapoints, and then to conduct the task in a normal supervised way, making this approach very similar to the normal supervised approach with the intermediate step of predicting pseudo labels for the unlabeled datapoints [Che21].

Self-Supervised Learning

Another variant of learning, if no labels are available, is to create such a task so that the unlabeled datapoints can become the labels themselves. This approach is called self-supervised learning [Che21].

One example would be to augment the images in a dataset and to train a classifier to find similar images. In this task, the original images and their augmentations would be a pair, and the other images would be seen as different. Thereby, a network would be trained that creates representations that are closer for similar images and more distant for not similar images [Che21].

This idea can also be used in a semi-supervised setting if just a few labeled and many unlabeled datapoints are available. In this case, first, a model can be trained

[9] http://yann.lecun.com/exdb/mnist/

to differentiate classes, and then with the final labels, those classes can get their referring class names [Liu+21b].

Reinforcement Learning

Another way of learning is reinforcement learning, where similar to unsupervised learning, an agent exists that indicates how well the method performs. With this approach, the computer is not told how to do things but what was good in the past. Therefore, a network can learn over multiple iterations a behavior how to do something without any description of how the task should be done. Crucial for this is creating an agent that can judge past decisions without having particular labels [Ayo10].

Transfer Learning

Regarding this division of methods and the earlier introduced fact that a lot of data without labels exists in the case of the Denmark dataset, only unsupervised and self-supervised learning seems feasible. However, research has shown that not only does direct learning yield good results, but it is also possible to use models that were trained with one of the above methods on a completely different dataset with a different data distribution than the one used in the end and still perform appropriately. This approach is called Transfer Learning [PNS21; Tan+18].

To do this, one option is to take a neural network model for classification and remove the last layer doing the classification. After that, a new so-called head can be added for the new task. Additionally, if just a representation is needed, for example, for similarity comparisons, adding a new head and conducting additional training is unnecessary. Instead, the intermediate vectors can be used directly for such tasks [Gér19].

Essential to mention at this point is that with the usage of different datasets also from other application areas, it is easier to get appropriate labeled training data. Therefore, with this flexibility, all learning types introduced earlier can be used within a transfer learning task.

3.2.3 Used Concepts

The literature on content-based image retrieval, especially in remote sensing environments is presented in this subsection. In the beginning, an overview is provided with different review papers. After that, concepts that are used for the tasks of content-based image retrieval are presented. Presenting concepts was chosen over

the presentation of whole methods because most approaches combine useful concepts that are integrated to yield the best overall design.

After the general overview, the most interesting concepts are pointed out, and exemplary methods fulfilling additional constraints for implementation are introduced in more detail.

Overview of Review Papers

Looking at the review papers, it becomes clear that Deep Learning algorithms have gained more and more popularity in recent years. In the 2019 released paper by Ma et al. [Ma+19], Convolutional Neural Networks, Recurrent Neural Networks, Autoencoders (AEs), Stacked Autoencoders, Restricted Boltzmann Machines, Deep Belief Networks and Generative Adversarial Networks were mentioned as the main methods in their meta-analysis on the remote sensing domain [Ma+19].

The papers also describe various comparably older statistical arithmetic methods to preprocess remote sensing images and extract features. Those are usually listed at the beginning of the survey papers, but recent activities in the domain are then dominated by approaches using Deep Learning [LMZ21].

In Petrovska et al. [Pet+20], those older statistical approaches are mentioned in a brief introduction, but they are discarded afterwards to focus directly on neural network approaches for the evaluation. After that, a state-of-the-art workflow is recommended to compare different architectures [Pet+20]. That shows that only Deep Learning methods seem relevant.

In the field of Deep Learning, the researchers detected that the different classification tasks are the most researched ones in the remote sensing literature among the ones already introduced in section 3.2 [Ma+19; YCZ20]. Another aspect mentioned in the papers is the limitation of training data. It is both a limit to the training and evaluation phases. Therefore, getting more labeled data and training without labeled data are two current research directions [LMZ21].

The Deep Learning approaches mainly use three of the presented learning taxonomies. The first group consists of unsupervised methods if there is only data without labels available or no information about the later task. Examples are Abdollahi et al. [Abd+20], Tsagkatakis et al. [Tsa+19] and Sun et al. [Sun+21b] showing significant relevance in the area.

The second group are the transfer learning methods. One example of this was carried out in Li et al. [Li+20]. In the beginning, the authors provide an overview of the topic, followed by training a network to identify 20 different objects. Using a network

for feature extraction in a transfer learning approach might be one possibility for suitable neighbor identification. This group of methods is widely mentioned in the literature. For example in Diakogiannis et al. [Dia+20], Zhang et al. [Zha+20] and Li et al. [Li+20].

The last group consists of the straightforward approach of training a new network specifically for neighbor identification with labeled data or data with pseudo labels. The dominant possibility for this is self-supervised learning, and especially contrastive learning [Che21]. The approaches in this group fit the research question best, but fewer papers exist on this topic. Also, the papers are newer and usually only tackle normal image data, not remote sensing data.

Basic Statistical Methods

Despite gaining more and more popularity, content-based image retrieval is not new. Several years ago, when Deep Learning was not invented yet, it was conducted with statistical methods [LMZ21].

Thereby, these earlier approaches of content-based image retrieval were based on features crafted by experts. Domain experts thought about what could be extracted from images to have a good representation. The result are features, for example, dependent on color, edges, or texture [Wan+14; JV96; MM96]. Those approaches belong to the class of unsupervised methods because the extraction methods are created without training on explicit images.

One exemplary approach was to use histograms in the category of color-based features [LMZ21]. Histograms show which colors in the RGB color scheme are how frequent in the image. To compare images, all those histograms can be created for the image query and the images in the database. Then the Euclidean Distance is used as an option to get similarity values [JV96].

Feature Extraction of Pretrained Networks

Already introduced as one learning type, another approach which is the one used in the reference paper by [Kei+19], is feature extraction from a neural network trained with another dataset. In this setting, the approach is to use the network without finetuning and just extract the features before the head.

One type of network architecture usually used for this is the so-called residual network architecture short ResNet [He+16]. These networks are especially good when it comes to transfer learning tasks, in general [Tan+18], and also in remote sensing applications [AK22]. The specific property regarding the authors is that they: "explicitly reformulate the layers as learning residual functions with reference to the

layer inputs, instead of learning unreferenced functions" [He+16, p.1]. This structure with residual networks is easier to train, and the success in different competitions like the ImageNet classification[10] proved how effective the networks are [He+16].

Self-Supervised and Contrastive Learning

Supervised Learning has become state-of-the-art in many tasks in computer vision. Nevertheless, the necessity of (human) supervision for usually large needed datasets is a big downside in many such approaches. In cases without enough labeled datapoints, alternative so-called pretext tasks can be used to get some good representations of the images and further train on those representations. With the pretrained representations, models can converge faster, meaning less training time is needed for the final task. Additionally, such an approach can prevent overfitting [JT19].

One of the earliest approaches in this category was to colorize greyscale images. The idea is to train a network to find colors for a greyscaled image. In the process, a good representation can be learned for further finetuning of models in a downstream task [JT19].

Despite some success with early approaches, many methods had limitations or very high computational costs. This could change with a breakthrough in self-supervised learning by Chen et al. [Che+20a] with their introduced method of contrastive learning. The underlining idea is to train a network with positive pairs. Meaning to train a classifier that learns representations of images that are similar for similar images and then automatically dissimilar for dissimilar images [Che+20a]. One of the most intuitive approaches is to think of image classes. A cat is similar to another cat, and a dog is similar to another dog. However, a cat is dissimilar to a dog.[11] This idea was intended to be reflected in the model and algorithm [Che+20a].

The general approach is not new and was already facilitated in a network architecture called Siamese Networks earlier. This architecture consists of two networks with tied parameters that are trained to provide similar outputs for similar inputs in their so-called energy function that is applied to the outputs of the two networks. They are trained with a labeled dataset using the same classes as positives and different classes as negatives [Koc15].

The difference to the new approach is that it needed a labeled dataset, which is usually not available in a real-world scenario. This is where Chen et al. [Che+20a] had the breakthrough idea by not using elements belonging to the same class, but using data augmentations to gain pairs. Once pairs are acquired, the images are fed

[10] https://www.image-net.org/challenges/LSVRC/
[11] https://towardsdatascience.com/understanding-contrastive-learning-d5b19fd96607

into a neural network. Then the resulting features without the dense layers on top are fed to some function calculating the similarity, and the network is adjusted to minimize it [Che+20a].

After the publication of the paper by Chen et al. [Che+20a], many other researchers have tried to enhance the system by, for example, changing the gaining of positive pairs with other augmentations or custom approaches. Some most interesting and best-performing methods amongst those are SwAV [Car+20], InfoMin Aug. [Tia+20] BYOL [Gri+20] and NNCLR [Dwi+21].

Although contrastive learning is mainly used as a pretext task, such approaches can also be used to directly use the vectors for other tasks like in [Koc15] where they are used for the task of one-shot image recognition, meaning the finding of similar images based on only one sample. Also, a Keras implementation[12] exists that facilitates the networks for nearest neighbor search directly.

Contrastive Loss Functions

The networks try to represent similar patches similarly. To assess the quality of the intermediate results, most of the papers, like the one by Chen et al. [Che+20a], use the so-called NT-Xent (normalized temperature-scaled cross entropy) loss [Che+20a].

The function uses the elements in a minibatch to calculate the loss. In Chen et al. [Che+20a] for every original image, two augmentations exist. One is the positive sample (z_i), and the other is the targeted pair (z_j). All other augmentations in the minibatch are seen as negative samples (z_k). So having 16 images, all 16 are augmented twice, resulting in $2N = 32$ images in the minibatch. The overall function then is [Che+20a, p.2]:

$$l_{i,j} = -log \frac{exp(sim(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} exp(sim(z_i, z_k)/\tau)} \tag{3.1}$$

The referring calculation for sim is the dot product of two l_2 normalized vectors (Cosine Similarity) and defined as [Che+20a, p.2]:

$$sim(u, v) = \frac{u^T v}{||u|| \, ||v||} \tag{3.2}$$

This function calculates for an augmented image the similarity with the other augmented image of the same original one. This result is divided by the sum of the

[12] https://keras.io/examples/vision/siamese_network/

similarities of the augmented image with all other images in the minibatch except with the second augmentation of the same original image. For better performances, this calculation is balanced using a temperature parameter (τ), the exponential function and a logarithm to norm the calculation [Che+20a]. Regarding the authors, this loss function with norming performs significantly superior. It regards the hardness of the negative samples due to respecting the numerical similarity between the query image and the negative samples [Che+20a].

Since the divisor of the function in the fraction is dependent on N, which is directly dependent on the batch size, the loss can have different values for different batch sizes, resulting in the fact that, especially for smaller batch sizes, it would be negative with a perfect model. In such a case, the similarity between the first augmentation and the second augmentation would be one, and the similarity between the first augmentation and the other augmented images in the minibatch would be zero. Then the theoretical loss would be below zero, and this could lead to problems during the training. It can be overcome using a good value in the normalization parameter τ.

Despite the superiority of this type of loss function, many authors state problems with it under special configurations. One such problem is the negative-positive coupling which occurs with small batch sizes. It makes it challenging for the network to learn appropriately because the self-supervised learning task can get too easy if the negative cutouts are unsimilar to the original one. This happens with small batch sizes because, for every pair of an image, only very limited amounts of negative samples exist. The paper by Yeh et al. [Yeh+21] presents an approach to decouple the success from good hyperparameters and large batch sizes [Yeh+21].

Nevertheless, the decoupling could not sufficiently solve the problem, and recently published papers like the one by Chen et al. [Che+22] still describe the necessity of large batch sizes in contrastive learning. Thereby, this question still seems to be comparably unanswered [Che+22; Yeh+21] and the only option regarding Chen et al. [Che+20a] to cope with it is to train for many more epochs if small batch sizes are used.

Contrastive Accuracy

Next to the contrastive loss, another metric used to rate the training advances is the contrastive accuracy. It is used by Chen et al. [Che+20a] in table 5. It is defined for every batch as the number of cases of the current batch, where the nearest neighbor of an element is the element augmented from the same original image [Che+20a].

This calculation also depends on the batch size because if just small batches are used, there are just a few wrong samples instead of many potentially neighboring samples with large batch sizes.

Optimizer

As an optimizer, the self-supervised learning methods use different stochastic gradient descent approaches based on Robbins and Monro [RM51]. The explicit algorithms used are variations published in recent years like the Adam algorithm [KB14] or the enhanced version called AdamW, which has, as the authors state, increased generalization ability [LH17].

In the very recently published papers on contrastive learning, the authors were able to use very large batch sizes. They state that this can lead to a degradation in the training accuracy [Che+20a]. Therefore, a usually used optimizer in such papers is the Lars optimizer, which can handle larger batch sizes appropriately without a decrease in accuracy [YGG17].

Autoencoder

Another method from the field of machine learning are Autoencoders. In this approach, the idea is to let a neural network conduct a recreation task. For this purpose, the approach consists of two parts, namely, an Encoder and a Decoder. First, in the Encoder, the data is inserted into an input layer with a defined number of neurons. After that, it is processed through several hidden layers. At least one hidden layer has fewer neurons than the input layer. The representation in the middle is called latent representation and is processed afterwards through the Decoder to recreate the input as well as possible. The recreated image is then fed to the output layer, which has the same amount of neurons as the input layer. To evaluate the quality of the reconstruction, a loss like the mean squared error can be facilitated [Liu+20].

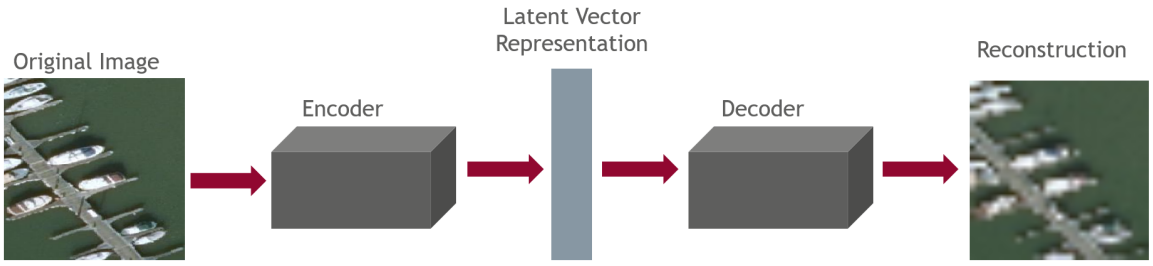


Figure 6 Architecture of Convolutional Autoencoder[13]

[13] https://sefiks.com/2018/03/23/convolutional-autoencoder-clustering-images-with-neural-networks/

Figure 6 is a graphical illustration of an Autoencoder for images. In the case of images, transposed convolutional layers can be facilitated in the Decoder to be able to work with convolutional layers in the encoding phase. Those are crucial for a good performance in the processing of images [DV18].

After training the Autoencoder, two approaches can be used to use the result further. First, for every image, the recreation error can be calculated. The idea is that the Autoencoder can easier reconstruct images that are more frequent in the dataset because it needs to focus on those frequent datapoints in the dataset to minimize the reconstruction loss. Rare datapoints are much worse recreated because the model could not focus on those during training. With this approach, anomalies can be found because the easy to recreate datapoints would be expected to be normal, and the hard ones to be recreated are the anomalies since those are frequent or unfrequent in the dataset [AC15].

The second and more relevant option for this work to further use the values is to use the latent representations of the Autoencoder directly for further computations. Those representations should tend to be similar for similar elements since it would make the reconstruction task easier if similar elements are mapped to similar representations. Therefore, the nearest neighbor search, as needed for the given case of finding similar patches or a clustering, can be approached with those latent representations [BKG20].

Applications of this can be found in Masci et al. [Mas+11], where multiple Convolutional Autoencoders are stacked on top of each other to train the generation of unsupervised features. Nevertheless, applying Autoencoders to images did not yield competitive results until the publishing of the paper Masked Autoencoders Are Scalable Vision Learners, which uses the idea of Autoencoders in combination with masking and Vision Transformers to get useful feature representations [He+21].

Vision Transformers and Attention Model

Nowadays, a powerful attempt in a variety of different applications are so-called Vision Transformers. They have a state-of-the-art performance in several tasks like classification on ImageNet[14] [Kha+22].

Vision Transformers consist of three main parts. The first part is built of Attention Layers, where an Attention Function is computed. The second part is the Transformer which combines several Attention Layers with additional layers and interconnections. The third part is the actual Vision Transformer. This part transforms the image data

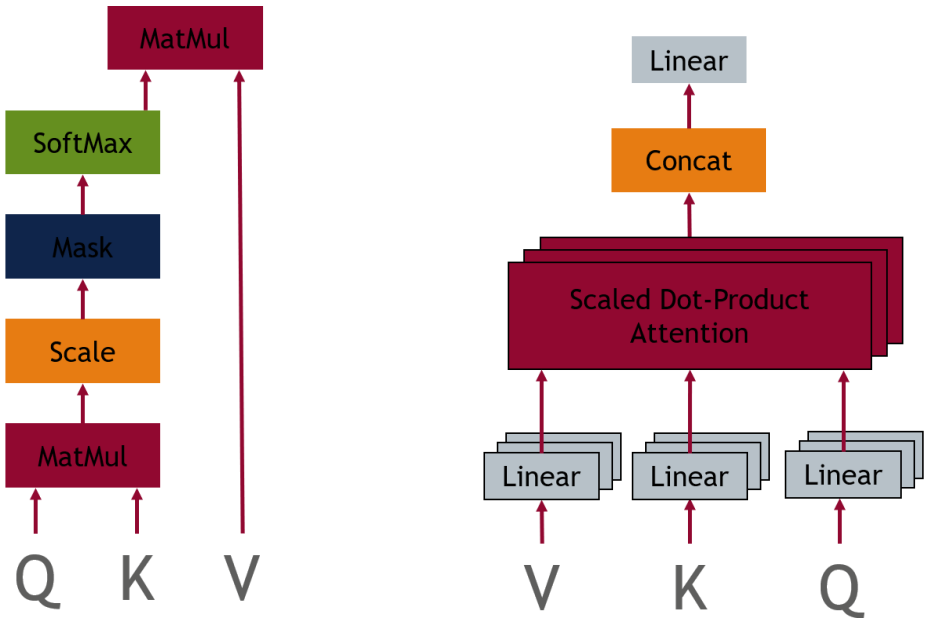
[14] https://paperswithcode.com/sota/image-classification-on-imagenet?p=deepvit-towards-deeper-vision-transformer

into a representation that can be processed by a Transformer and conducts the specific task like classification. The three main components are explained subsequently [Vas+17].

Attention

To conduct an Attention computation, the input, which is usually text or, in the given case, images, must be transformed into a vector representation. This can be done for text with word2vec [Mik+13] or for images with the later introduced architecture of the Vision Transformer. Additionally to the input transformation, a positional embedding is added to the representations to regard the order of the input [Vas+17].

After that, the Scaled Dot-Product Attention as seen in figure 7a can be calculated. Therefore, three vectors, namely queries, keys and values, are needed. Those key, query and value can be seen as representations of the input vector and are calculated by multiplying the initial vectors with a set of weights for each of the three values.[15] After that, the actual computation is conducted. First, the dot product of the query with all keys is calculated, and every result is then divided by the square root of the dimensionality of the keys and queries. After that, Softmax is applied to the result and the result is multiplied by the values [Vas+17].



(a) Scaled Dot-Product Attention [Vas+17] (b) Multi-Head-Attention [Vas+17]

The type of Attention used in Transformers is called Multi-Head Attention. The general idea is to use linear projections of the three vectors (key, query, value) multiple times. After that, the Attention Scores can be calculated for each *Head* and

[15] https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

concatenated afterwards. After another projection, the final vector output is calculated. An overview of Multi-Head Attention can be seen in figure 7b. By reducing the dimensionality of each Attention instantiation and parallel computing the results, the computational cost is not increasing with the number of heads linearly [Vas+17].

In several papers, it is assumed that Multi-Head Attention is superior because it has "the ability of jointly attending multiple positions" [Liu+21a, p.1]. Nevertheless, this assumption is refuted by Liu et al. [Liu+21a], showing that the main advantage is the training stability of the models when using Multi-Head Attention [Liu+21a].

Transformer

For the Transformer, the recently emerged approach of an Encoder-Decoder structure for sequences is used. First, the input is encoded in the Encoder. Afterwards, it is processed by the Decoder, and thereby, the model uses elements from earlier in the processed sequence to compute the latter ones. The Encoder and Decoder each combine multiple Attention Layers for the computations [Vas+17]. The in this work relevant Vision Transformer approach only uses the Encoder of the Transformer. One example of it is depicted in figure 8 [Dos+20].

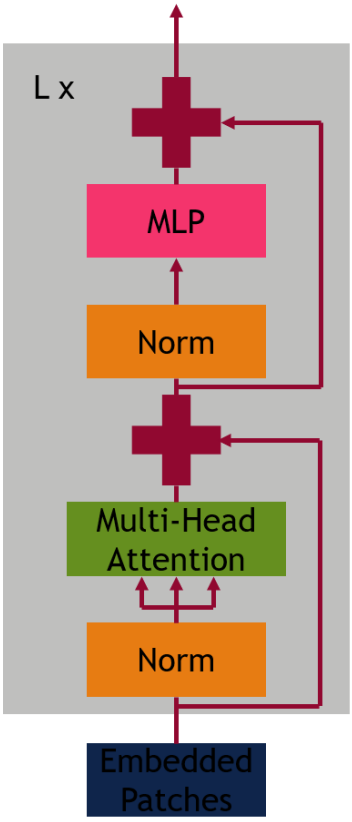


Figure 8 Transformer Encoder of Vision Transformer [Dos+20]

This Encoder consists of Attention Layers that are combined with a feed-forward layer and a Norm step after each layer [Dos+20]. The used Attention is Self-Attention allowing it to access all keys, values and queries from the output of the previous layer [Vas+17]. This construction is stacked an arbitrary number of times. Additionally, the authors use a residual connection around the Feed-Forward and Attention Layers to the Norm step to allow the later network parts to use earlier inputs for learning. For the stacking to work, all sub-layers and embedding layers produce outputs of the same dimension [Vas+17].

Humans can interpret the results of the Self-Attention Layers in Transformers. For text, Attention would assign a score to words and the scores are high for words that are important for another word for example between a noun and an adjective [Vas+17]. For images, Attention highlights the essential objects in an image conducting an image segmentation between important and less important pixels [Car+21].

Vision Transformer

Initially, the Transformer architecture with Attention Blocks was created for text. Vision Transformers presented by Dosovitskiy et al. [Dos+20] adapt the ideas to image processing. By doing so, they achieved state-of-the-art accuracies in competitions like ImageNet Classification[16] competing with the Convolutional Neural Networks which dominated the area for years.

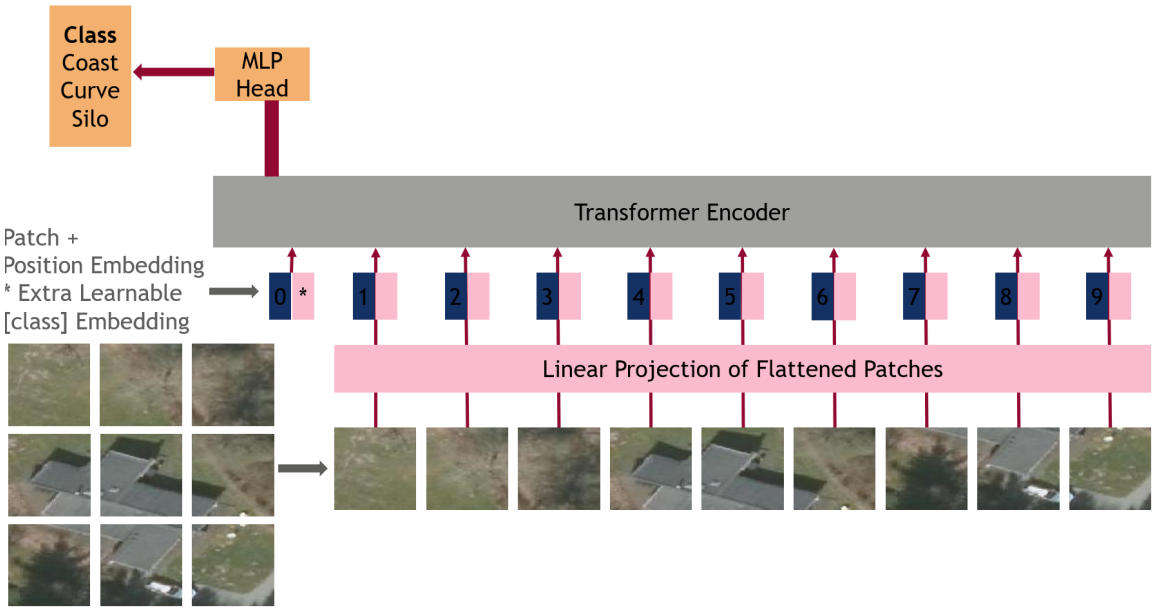


Figure 9 Vision Transformer Architecture [Dos+20]

[16] https://paperswithcode.com/sota/image-classification-on-imagenet?p=deepvit-towards-deeper-vision-transformer

The overall architecture of a Vision Transformer is depicted in figure 9. It follows the idea of the Encoder of a normal Transformer with a classification layer. Same as text, images are not consisting of 1-D numeric arrays. Therefore, the first step is splitting the images into equally sized patches. Those patches are then flattened and mapped to a fixed input length D with a trainable linear projection. After that, a positional embedding is added for every patch to keep the positional information of the image. Those transformed patches are afterwards fed into the Encoder, and the results are fed to an MLP Head to conduct the classification [Dos+20].

With this approach of inserting all cutouts with a positional embedding and letting all attend to each other, the model can learn long-range dependencies easier in early layers than Convolutional Neural Networks [Rag+21].

Vision Transformer Applications

Originally, Vision Transformers were used in a supervised setting for classification. However, approaches for a self-supervised approach have come up recently. This enables it to use them for pretraining tasks without the need for labeled training data. Before the upcoming of the self-supervised approaches, the pretraining was mainly conducted on not even always publicly available supervised datasets [AAK21].

The most famous methods for this are most likely DINO [Car+21], Moco v3 [CLW21], and MAEs [He+21]. The first two facilitate Vision Transformers instead of Convolutional Neural Networks for self-supervised tasks and build frameworks on their findings. The third describes an approach that was only enabled by the architecture of Vision Transformers since the described masking, which the approach uses for an Autoencoder recreation task, is not easily possible with Convolutional Neural Networks [He+21].

Nevertheless, the use of the Attention Model did not start after the release of the Vision Transformer. The Attention Model was already facilitated in other neural network architectures with convolutional layers like in [Noh+17]. There the Attention Mechanism was used to focus on image parts that are especially relevant to the semantics of the image. This results visually in a keypoint selection. If two images are similar, the same keypoints are selected, and those tend to have similar vectors compared with each other [Noh+17].

3.2.4 Additional Interesting Approaches

The previous concepts are the major ones and most advanced in conducting image feature extraction. Nevertheless, several other approaches could also be found. Those

also cover aspects not directly connected to a model's training but to supportive functionality. Some of those approaches are briefly shown subsequently.

Preprocessing Approaches

One group of approaches, like the one by Radenovi [Rad19], focuses on possible preprocessing steps that are applied before any model is created. In this step, for example, wrongly labeled training data can be handled if this is necessary for a specific dataset. This step can be crucial to learn good image representations in a later task [Rad19].

GAN

Another approach, called General Adversarial Networks, normally used to generate samples similar to input samples, can also be facilitated in the search for good features of images. During the generative step of those networks, a binary representation is created by the model, which can be used as features [Son+18]. This approach has a high performance for Object Detection of small objects in the area of satellite images, according to Rabbi et al. [Rab+20].

Image Clustering

A self-supervised attempt with an artificial dataset labelling is the so-called DeepCluster algorithm. It uses a Convolutional Neural Network and a basic clustering algorithm like k-means in combination. First, the objects are clustered, and second, the neural network is trained using the clusters as labels for the training. This is done several times iteratively [Car+18]. Another approach also using clustering is the one presented by Gansbeke et al. [Gan+20], who combined self-supervised learning in the first step and clustering in the second step and thereby avoided relying on low-level features. Code for this is accessible at Keras[17].

Hashing

The step after the feature extraction is also regarded in several papers by tackling the problematics of the search in the huge amount of data. Some examples are Fernandez-beltran et al. [Fer+20], Liu et al. [Liu+21a] and Araujo et al. [Ara+18], which facilitate hashing to increase the comparison speed of the image vectors.

U-NET and Pyramid Networks

In recent years, network architectures of different domains were also used in remote sensing. Some of those are the pyramid network architecture by Lin et al.

[17] https://keras.io/examples/vision/semantic_image_clustering/

[Lin+16] or the similar U-NET architecture by Ronneberger, Fischer, and Brox [RFB15] [Dia+20]. For example, those networks could help with differently sized images.[18]

Object Detection Improvements

In addition to the general approaches that can be applied to normal image data and remote sensing images, some authors focus especially on the specific requirements regarding remote sensing data. In Sun et al. [Sun+21a], the authors are addressing the problem of composite objects in images because it can not be assumed, as in most classification datasets, that every image contains only objects strictly belonging to one class. Another research in this direction was conducted by Fu et al. [Fu+20] addressing the rotation invariance and different sizes in remote sensing data because, for example, a small aircraft is similar to a larger one, and it does not matter in which direction it is headed. On remote sensing images, though, those aircraft can look quite different.

Text Based Similarity

Some researchers in the domain also try to use the recent good NLP advances like in ChatGPT[19] for image classification. First, text is extracted from images like in the paper by Chen et al. [Che+19] and then the textual output is compared and further processed.

3.2.5 Remote Sensing Datasets

For the evaluation of remote sensing data, several labeled datasets exist. They address a variety of applications. Nevertheless, the technologies introduced in subsection 3.2.3 are mainly related to granular Object Detection. A person is generally interested in finding similar objects and is not interested in only getting patches that also have, for example, artificial surface as class. Therefore, LULC datasets are excluded. Also excluded are Change Detection datasets since the Denmark dataset is no multitemporal dataset, and Change Detection is therefore not feasible. The datasets relevant in the context of the possible tasks are listed with their parameters in table 1.

The presented datasets differ mainly in the number of samples and the resolution as seen in the table 1. Additionally, they differ in scaling. While datasets like BigEarthNet have one pixel every 10 meters, MLRSNET is much more detailed, with a resolution of up to a pixel per 0.1 meters. There also, much smaller objects can be

[18] https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c

[19] https://openai.com/blog/chatgpt/

Dataset	Classes	Size	Resolution	Type
UCMERCED	21	2100	256	One-Class classification
AID	30	30000	600	One-Class Classification
BigEarthNet	43	590326	120	Multi-Class Classification
Eurosat	10	27000	256	One-Class Classification
DIOR	20	23463	800	Object Detection
MLRSNET	46	109161	256	Multi-Class Classification
Dota	15	2806	800-4000	Object Detection

Table 1 Overview of Remote Sensing Datasets

identified. This makes the datasets usable in different situations depending on the size of the searched objects.

Another differentiation in the datasets is whether the resolution is fixed, like in BigEarthNet, or not, like in AID. There, the cutouts all have the same length and width but not necessarily the same amount of pixels per meter [Xia+16]. This can have the advantage that small and larger, for example, aircraft, can be compared. Overall, the decision which dataset to use in a scenario needs to be made carefully regarding the specific application and intended size of objects that should be detected.

Usage in Literature

All the beforementioned datasets are also used in the remote sensing literature for different applications and are used in several review papers for comparison. Some of them are Dimitrovski et al. [Dim+22], Li, Ma, and Zhang [LMZ21] and Petrovska et al. [Pet+20]. It stands out that despite its small size, the UCMERCED Dataset is still distributed the widest. AID, BigEarthNet and Eurosat were mentioned in two of the papers and the others were only mentioned by one author. However, this does not necessarily have any meaning for the quality because some of the presented datasets are comparably new and could not be used by many authors so far.

Topic Specific Datasets

Next to the compared general datasets useful for the evaluation, interesting to mention is that there also exists a variety of very specific datasets for particular applications. One example is the WHU-RS dataset[20] collection, which consists of multiple specific datasets, like one consisting only of cities. Those datasets are primarily intended for one research scenario but are also mentioned in some of the review papers like in Dimitrovski et al. [Dim+22], Li, Ma, and Zhang [LMZ21] and Petrovska et al. [Pet+20].

[20] http://gpcv.whu.edu.cn/data/building_dataset.html

Datasets that only cover a very small region are also regarded in literature. An example is the Rosis-03 indian pines dataset that consists of images from 1992 over Northwest Indiana. It was mentioned in the papers by Ma et al. [Ma+19] and Yuan, Shi, and Gu [YSG21]. Other datasets are purely constructed to display only one type of object. One example is the HRSC2016[21] dataset that displays a collection of high-resolution ships. Despite having some significant relevance in the literature, those specific datasets will not be helpful for the evaluation of the task formulated in the research questions.

Datasets From Foreign Domains

Not all methods that are useful for image feature extractions were already applied to remote sensing data. Most of them use the ImageNet[22] dataset to compare the performance with other methods. The dataset is included in this presentation to compare methods from the literature that were not adapted to remote sensing scenarios yet.

The ImageNet dataset was first published in 2009 by Deng et al. [Den+09] and contains over one million images divided into 1000 classes making it a vast and diverse dataset with objects of different domains like animals, vehicles, electronic devices or buildings. The current best performance for classification is a 91 % Top-1 accuracy[23].

3.3 Method Selection

In the previous sections, a variety of promising methods in the area of image feature extraction was presented. For the second part of this work, some of the most promising methods are chosen and implemented to find a method that can provide actual good results under the given general conditions.

Whilst the basic statistical approaches were the only ones existing for a long time, several authors suggest that they are not competitive. It does not seem likely that those can perform well in the given application. Therefore, the focus is laid on the machine learning approaches that were published after 2019 to align with the exemption explained in the approach. In the paper by Keisler et al. [Kei+19], feature extraction from models trained with domain foreign datasets is used. It shows the usefulness of this approach that is similar in Du et al. [Du+19]. Since this work tries to enhance the results of this paper, the approach with a ResNet50 is chosen as a baseline to assess the quality of the other approaches. Additionally, it can be tried

[21] https://www.kaggle.com/datasets/guofeng/hrsc2016
[22] https://www.image-net.org/
[23] https://paperswithcode.com/sota/image-classification-on-imagenet

to improve the results by exchanging the dataset the networks are trained on or the network architecture.

The next block of interesting methods began with SimCLR as the breakthrough method in contrastive learning. Since this is the most popular method in the area and the foundation for various others, it is chosen for implementation. To also see the progress in the performance of the improvements to the method, NNCLR was chosen as the best current contrastive learning method in literature based on Convolutional Neural Networks [Dwi+21]. Additionally, the approach used in NNCLR is especially interesting for the task at hand since it already uses nearest neighbors as positive samples during the training.

After the introduction of Vision Transformers, those were also used for self-supervised learning tasks. Of the three most popular methods, the MAE is the newest and, according to the paper, the best-performing approach. Therefore, this is an approach to see whether the current state-of-the-art performance on the ImageNet dataset can also be transferred to the Denmark data with fewer computational resources.

The other approaches described in the previous chapter are also promising but less prominent in recent research or do not have a benchmark on a highly used dataset, making them hard to evaluate. Therefore, the five chosen approaches are feature extraction from ImageNet as the baseline, Transfer Learning as an improved version of the feature extraction, SimCLR and NNCLR for contrastive learning and the MAE with Vision Transformers and Autoencoders. Those cover the most recent and most promising domain advances and can be seen as good representatives.

Feature Extraction and Transfer Learning

The approach of extracting features from a pretrained network is straightforward. The two relevant components are the training dataset and the model architecture. After choosing them, the network is trained normally, and the head is removed. The approach with a ResNet50 pretrained on ImageNet as in Keisler et al. [Kei+19] is called Feature Extraction in this work and is used as a baseline. However, this approach can be improved by interchanging the two main components, namely the dataset and the network. The approach where components are exchanged is called Transfer Learning subsequently.

In recent years, many publications of neural network architectures could perform better on ImageNet than the basic ResNet50 architecture. Those networks probably provide better features for a nearest-neighbor search. A less specific smaller model could also yield better results. For other transfer learning approaches, there exists information in the literature that better-performing networks perform better

in transfer learning [KSB19]. Therefore, this might also be a promising approach for the neighbor search. Some possible alternative networks and their performance on ImageNet are shown in table 2.

Network	Parameters	Depth	Top-1 Accuracy
VGG16	138.4 M	16	71.3 %
VGG19	143.7 M	19	71.3 %
ResNet50	25.6 M	107	74.9 %
ResNet101	44.7 M	209	76.4 %
ResNet152	60.4 M	311	76.6 %
EfficientNetB0	5.3 M	132	77.1 %
Xception	22.9 M	81	79.0 %
EfficientNetB7	66.7 M	438	84.3 %

Table 2 Overview of some Potential Network Architectures[24]

The second component of the Transfer Learning approach is the dataset which can also be interchanged. The ResNet50, which is used as a baseline, was trained on the ImageNet dataset. This dataset does not consist of remote sensing data but normal images. It could be beneficial to extract features from a network directly trained on remote sensing data. To expect the best results, the normal requirements for datasets for the training of neural networks should be fulfilled, namely to be as big as possible and generalizable [HAE16].

Of the Datasets mentioned in subsection 3.2.5, the MLRSNET and BigEarthNet Datasets seem to fulfil the criteria the best by having the most patches and classes. Since this work focuses on images with high resolutions, MLRSNET seems to be better, having patches with resolutions of up to one pixel for 0.1 m in comparison to the 10m with BigEarthNet. Additionally, a primary class is provided instead of multiples in BigEarthnet. Therefore, the 2020 published MLRSNET dataset is chosen for the pretraining of a network and to extract features from it later on [Qi+20].

SimCLR

The approach named Simple Framework for Contrastive Learning of Visual Representations proposed by Chen et al. [Che+20a] consists of four major parts that enable it to learn useful representations. In the beginning, the data is augmented, and then a Base Encoder is used to get features. After that, the output is processed by a projection head, and the result of this is evaluated by the already introduced special loss function. In figure 10, the overall architecture is depicted. Denoting x as one sample image, $t \sim T$ as the augmentation function and $\tilde{x}_i$ as well as $\tilde{x}_j$ as the augmented samples. $f(\cdot)$ is the Base Encoder and h_i and h_j the extracted features that can

[24] https://keras.io/api/applications/

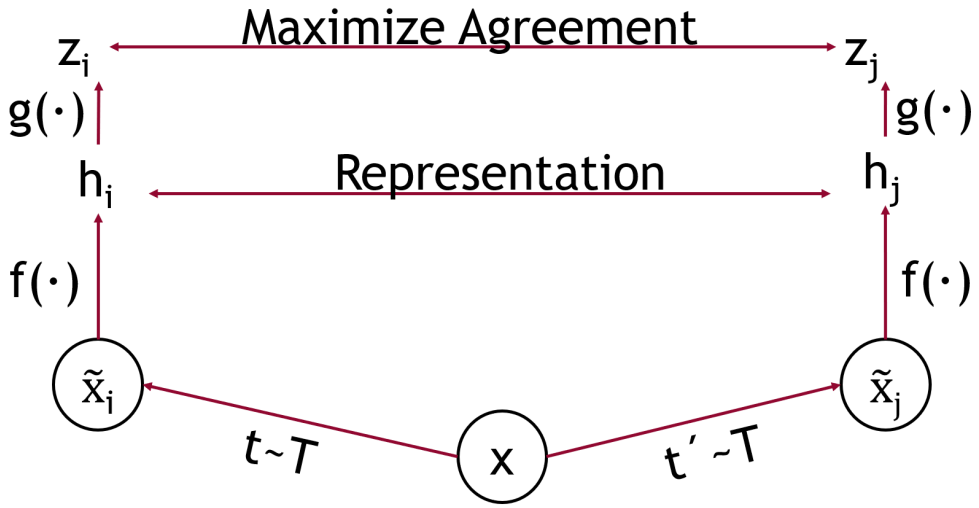


Figure 10 Architecture of SimCLR [Che+20a]

represent the images. After that, $g(\cdot)$ denotes the simple projection head, and z_i and z_j are the outputs of the projection head that are supposed to have high similarity. Those four main parts are explained in detail subsequently.

Data Augmentation

The first step of SimCLR is the stochastic augmentation of the data. By doing this twice, the two resulting elements $\tilde{x}_i$ and $\tilde{x}_j$ are created. Those two elements are seen as a pair.

To get representations that are actually useful for training, appropriate augmentations need to be chosen. Some relevant augmentations applied to remote sensing data can be seen in figure 11.

Figure 11 Exemplary Data Augmentations

In the paper, different augmentations were tried for ImageNet. The authors used combinations of two augmentations and predicted the accuracy of any combination with their approach. Especially color distortion, meaning an artificial change in the colors and the cropping of one part of the image and resizing the image again is crucial for a good result [Che+20a].

When adapting this approach, the suggested augmentations are most likely a good starting point for experiments. Nevertheless, domain-specific adaptions can be considered. For example, the ImageNet dataset consists of various vehicle classes. For vehicles, the color is not relevant, but the shape is. Therefore, a color distortion might be appropriate. Nevertheless, for remote sensing images, if grassland or agricultural soil should be distinct, the color is most likely more relevant.

Base Encoder

After the images were augmented, the next component is the Base Encoder. The Base Encoder is used to extract feature representations from the augmented images. The SimCLR framework is flexible with the Base Encoder, so all common network structures also used with image classification can be used. Therefore, those are the same as seen earlier in table 2. For simplicity and comparability, the authors of the paper decided to go with residual networks [He+15][Che+20a].

Projection Head

One could directly calculate a loss with the representations generated by the Base Encoder. Nevertheless, the authors of SimCLR decided to add a projection head as an additional step. It consists of a small neural network. The one used in the paper is a simple MLP that only consists of one hidden layer with the ReLU activation function. It is not used because of any necessity, but experiments have shown that the performance is superior if such a simple projection head is used [Che+20a].

Loss Function

The last part of the SimCLR framework is the contrastive loss function. For this, the approach uses the NT-Xent (the normalized temperature-scaled cross entropy) loss function, which was explained in detail in subsection 3.2.3. It is pretty standard in contrastive learning [Che+20a].

With those parts combined, the authors managed to acquire state-of-the-art results. They describe that many epochs, big models and huge datasets benefit the final accuracy much more than this is the case for supervised learning. Therefore, they use those for their experiments. The benefit of such configurations is also confirmed by a paper by the same authors further elaborating and improving the method [Che+20a; Che+20b].

NNCLR

Nearest-Neighbor Contrastive Learning of Visual Representations, another contrastive learning approach, builds on top of the idea of SimCLR. The main change is that in NNCLR similar elements are not artificially created with data augmentation, but the nearest neighbors are searched in the embedding space generated by the Base Encoder. Those identified neighbors are then seen as similar images and used for training. The authors claim to have improved the performance on ImageNet classification by 3.9 % in comparison to SimCLR with this approach [Dwi+21].

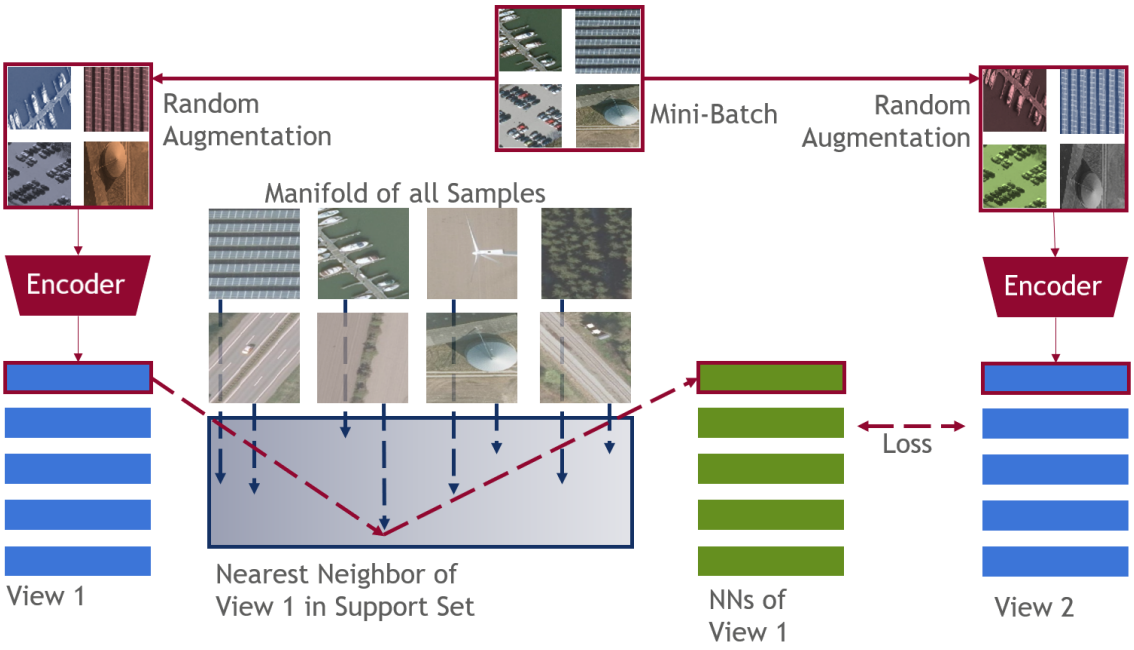


Figure 12 NNCLR Architecture after Dwibedi et al. [Dwi+21]

To conduct the approach, the first step is the finding of neighbors. It begins with the creation of a support set, which contains a number of random samples of the dataset. The size of this support set should be large enough to always have at least one element of every class (the paper intends to be used on a classification dataset) in it and can thereby represent the whole dataset. The support set is sampled over time new to gain different neighbors [Dwi+21].

After the definition of the support set, a mini-batch is drawn. All the elements in the minibatch are augmented with the same augmentation methods twice. For the augmentations, the authors point out that they are beneficial but less crucial for the performance compared to SimCLR. Then, the vectors representing the images are calculated with the Base Encoder. After that, for the vector of the first augmentation, the nearest neighbor is searched in the support set. Once it is found, the loss is calculated with the vector representing the second augmentation of the same original image and the augmentation of the found nearest neighbor from the support set in

combination with the augmentations of all images in the minibatch as negatives. The approach can be seen in the visualization in figure 12 [Dwi+21].

The nearest neighbor is found by using the element with the highest similarity to the query. The formula for this is [Dwi+21, p.4]:

$$NN(z, Q) = \underset{q \epsilon Q}{arg\,min} ||z - q||_2 \tag{3.3}$$

The loss function used for NNCLR is the same as for SimCLR with the adaption to the situation, that the similarity is not calculated between augmentations, but between the identified and augmented nearest neighbors and other elements. Therefore, the complete function is defined as [Dwi+21, p.4]:

$$\mathcal{L}_i^{NNCLR} = -log \frac{exp(NN(z_i, Q) \cdot z_i^+ / \tau)}{\Sigma_{k=1}^n exp(NN(z_i, Q) \cdot z_k^+ / \tau)} \tag{3.4}$$

Same as in SimCLR, all the embeddings are normalized by l_2 normalization, which is done before the neighbor search [Dwi+21].

Using such an approach with neighbors as positive samples instead of augmentations helps to learn not only the semantic variations in the augmentations but actual variations from the dataset, which increases the diversity of pairs the method should predict similar outputs for [Dwi+21].

Masked Autoencoder

The masking idea of the last chosen method, namely Masked Autoencoders, is not new and was facilitated in Bert [Dev+18], where some of the inputs were masked out. But in the domain of image feature extraction, this approach had a negligible role. This was also dependent on the fact that usually, in image processing tasks, for many years, Convolutional Neural Networks were state-of-the-art. Those make it complicated to mask parts of the input and compute useful things. All this changed with the upcoming Vision Transformer, which naturally divides an image in patches [He+21].

With this division in patches, the masking can simply be done by removing some of those patches. While in text processing applications, usually around 15 % of the data in for example Bert [Dev+18] is masked out, for MAEs for vision, the authors state that 75 % would be a good proportion. Selecting patches for masking randomly and not systematically had the best performance in experiments and was thereby used. After masking, the Autoencoder's task is to recreate the whole image by predicting

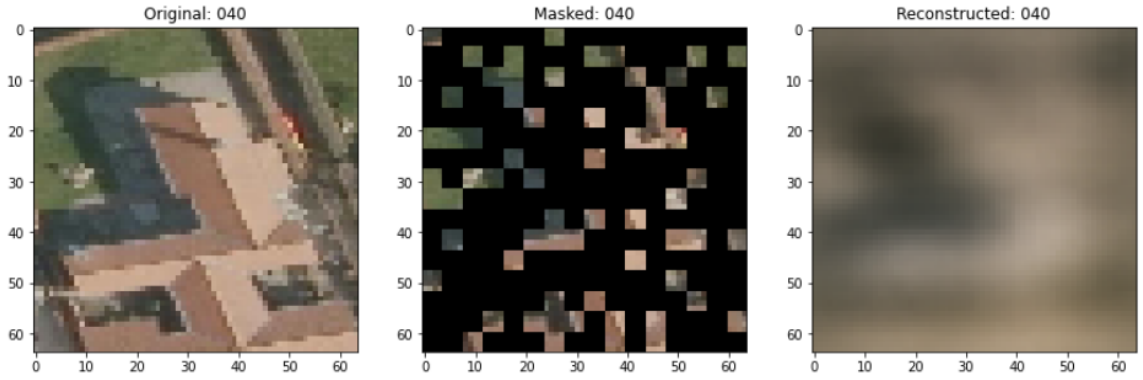


Figure 13 Example of an Image, its Masking and the Reconstruction

the missing patches [He+21]. An example of the masking and one reconstruction can be seen in figure 13.

The overall architecture of the approach is depicted in figure 14. After masking, only the remaining patches are fed into the Encoder. No masked tokens or similar things are used. The unmasked patches are then encoded into the hidden space. After encoding, the missing masked tokens are inserted as a dummy to let the Decoder know to reconstruct also the formerly removed patches. Those masked tokens also get positional embeddings for the Decoder to know where to reconstruct what. After that, the image is reconstructed by several Transformer blocks in the Decoder [He+21].

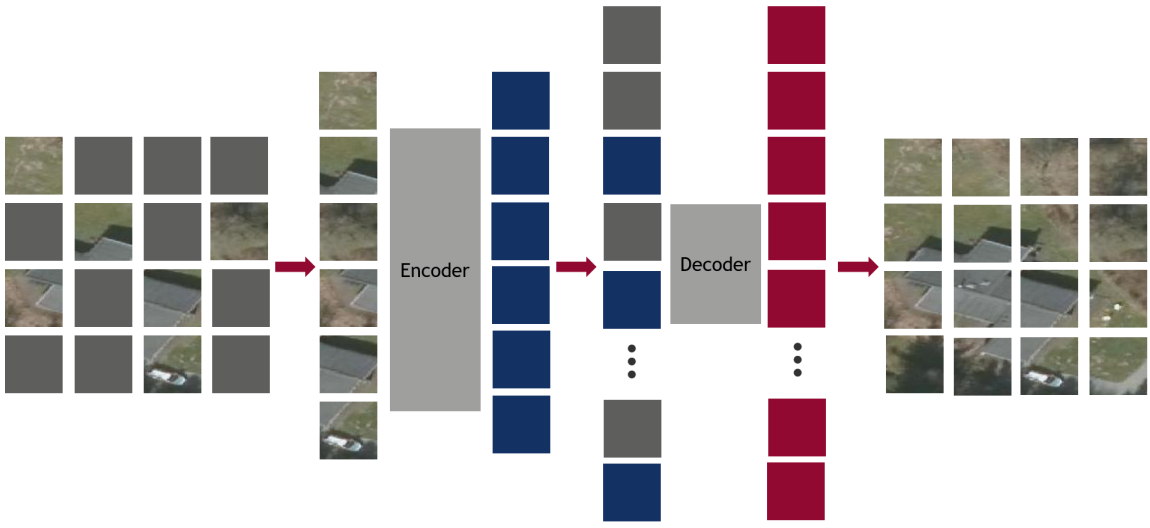


Figure 14 Architecture of Masked Autoencoder [He+21]

For feature extraction, the most relevant part is the Encoder since the Decoder is only used for the reconstruction. The interesting feature vectors are the ones in the latent space after the encoding step. Therefore, the authors state, that the Decoder can be much more shallow than the Encoder with just a few layers resulting in a significant reduction of runtime since in the more complicated Encoder, just a fraction of data

is computed. In the Decoder where the whole images are processed, the complexity is comparably low [He+21].

Regarding the paper by [He+21], the approach is highly capable and achieves 87.8 % Top-1 Accuracy on the ImageNet dataset[25]. Nevertheless, it should be kept in mind that the training is described as not trivial and challenging, especially without pretraining. Usually, huge amounts of data are needed to yield meaningful results [He+21]. This aligns with the findings of the original Vision Transformer paper that also suggests that pretraining on large amounts of data would be necessary to achieve excellent results [Dos+20].

3.4 Nearest Neighbor Search

The previous methods have shown how images can be transformed into a vector representation conveying semantic information. In the next step, ways to compare vectors are presented. After introducing general distance functions, a method from literature is introduced that can do a nearest neighbor search in millions of vectors in less than a second. Thereby, it fulfils the constraint described in the problem description.

Similarity Measures

The idea of a similarity measure is to have a numeric indicator of how likewise two elements are. The similarity should be high if the two elements are nearly the same and low if the two elements are completely different. The opposite of the similarity is the distance, which indicates how different the two elements are. Having similarity and distance as the exact opposites of each other, one can calculate the similarity with $similarity = 1 - distance$ and the distance with $distance = 1 - similarity$ [LRU14, p.92].

Regarding Leskovec, Rajaraman, and Ullman [LRU14], a distance measure has three additional properties:

- A distance measure is always a positive value except for comparing an element with itself. Then the distance is zero [LRU14].
- Distance measures are symmetrical, meaning that element M has the same distance to element N as N has to M [LRU14].
- The shortest path from one point to another is the direct one. Detours about other points must not be shorter [LRU14].

[25] https://www.image-net.org/

Those aspects also hold analogously for similarity with the difference that a perfect match has a similarity of one, and only complete dissimilarity would have a value of zero [LRU14].

Vector Comparisons

In the given scenario of comparing decimal vectors, the most prominent measures are the Euclidean Distance and the Cosine Similarity, which both do not simply compare the elements one by one in the vectors.

Euclidean Distance

The Euclidean Distance is formally defined as [LRU14, p.93]:

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{3.5}$$

Meaning for every entry, the values are subtracted one by one from the other element, and the result is squared and summed up. After that, the square root of that sum provides the Euclidean Distance. The Euclidean Distance can also be seen as a distance of two points in the n-dimensional space [LRU14].

Cosine Similarity

For the Cosine Similarity, a distance is not directly calculated between points, but the idea is to compare the angles between the two straight lines created by the vectors. Therefore the definition is [HKP12, p.78]:

$$\frac{a \cdot b}{||a||||b||} \tag{3.6}$$

This measurement is especially helpful if the vectors have a different length and this length is not supposed to be regarded [LRU14].

Categories of Algorithms for Nearest Neighbor Search

When a suitable approach for nearest neighbor search needs to be identified, three things categorizing those approaches should be considered.

To build an application setting, first, a distance measure needs to be chosen. Once this is done, one can decide whether exact results are needed or whether approximate neighbors are sufficient. This is especially relevant for high dimensional data since it can get computationally infeasible to calculate exact values for those [ML14]. The last property highly influencing the runtime is the number of neighbors that are

identified, so whether just the most similar element is searched like in Keller and Gray [KG85], or whether for every element, the similarity to a particular candidate is required [Cla83].

FAISS

Searching through the literature and recent publications, various methods seem feasible to meet the requirements of finding neighbors in real-time. One example is Spotify Annoy[26] from Li et al. [Li+16], which is used in the music service application with the same name. Another possibility is to use tree structures, like k-d-trees, that can provide fast search after initially creating a tree structure [Ben75]. Also common for searches is the use of hashing as either a concept for indexing in Locality Sensitive Hashing or as a concept of storing data as applied in Semantic Hashing where the data is stored in a particular area of disk regarding their content [SH07]. Several of those approaches seem feasible and can most likely fulfil the formulated requirements. For the application at hand, the algorithm by Johnson, Douze, and Jégou [JDJ17] called FAISS is tried due to its recency and very promising results in the paper.

The chosen approach was published in 2017 and uses the l_2 norm in the form of the squared Euclidean Distance. It can work with an arbitrary number of neighbors and uses approximations for faster comparison [JDJ17].

Using the approach, first, an index structure is built. This is the basis to enable the fast search. In the use case of near neighbor search for satellite data, the runtime of this index creation is not highly relevant. The remote sensing data is not usually changed, so waiting some time to get a new index structure is no problem. Once the index is built, the approximate search can be conducted [JDJ17].

Regarding performance, the approach is state-of-the-art. Referring to the paper, the method can beat older ones by a factor of 8.5 compared on the Sift1M[27] dataset and returns results in just a few milliseconds [JDJ17].

[26] https://github.com/spotify/annoy
[27] https://www.tensorflow.org/datasets/catalog/sift1m

4 Experimental Setup

With the identified methods usable for image representation, the practicability of the nearest neighbor search can be evaluated and the second research question addressed.

4.1 Dataset Description

To evaluate the search, a dataset of the whole country of Denmark is available. It has a size of 3.5 TB of data with a resolution of one pixel for every 0.125 m. With this detailed resolution, even smaller objects like cars, trees and houses can be recognized. The dataset consists of the three colorbands red, green and blue. No infrared-bands are included. All images refer to one point in time, and no multi-temporal data is available. In figure 15, an excerpt of the dataset is visualized. It measures $0.5\,km^2$ in size and displays agricultural fields on the left and a couple of houses on the right.

Figure 15 Exemplary Cutout of Denmark Dataset

The total size in pixels of the dataset is 2018483 by 4486338. All presented methods work on small images. Therefore, it is necessary to break the big image down to be able to use it for the methods. For this, a quadratic size of 256 pixels is chosen to align with the MLRSNET dataset that is supposed to be used for comparisons. The chosen size results in a total amount of 138177444 cutouts, each having a size of 32 m by 32 m. Of those cutouts, only 42775683, so roughly 31 % are actually covered by land. The others are covered by water. The number of cutouts aligns with the official size of Denmark of $42.920\,km^2$ [28] resulting in a theoretical amount of cutouts with data of 41914063 cutous.

[28] https://www.laenderdaten.info/Europa/Daenemark/index.php

A statistics department in Denmark[29] provides detailed insights about the land use and land cover percentages of Denmark. The values are shown in table 3. The majority of the country is filled by agricultural crops, followed by forests and other natural surfaces. This results in the fact that roughly 84.3 % of the whole country is filled by some kind of natural surface. Most of those natural surfaces have very monotonous structures, as can be seen in the sample in figure 15. This unbalance needs to be kept in mind for the training in the later steps since it can be assumed that for users of the service, interesting cutouts are those with objects and not patches solely brown(agricultural), green(forest, field) or blue(water).

Type	**Proportion**
Roads, railroads and runways	5.6 %
Buildings and built-up areas	7.6 %
Other artificial surfaces	1.0 %
Agricultural crops	59.0 %
Forest	13.3 %
Nature (dry and wet habitat types)	9.2 %
Lakes and streams	2.8 %
Unclassified	1.6 %

Table 3 Table of Percentage Distribution of Landcover of Denmark

4.2 Experimental Setting

To allow reproducibility, the general setup for the experiments is described. In the beginning, the preprocessing choices for the dataset are shown. This is followed by the general setup of the development environment and the code. In the end, the chosen evaluation methods are explained.

4.2.1 Dataset Preprocessing

The dataset is provided in one 3.5 TB sized file of the type *.tif* and first needs to be cut down in cutouts. Due to limitations of computational resources, the whole dataset is not split up and saved redundantly beforehand, but only the position of the top left corners of every cutout is saved. During this saving of the coordinates, only valid patches are selected, meaning that patches of the ocean are filtered with the *numpy.count_nonzero* command since they are just black in the dataset. Similar is the approach for some completely white patches, which likely refer to data errors. Those are filtered by the *numpy.std()* command assuming that remote sensing images do not contain only precisely one color but at least slight deviations in the color. So patches with a standard deviation of zero along all bands are removed.

[29] https://www.dst.dk/en/Statistik/emner/miljoe-og-energi/areal/arealopgoerelser

As stated in section 4.1, the Denmark dataset is very unbalanced consisting mainly of natural patches and especially agricultural surfaces. It is assumed that those will not be the prior interest of the usage of the nearest neighbor search and could be a problem during training [CJK04]. Also, in the other datasets introduced in subsection 3.2.5, the main focus is on cutouts with objects. Unfortunately, for the Denmark dataset, no labels are available, making it impossible to filter such cutouts accurately.

One option to filter the dataset anyways is to filter the patches by general features that do not need any supervision. Possibilities would be the use of the colors in a color histogram or the standard deviation of the images. The approach with a filtering based on the standard deviation created very promising results. The idea is that cutouts with a high standard deviation tend to have multiple different colors in the cutouts and thereby contain multiple objects or an object and a background. The cutouts with a low standard deviation tend to contain just one color and not many different-looking shapes, colors and objects. Therefore, the cutouts are filtered regarding their average standard deviation, averaging the standard deviation of the three individual colorbands. Some examples of this filtering are shown in figure 16 and make it clear that images with high standard deviations are perceived by a human as more interesting. The proportion of how many cutouts exist in regard to the filtering can be seen in table 4.

	std>0	std>10	std>20	std>30	std>40	std>50
Proportion	100.00 %	63.35 %	29.02 %	10.05 %	2.70 %	0.66 %

Table 4 Proportion of Standard Deviation in Images

To be able to access patches with different levels of standard deviation fast, the used index for valid cutouts was built for every standard deviation between 0 and 50 in steps of 10 separately.

If a machine learning task is conducted, only the needed amount of patches with valid information and the requested standard deviation is extracted from the big Denmark file and saved locally on an SSD. The intermediate step of saving the cutouts on an SSD is not mandatory but overcomes the need of extracting them from the big *.tif* file in every epoch and reduces computation time significantly.

During a machine learning training task, the data can then be read from disk with the *flow_from_directory* method from the Keras *ImageDataGenerator* class to read the data subsequently since for big datasets, the data does not fit in memory. Nevertheless, for smaller amounts of data and downscaled cutouts, the whole dataset fits directly in memory.

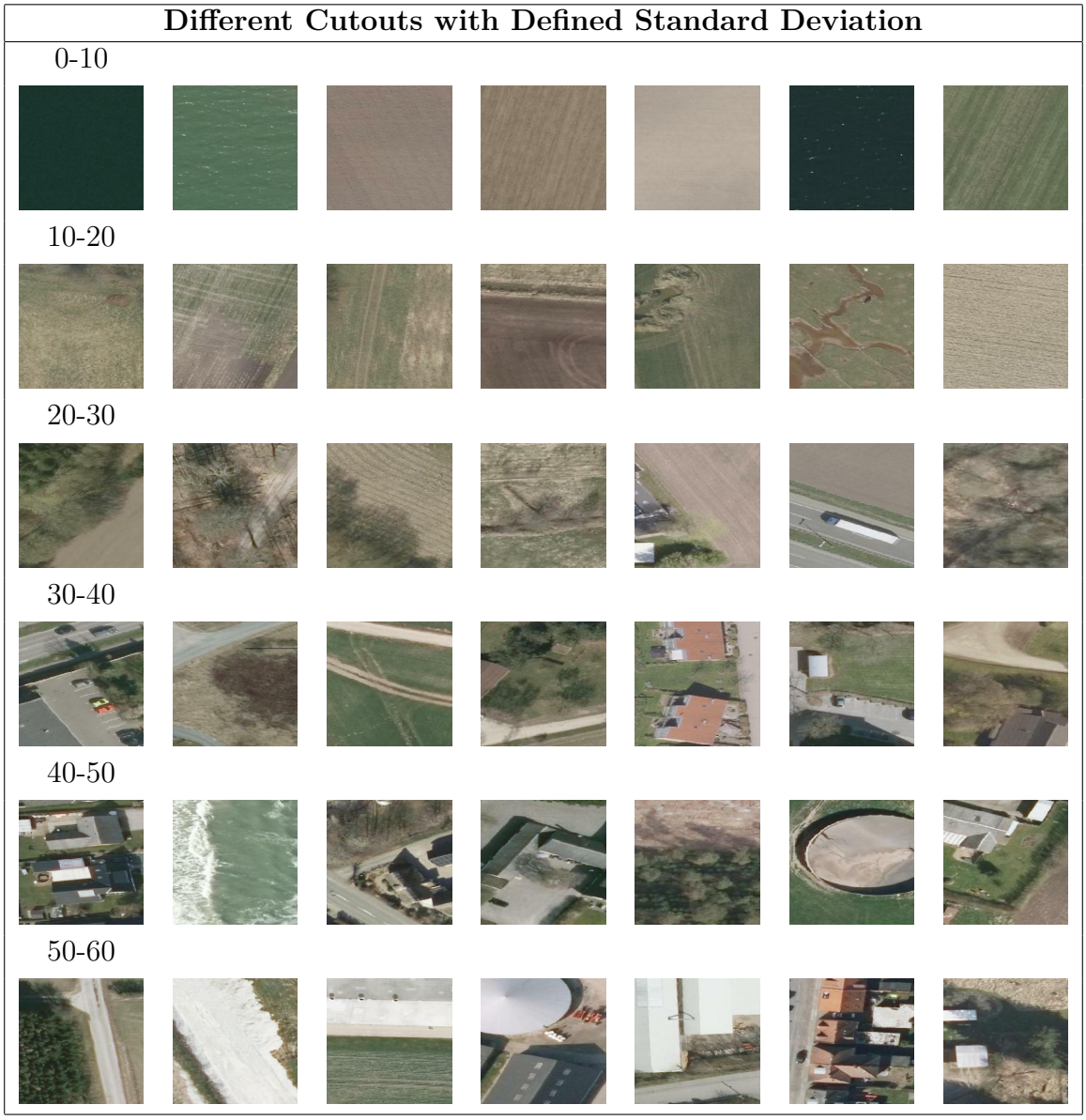


Figure 16 Examples of the Standard Deviation Filtering

Filtering of Corrupted Data

After the filtering of the data regarding the standard deviation, it can be noticed that several patches are corrupt. Only a part of these corrupt images is covered by remote sensing data, and the other part is either completely black or completely white. Some examples of this can be seen in figure 17.

Those corrupted cutouts are not regarded as helpful and should be filtered out. For this, the amount of entirely white or completely black pixels was counted per image and cutouts with more than 1000 black or white pixels were discarded. With this comparably high threshold, it can happen that cutouts with just a small part of corrupted data stay within the dataset. Nevertheless, this high threshold was chosen to keep cutouts that naturally have a lot of black or white in them. Exemplary

Figure 17 Examples of Corrupted Cutouts

cutouts slightly below the threshold can be seen in figure 18. In total, only 0.6 % of the cutouts are corrupted, but this number rises with the standard deviations and can be seen for different filterings in table 5.

	std>0	std>10	std>20	std>30	std>40	std>50
Corrupt	0.6 %	0.9 %	1.8 %	4.6 %	15.0 %	55.7 %

Table 5 Proportion of Corruptions after Standard Deviation Filtering

Figure 18 Cutouts Slightly Below Corruption Threshold

4.2.2 General Aspects

The described experiments were carried out on a machine running Linux in version 20.04.4 LTS. As CPU, the AMD EPYC 7402P 24-Core Processor with 48 logical cores could be used. The GPU of the system was the RTX 3090 with 24GB of VRAM and CUDA version 11.7. The system was equipped with 192 GB of DDR4 RAM.

All coding was done in python 3.8.10 using Jupyter Notebooks and python files. The main framework for machine learning used was Keras[30] since also most of the method papers like SimCLR [Che+20a] were initially programmed using it. Moreover, Keras provided some notebooks[31] with practical implementations that could be facilitated.

For the management of the satellite data, rasterio[32] was used to access the *.tif* files and to extract the image cutouts. Another main library was numpy[33] for managing

[30] https://keras.io/
[31] https://keras.io/examples/
[32] https://rasterio.readthedocs.io/en/latest/
[33] https://numpy.org/

the images as arrays and other supportive features. For visualizations, pyplot from matplotlib[34] was used.

4.2.3 Code Foundation

The work uses several sources for the implementations. The supportive functionalities for accessing the data originate from a project seminar at the University of Münster. For the code for the models, the sources are shown in detail subsequently in combination with a description of additional functionalities to comprehend the approach to the experiments.

Optimizers

The stochastic gradient descent optimizer and Adam optimizer could be used directly from the TensorFlow library. For the Lars optimizer, the implementation was available in the GitHub repository[35] of SimCLR. The formula $LearningRate = 0.3 \cdot BatchSize/256$ was used based on Chen et al. [Che+20a] to calculate the value for the learning rate.

The fourth optimizer, namely the AdamW optimizer, was used from the Tensorflow addons library. It was used with a learning rate schedule that was already used for similiar tasks by Keras[36]. It is an adoption of one from Kaggle[37]. It uses a warmup cosine in the beginning and then cosine annealing afterwards. The learning rate per step for 50 epochs and 100000 datapoints with a batch size of 512 is shown in figure 19.

Pretrained Feature Extraction

The authors of the paper by [Kei+19] did not provide the code they used. Anyways, direct feature extraction can be just conducted with Keras applications[38].

Transfer Learning

For the training of a custom setting for feature extraction with remote sensing data and other networks, the basic Keras implementations partially used in the Keras image classification[39] could be adapted, and the models available in the Keras application used. To get the best-performing model, Keras *ModelCheckpoints*[40] were

[34] https://matplotlib.org/
[35] https://github.com/google-research/simclr
[36] https://keras.io/examples/vision/masked_image_modeling/
[37] https://www.kaggle.com/ashusma/training-rfcx-tensorflow-tpu-effnet-b2.
[38] https://keras.io/api/applications/
[39] https://keras.io/examples/vision/image_classification_from_scratch/
[40] https://keras.io/api/callbacks/model_checkpoint/

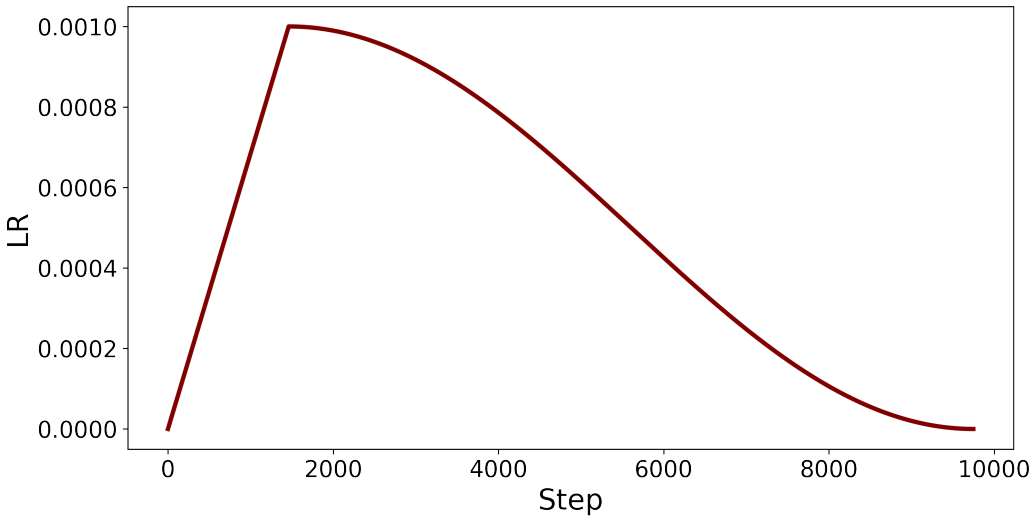


Figure 19 Visualisation of Learning Rate Scheduler used with AdamW

facilitated to save the model after the epoch with the best validation accuracy. This was to prevent problems with overfitting and fluctuating accuracies. To get the intermediate representation without the head, the values at the last layer before the head were extracted and combined with a *GlobalAveragePooling2D* layer.

SimCLR

For SimCLR, there is a basic implementation available at Keras[41]. Additionally, the authors of Chen et al. [Che+20a] provide a version in their GitHub repository[42]. This code was adapted for the given scenario. Especially the semi-supervised approach with optimizing for classification was removed, and the notebook was transformed to work in a way without any labels, focusing on meaningful pretraining. For the different networks, the Keras applications[43] were used. As color augmentations, the suggested ones consisting of horizontal flips, zooming, random color affine and random translation were applied. Some augmented images can be seen in figure 20.

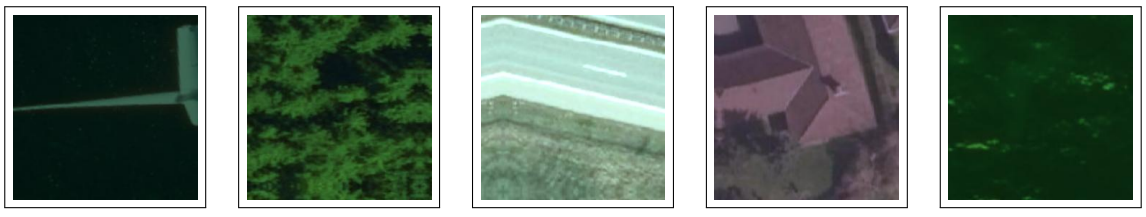


Figure 20 Visualization of Augmented Cutouts

To achieve good results with this approach, a so-called classification augmenter is used during the extraction of the features. This means that before vectors are extracted

[41] https://keras.io/examples/vision/semisupervised_simclr/
[42] https://github.com/google-research/simclr
[43] https://keras.io/api/applications/

for the nearest neighbor search, the same augmentations that were used for training are applied in a weaker version to the cutouts.

NNCLR

For NNCLR the situation was quite similar to the one in SimCLR. There is also a basic implementation available at Keras[44]. Unfortunately, the authors did not publish the full code themselves, but some parts like the optimizers could be used from the SimCLR GitHub repository[45], and additionally, some other third-party installations exist that could be facilitated for comparisons and mostly seemed to align with the Keras implementation. Again this code was adapted, especially in regards to using it without labels. Also, again the Keras applications[46] were used as Base Encoder.

Masked Autoencoder

For the MAEs, a basic implementation also exists in Keras[47]. The probing part for further training with labels was removed, and an extractor was built that consists of the definition of the input, the augmenting for testing, followed by the patching of the image, transforming it to be usable in the Vision Transformers. After that, the Encoder creates the latent representation of the images, and the outputs are processed by a *BatchNormalization* and *GlobalAveragePooling1D* layer.

Because of computational limitations with larger models, a high batch size of 512 could not be used with the intended Vision Transformer Base architecture from the paper of Dosovitskiy et al. [Dos+20]. Nevertheless, since this approach does not use a contrastive loss function, this should not significantly impact the accuracy.

FAISS

For FAISS, a library was provided by the authors directly on GitHub[48]. For all experiments, only the CPU approach was used. As parameter configuration, the *nlist* parameter was set to 20, and as quantizer, the FAISS *IndexFlatL2* was facilitated.

4.2.4 Overall Project Structure

The project structure can be divided into two big components. The first component consists of the parts necessary to process the data, train a model and evaluate it. The second component is the process triggered if a search is started. This second

[44] https://keras.io/examples/vision/semisupervised_simclr/
[45] https://github.com/google-research/simclr
[46] https://keras.io/api/applications/
[47] https://keras.io/examples/vision/masked_image_modeling/
[48] https://github.com/facebookresearch/faiss

part of an actual search is visualized in figure 21 for MAE, SimCLR and NNCLR. The Transfer Learning approach works analogously with the different datasets.

Training and Evaluation

In the beginning, the image of Denmark is divided into cutouts of the predefined size of 256 by 256, and the indices of those are saved. After that, the indices are filtered regarding the dataset preprocessing to only keep indices for cutouts that fulfil the preprocessing requirements. After that, those cutouts are extracted, sampled down to the intended size of 64 by 64 pixels and saved on the disk.

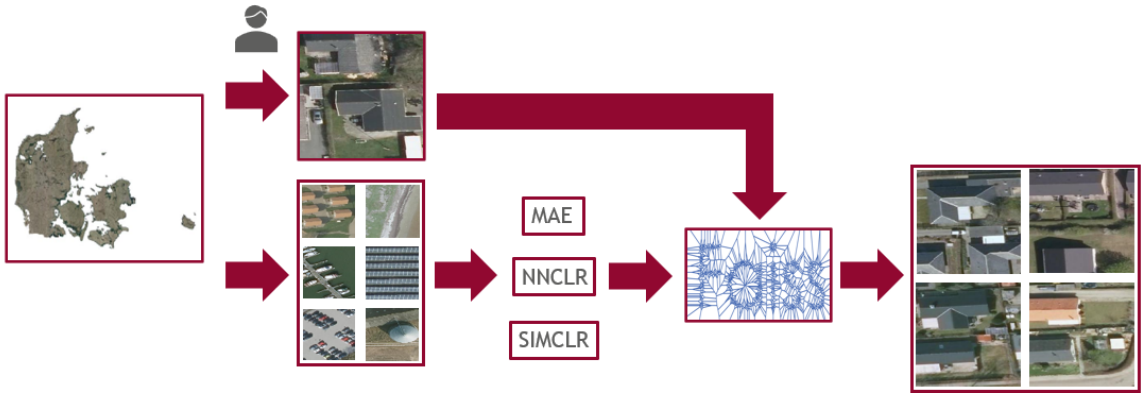


Figure 21 Overall Architecture of Feature Extraction and Neighbor Search

The next step is the training of a model. Therefore, one of the models is chosen and trained with the 64 by 64 sized predefined patches. Once this is done, the evaluation cutouts are taken, also sampled down to 64 by 64 pixels, and the evaluation takes place. This step is described in more detail in section 4.3.

Once a trained model is available, the second part, namely the search, can take place. For this, an arbitrary subset of the Denmark dataset is chosen. The part can also contain the whole country of Denmark. Nevertheless, for the reason of computational feasibility, just a subset was chosen. For this subset, all cutouts are extracted and sampled down to the size of 64 by 64 pixels, and after that, features are extracted for each cutout. Once this is done, FAISS is used to build an index structure to enable the fast search with the datapoints. In the end, a user can select an arbitrary cutout. The referring feature vector is then used as a query in the FAISS index structure, returning the indices of the closest neighbors. The cutouts for the indices are then accessed in the dataset and returned to the user.

4.3 Evaluation Methods

The approaches used to evaluate the results of the models are described subsequently. Because the main dataset of Denmark has no labels, first, the manual creation of a

labeled dataset is described. After that, the method for calculating the accuracy in the specific case of neighbor search is explained. The section concludes with a description of the used evaluation pipeline.

Manually Labeled Dataset

For the main dataset of Denmark, no evaluation data is available. Simply using values like contrastive accuracy and contrastive loss is not appropriate since they are batch size-dependent, and the loss also depends on the τ parameter. Therefore, those metrics are not appropriate for comparing different settings. Additionally, it is beneficial to test somehow with a setting very close to the final application. This led to the decision to manually create a dataset.

During the dataset creation, the question arose of how a dataset for neighbor detection would need to be constructed. Since it is not possible for a human to decide numerically if images are similar or not, a more generalized approach is needed. In the end, the decision was made to create a labelling for classification to build a custom method for getting an accuracy value on top of this. This has the advantage that the approach would also be usable with other classification datasets like MLRSNET. The created dataset consists of 500 cutouts in 25 classes. Examples of the classes can be seen in figure 22.

Accuracy Score Calculation

The general idea of the accuracy calculation is that if two cutouts are in the same class, they are defined as similar. The score is then computed by calculating for every cutout a number of neighbors, and after that, it is compared to how many of those calculated nearest neighbors are in the same class. In the given case, for each of the 500 patches, ten nearest neighbors are calculated with the extraction methods. After that, the fraction of cutouts of the same classes in those 5000 returned images with the respective class as query is calculated.

The code for the extractor in Python is shown in algorithm 1. After the first two initialization lines, in line 2, the intended extractor is used to transform the test data in the vector space. After that, a counter is initialized in line 4. The loop starting in line 5 iterates over every feature vector representation of the cutouts. Then in lines 7 and 8, the similarity of the element of the iteration is calculated with every other element in the feature vector list. After that, the similarity list is sorted stable with an index in lines 10 and 11, meaning that a list is created indicating which indices have the highest similarity to the candidate element. After that, from line 13, n neighbors are taken, and it is checked whether the candidate element and the n

Figure 22 Classes in Evaluation Dataset

neighbors have the same class. For every neighbor where it is the case, the counter is raised. In the end, the counter is divided by the number of neighbors and the length of the testset to get an average similarity value.

Such an approach has the following advantages:

- Accuracies can be compared for easier and harder patches separated.
- Every patch has multiple neighbors and not only a single one.
- With exact classes, neighbors can be precisely distinct.

Algorithm 1 Code of the Used Evaluation Calculation

```
def evaluate_extractor(extractor, x_test, y_test, neighbors=10):
    test_vec=extractor(x_test)

    counter = 0
    for elem_idx in range(0,len(x_test)):
        min_list = []
        for element in test_vec:
            min_list.append(cosine(element,test_vec[elem_idx]))

        indices_red = range(len(min_list))
        a, indices_red = zip(*sorted( zip(min_list, indices_red)))

        for index in indices_red[0:neighbors]:
            if np.array_equal(y_test[index], y_test[elem_idx]):
                counter += 1

    return(counter/neighbors/len(x_test))
```

FAISS Evaluation

Next to the runtime of a search and the index building in FAISS, also the quality should be assessed numerically to have values that can be compared precisely. Therefore, two evaluation values are used.

First, the accuracy was tested by taking the first n neighbors of the brute force search and comparing them with the first n elements of the approximate FAISS search. Afterwards, it was checked how many neighbors were identical and only in a different order within those top n elements.

Second, the ratio was calculated, how much worse, the sum of all distances of a predefined number of neighbors to the query is, if they are calculated with FAISS instead of the brute force approach. With this, it can be checked whether the approach has difficulties finding the best neighbors but still finds good ones.

Evaluation Pipeline

For easier evaluation of the different model configurations, a Jupyter notebook was created that makes it easy to just exchange the models and then run all evaluations. One can decide to either evaluate with the Denmark dataset or the MLRSNET dataset for comparison. The values of the evaluation on the MLRSNET dataset could

also be used in the future for comparisons in the literature. The pipeline consists of the following seven steps:

Extraction Block: At the beginning of the notebook, the test dataset and model is loaded, and the feature vectors are calculated for every cutout of the testset.

Calculating Neighbor Accuracy: In the next step, the neighbor accuracy is calculated in regards to the function explained earlier in algorithm 1.

Neighbor Accuracy per Class: The next value that is calculated is the neighbor accuracy per class. This is the normal neighbor accuracy calculated for every class individually. Therefore, it can be analyzed whether models achieve good accuracies in certain classes.

2-D Plot of Vectors: After that, the vectors of the testset cutouts are reduced in dimensionality and visualized in 2-D. Therefore, the dimensionality reduction approach UMAP by McInnes, Healy, and Melville [MHM18] is used. UMAP is chosen to conserve the global structure [MHM18]. With this visualization, a deeper understanding of the approaches of the algorithms can be achieved.

Neighbor Accuracy Mistakes per Class: In this step, it is counted for each class, to which class the cutouts belong that were identified as neighbors. With this, it can be seen which classes are confused with each other.

Visualization of Neighbors: After the numerical evaluations with the testset, random cutouts are shown with their most similar calculated neighbors from the testset and a not preprocessed dataset representing parts of Denmark.

Evaluating FAISS: In the final step, Faiss is evaluated in regards to the runtime and with the two beforehand introduced measurements.

4.4 Base Model Parameters

The general parameters described in table 6 were chosen as a basic setting for the comparison of the models. They are used for training and comparison of all models in the next chapter if not stated otherwise.

Dataset filtering	Batch Size	Optimizer	Dataset Size	Cutout Size
avg_std 30	512	AdamW	100000	64

Table 6 Parameters of Base Model

5 Interpretation of Results and Adaption

This chapter deals with the results and insights of the conducted experiments. In the beginning, the different parameters of the base model are analyzed in detail, and the reasons for the distinctive selection are explained. This is followed by the presentation of the results of the methods with those base parameters. After that, the results are interpreted, and reasons for better and worse performances are analyzed. Finally, all gathered insights are combined into a final model that is afterwards interpreted in the general context of the domain and machine learning.

5.1 Elaboration on Parameters for Experiments

The basic parameters were analyzed in isolation. This was done to identify the best ones in the given setting and to elaborate on specialities.

5.1.1 Downsizing of Images

In subsection 3.2.3, the most common loss function of contrastive learning is explained, and the significant impact of large batch sizes for good model qualities is assumed. Since the computational resources are limited, comparably large batch sizes above 100 are not possible with the full images of 256 by 256 pixels.

To be able to conduct those tests despite that, the images can be scaled down. The size of 64 by 64 showed promising results. It enables training the model with a batch size of 512 along all for this work relevant configurations of contrastive learning and is still visually precise enough to recognize the structures. Some images exemplary downsampled can be seen in figure 23.

The still high quality of evaluations based on the downsampled images can also be confirmed by the baseline performance of feature extractions seen in table 7, which decreases just roughly 4 % if the images are sampled down to the resolution of 64 by 64.

In addition to the benefit of being able to use several different batch sizes, the downsampling also enables it to train the models faster with the larger batch sizes, reducing the time per epoch from approximately 15 minutes to one minute.

Figure 23 Exemplary Downsampled Images

	32	48	64	96	128	192	256
Accuracy	47.52 %	54.68 %	59.52 %	62.36 %	63.38 %	62.66 %	63.04 %

Table 7 Neighbor Accuracy of Differently Downsized Feature Extraction from ResNet50 Pretrained on ImageNet

5.1.2 Optimizer

Along the four chosen optimizers, the AdamW optimizer with a learning schedule performed better than without a learning rate schedule and best for the general setting, achieving 56.82 % accuracy on the testset. The Adam optimizer achieved 52.22 % and the Lars optimizer 51.3 %. Thereby, the suggested Lars optimizer could not keep the promise of achieving the same accuracy with larger batch sizes as the Adam Optimizer [Che+20a]. The older stochastic gradient descent approach could only achieve 48.16 %.

Regarding the superiority, for experiments with up to 100000 cutouts, AdamW is the first choice for all approaches. Unfortunately, the AdamW optimizer and Lars optimizer showed degenerative behavior in some cases if a setting different to the basic one was used. In settings where this occurred, the Adam optimizer was also applied to have a comparable value. It is explicitly mentioned in the upcoming sections if this was the case.

5.1.3 Batch Size

In subsection 3.2.3, challenging aspects in the context of the batch size were already mentioned. Those assumptions could be proven experimentally. By downsizing the images, different batch sizes could be tried, and the referring neighbor accuracies for all sizes between 32 and 1024 in steps of factor two were calculated.

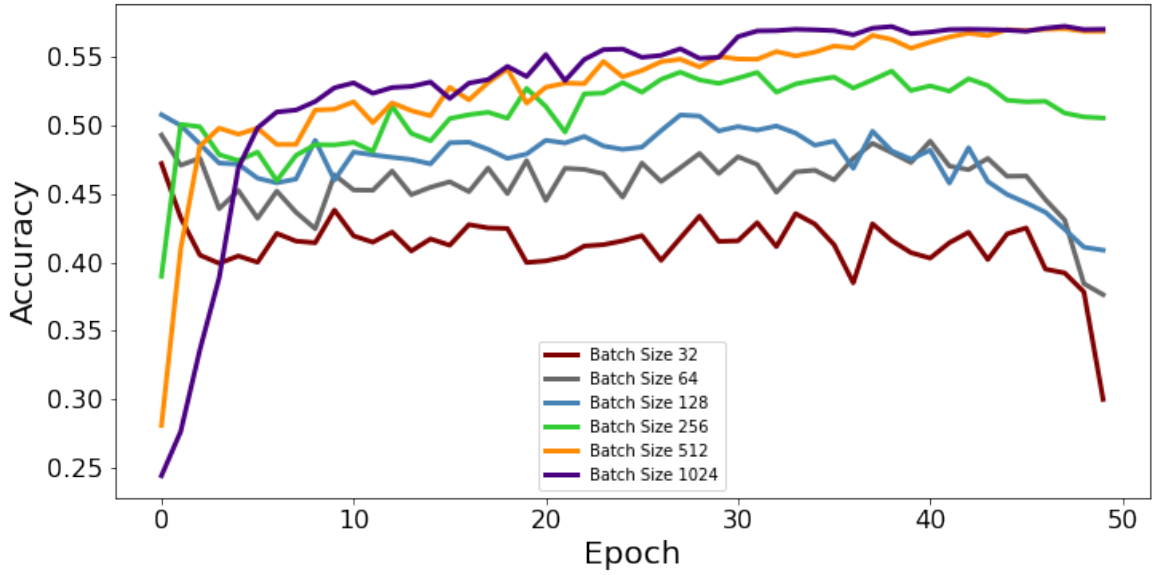


Figure 24 Comparison of Different Batch Sizes

In figure 24, several interesting effects can be seen. First of all, the final accuracy after 50 epochs is clearly rising with larger batch sizes. However, especially the training curves are of interest. For the larger batch sizes, a clear improvement can be seen. The accuracy is rising with the epochs. Nevertheless, for the small batch sizes, the accuracy climbs to a decent level in the beginning but then degrades after several more epochs.

This degenerative effect is most significant with the chosen Optimizer AdamW with a learning rate schedule, but it can also be seen with the Lars optimizer. With the Adam optimizer, the degenerative effect is not existing for those cases. However, the overall performance after 50 epochs with small batch sizes is only 49.48 % and thereby significantly lower than the performances with larger batch sizes and Adam at 52.22 % or even with larger batch sizes and AdamW at 56.82 %.

To further elaborate on the differences, two models were trained using the Adam optimizer with a batch size of 64 and 512 using only 20000 cutouts for computational feasibility. After 50 epochs, the approach with the larger batch size was 2.72 % better; after 100 epochs 2.48 %; after 150 epochs 0.58 % and after 200 epochs, only 0.18 %. This shows that with sufficient epochs, a convergence happens. The accuracies are shown in figure 25 in comparison to a model trained with a batch size of 512 with the AdamW optimizer on the dataset. The convergence that was seen in the training with the Adam optimizer aligns with the findings of Chen et al. [Che+20a], who have also seen this alignment of different batch sizes with more epochs.

The idea of adjusting the temperature parameter, especially in the cases where small batch sizes were used, could not lead to any positive effect. Several parameters were

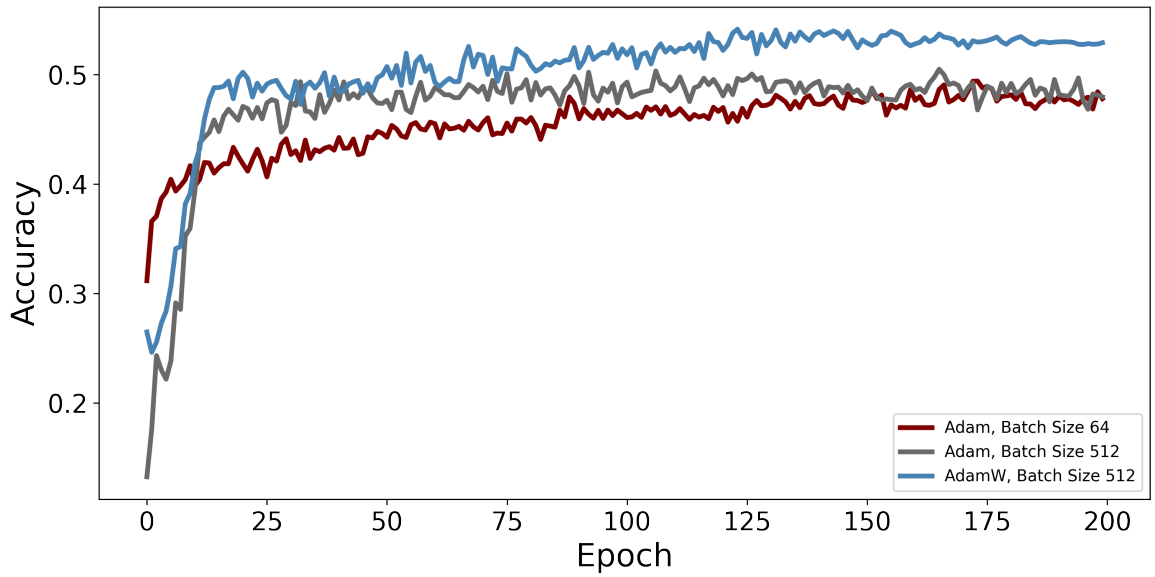


Figure 25 Comparison of Learning Convergence with Different Batch Sizes

tested like 0.01, 0.05, 0.1, 0.25, 0.5 and 1 with different batch sizes. All experiments lead to the suggested temperature parameter being 0.1 as suggested in Chen et al. [Che+20a].

All in all, models trained with a larger batch size have a higher accuracy and faster training. Additionally, larger batch sizes speed up the training significantly. Therefore, choosing them for all experiments is the best choice.

5.1.4 Dataset Preprocessing

The preprocessing of the dataset before the training was done to concentrate the training on interesting patches with objects. Table 11 in the appendix shows the result of the different accuracies per class and the total accuracy for every standard deviation.

Using the preprocessing, the accuracy can be raised from 53.18 % to 56.82 %. This is done without reducing the accuracy of easy classes. For example, the *agricultural_brown* class is reduced by 3 %, but *agricultural_green* could even be improved by 1.5 %. Additionally, more rare classes could be improved significantly as intended. For example, the accuracy of *windmills* was raised by 8 %, *river* by 8.5 % and *silo_small* by 9 %.

The additional removal of corrupted cutouts could only slightly improve the accuracy by 0.66 %, which is most likely reasoned by the limited number of cutouts that are corrupted. This hypothesis is supported by the fact that the improvement for the case

of no standard deviation filtering, where only 0.6 % of the cutouts were corrupted, was even less with only 0.22 %.

5.1.5 Evaluation of Evaluation Dataset

The evaluation setting for the methods was manually created. Therefore, its quality was also assessed. Seeing differences in the accuracies per class in different scenarios shows usefulness for evaluation. It was achieved to create a dataset consistent of cutouts of classes with different difficulties. To further elaborate on the dataset and the distribution and similarities of the classes, it was calculated which classes the results belonged to dependent on the class of the query. The exact numbers can be found in the appendix in figure 37 for the base model.

It can be seen that the classes *parking_lot_crooked* and *parking_lot_straight* are usually misclassified for each other, which is understandable because they only differ in the rotation of the cars to each other. A similar situation goes for the *round_water*, *silo_big* and *silo_small* classes since those objects mainly differ in size and color, but not in shape. A third such case is with agricultural_brown and *agricultural_green* and the same with *big_buildings* and *brown _buildings*. With those classes, it can be tested, whether the model can also identify minimal differences between classes. Nevertheless, it should be rechecked whether the classes are definitely separable or whether there is uncertainty.

5.1.6 Comparison of Results of Approaches

Using the base model parameters described in table 6, all models were trained and evaluated. The best-performing one was SimCLR achieving an accuracy of 56.82 %. In second place was the Transfer Learning approach with 50.6 % with a pretraining on ImageNet and finetuning with MLRSNET. Third was NNCLR with 47.42 %, and in last place was MAE[49] with only 32.08 %. Nevertheless, the baseline could still beat those basic model settings that were achieved after limited training time with an accuracy of 59.52 %.

In the figures 26, 27, and 28, different cutouts with different complexities can be seen. In the first column, the query images are displayed and the remaining columns show the second nearest neighbor predicted by each approach. The first cutout was not taken, because it is always the same as the query. The neighbors were calculated from 500000 unpreprocessed cutouts.

[49] Batch size of 256 to use Base Model of Dosovitskiy et al. [Dos+20]; experiments showed no batch size sensitivity

The figures verify the percentage finding of the accuracy of the testset. While for the easy cutouts, all methods could perform similarly well by finding very similar neighbors, especially the MAE had severe difficulties with the more challenging cutouts. The other approaches still performed similarly on the medium cutouts. However, for the most challenging cutouts, the baseline feature extraction and SimCLR are slightly ahead. Visually, SimCLR is even a bit better by better detecting a parking lot in the fourth sample of figure 28.

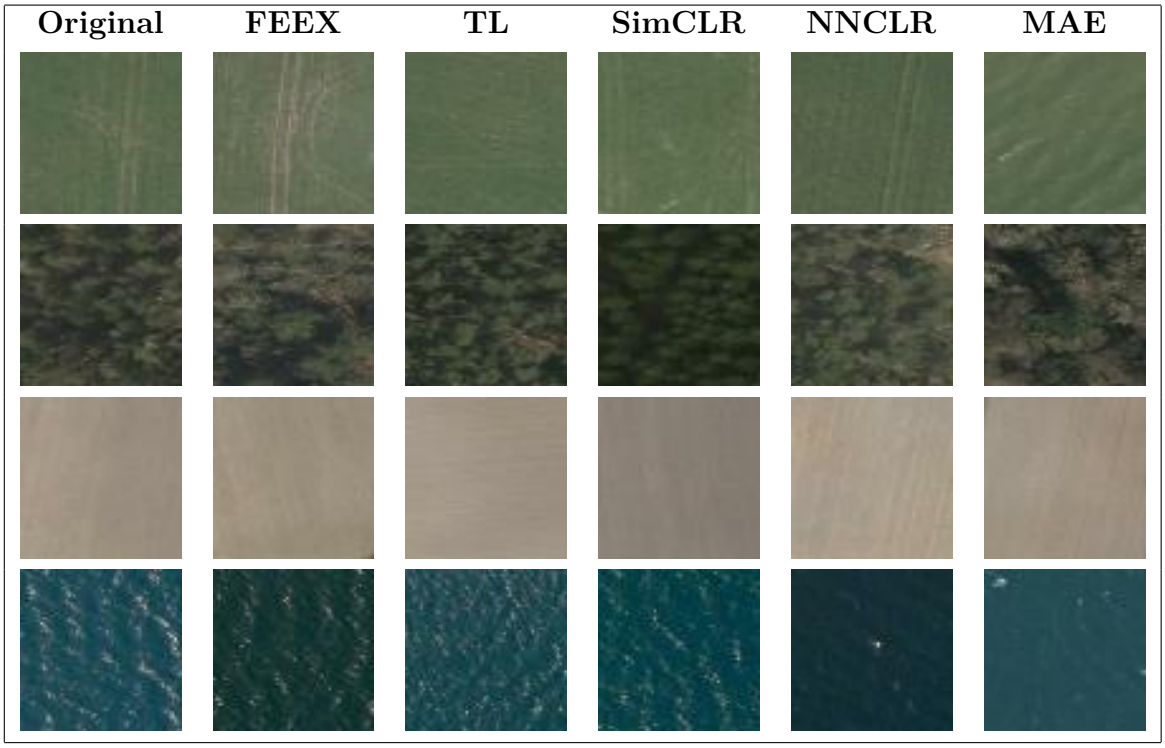


Figure 26 Neighbors of Easy Cutouts

Interesting is also the comparison of the different class accuracies of the approaches. It stands out that the transfer learning approach is the only one achieving very high accuracies in the simple agricultural classes, with an accuracy of 98.5 % in both of them. All other methods only achieved values up to 75 %. This is most likely due to the fact that all other methods use some type of (color-)augmentations that tend the network to ignore colors. Detailed accuracies for every class can be seen in table 14 in the appendix.

Also, an effect on what the methods were trained on can be seen. For remote sensing specific classes and shapes like *big_buildings*, *harbor* or *rail*, SimCLR beats the feature extraction by several percent. Nevertheless, for more general shapes like *coast*, *single_road*, or *tree line*, the feature extraction approach is superior. Showing that the feature extraction is good in detecting a straight line as in a street, but does not detect the differences of streets and rails appropriately. The only class that surpris-

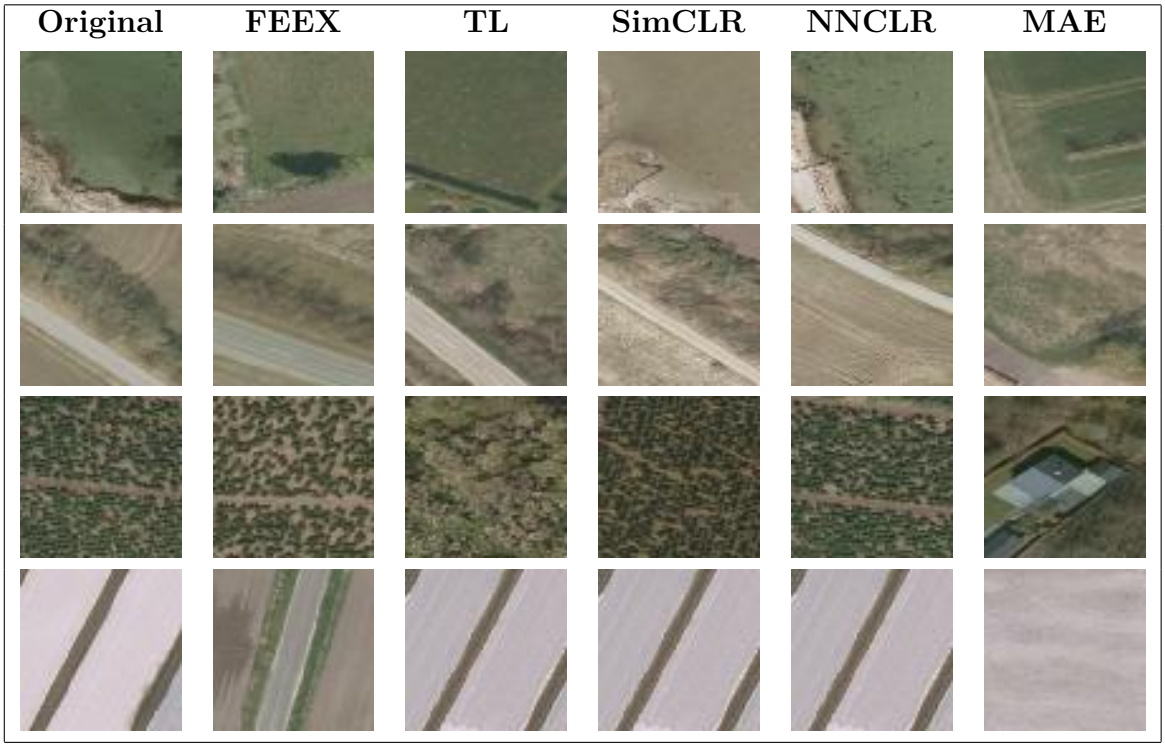


Figure 27 Neighbors of a bit Challenging Cutouts

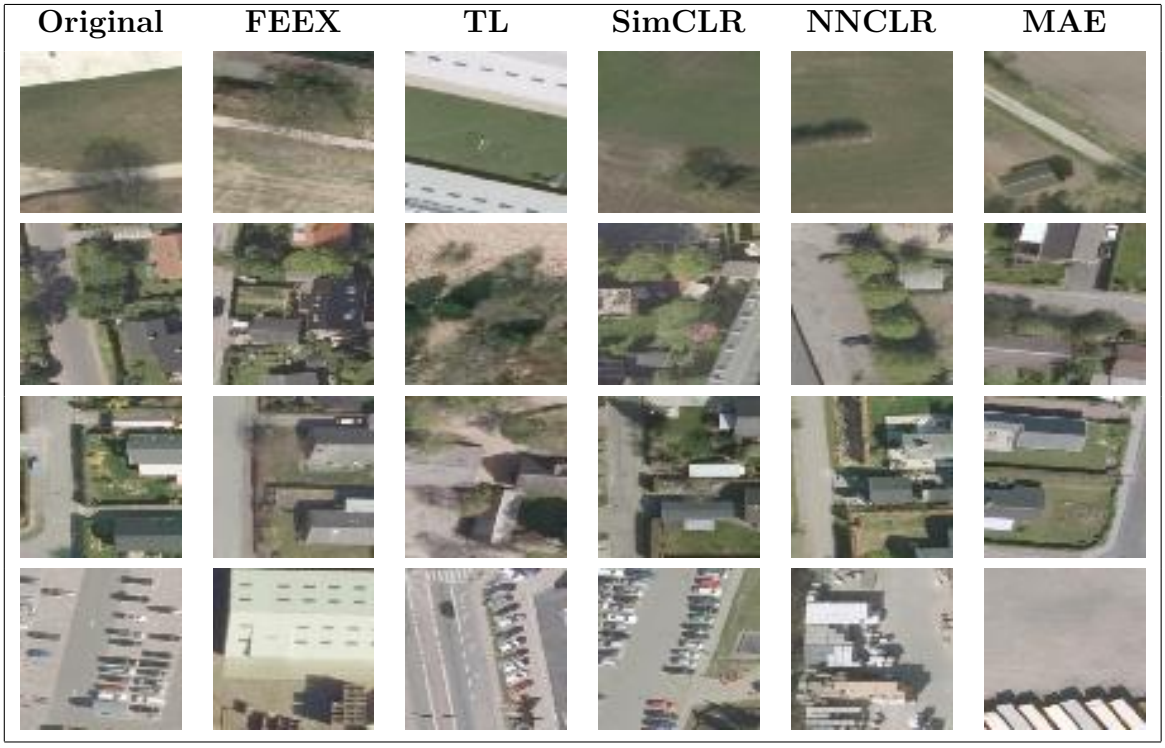


Figure 28 Neighbors of Challenging Cutouts

ingly stands out is the *windmill* class, where the feature extraction achieves 57 % and is better in this category by nearly 25 % over the second best approach.

5.1.7 Performance of FAISS

For the comparison of vectors, FAISS was chosen. It showed satisfactory performance for the task at hand. For 100000 datapoints, the search took 25.5 milliseconds, 261 milliseconds for 500000 datapoints and 857 milliseconds for 2000000 datapoints. In comparison, brute force search took 3.61 seconds for the neighbors of 100000 cutouts and 18.2 seconds for the neighbors of 500000 cutouts.

Those results make FAISS appropriate in terms of speed of comparisons and highly superior to brute force search. Also, when many more datapoints are used, the parameters of FAISS can be altered to allow a bit more approximate results increasing the speed even more.

The creation of the indices that enable such a fast search could be done in minimal time, needing 4.46 seconds for 100000 datapoints, 19.2 seconds for 500000 datapoints and 36.8 seconds for 2000000 datapoints.

Also, regarding the neighbor quality, FAISS could achieve the intended results. With the two testing approaches of taking n neighbors of the brute force search and the approximative search and comparing their overlap and the ratio of the total similarity, excellent results could be seen for the dataset of 500000 cutouts. The top 5 neighbors are identical, and if the number is raised to 100 it only gets slightly worse. Nevertheless, looking at the total ratio makes clear that those wrong-identified neighbors are not completely bad, but just very slightly worse than the ideal brute force ones.

	5	10	100	1000	10000	100000
Overlap	1.0	0.9	0.82	0.817	0.8016	0.9845
Ratio	1.0	1.027	1.067	1.092	1.148	2.145

Table 8 Comparison of FAISS Neighbor Quality With Brute Force Neighbors

So probably not the best neighbors could always be found if more than a few were searched, but excellent ones, not significantly decreasing the quality. This can be confirmed with the perceived quality, as can be seen in some examples in figure 29, where qualitative differences cannot be identified. The order of the results differs, but similar cutouts were returned as neighbors.

5.2 Interpretation of the Method Results

The performances of the methods presented beforehand do not align with the expected results in the literature since NNCLR could not outperform SimCLR. Additionally, MAE was indeed worse than both of them and not, as expected, the

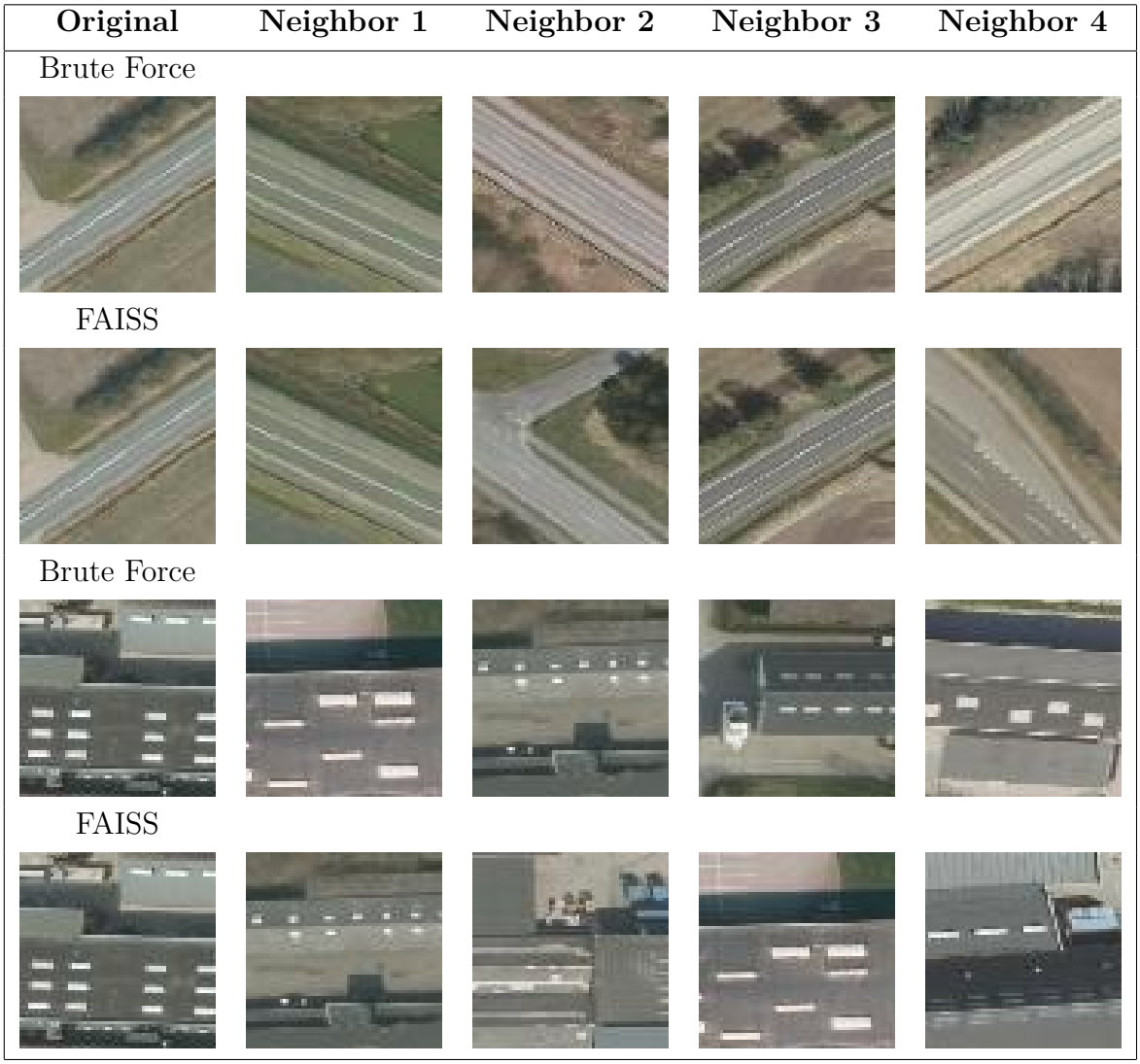


Figure 29 FAISS Reconstructions vs Brute Force Reconstructions

best-performing approach. In this section, reasons for the limited performance of the methods in the domain are discussed, and possible improvements are presented.

5.2.1 Feature Extraction

The feature extractor based on a ResNet50 trained with ImageNet already yielded promising results. Many other network architectures exist that are also available as Keras applications that could potentially improve that result. Nevertheless, none of the tested architectures could succeed using the downsampled testset. Therefore, the ResNet50 approach, with 59.52 %, performed best. For the cutouts not sampled down with a resolution of 256 by 256 pixels, the ResNet152 of the chosen architectures could work best and even beat the ResNet50 Feature Extraction by 3 %. The results of the downsampled values are shown in table 9.

Network	Accuracy
VGG16	57.98 %
VGG19	57.72 %
ResNet50	59.52 %
ResNet101	57.36 %
ResNet152	58.36 %
EfficientNetB0	52.2 %
Xception	-[50]
EfficientNetB7	35.24 %

Table 9 Accuracies of Feature Extraction from Different Network Architectures Pretrained on ImageNet

The improved performance on the full images can be likely reasoned in the fact that wider models can generalize better [BG18], and since the extracted vectors are used in a setup with completely different data, it helps to extract as generalized features as possible. The situation of generalized models performing better also applies to ResNets in general since they are excellent when it comes to generalization [He+15]. This might be the case why they perform better than the other architectures. Why the order of the best architecture differs in the downscaled tests has most likely something to do with the internal scaling since they are not trained for every shape.

5.2.2 Transfer Learning

The transfer learning attempt based on training a ResNet50 with remote sensing data did not achieve better results than comparing results trained on ImageNet despite being in the domain of remote sensing.

One possibility of this result might be that in contrast to the used MLRSNET dataset, ImageNet consists of many more classes and can thereby learn more generalized features than a dataset of just 46 classes. To test this hypothesis, two sets of 20000 images are extracted from the MLRSNET dataset. One with cutouts of all 46 classes and one with cutouts of only 10 of the classes. Afterwards, normal transfer learning is conducted, and the results are evaluated.

In comparison to the training with all the data, the sample with all classes with 20000 images could achieve an accuracy of 40.54 % which is already a significant decrease in comparison to the 47.26 % of the whole dataset without ImageNet pretraining. The decrease to only 10 used classes further reduced the performance to 37.16 %.

[50] The Xception Network could only be applied to cutouts of size 71 and above, therefore the value is missing.

This experiment supports the assumption, and it can be expected that datasets with more classes and more labeled cutouts could improve the results of this attempt and make it more generalizable. Those experiments were also conducted on the upscaled dataset with a size of 256 by 256 instead of 64 by 64 and there, the results were roughly 10 % better in every case. This shows that for the MLRSNET pretraining, higher resolutional cutouts are highly beneficial.

5.2.3 NNCLR

NNCLR was the second contrastive learning approach introduced in this work. Referring to Dwibedi et al. [Dwi+21], this approach should outperform SimCLR by several percent. But this behavior could not be reconstructed.

The authors describe, that the support set for the nearest neighbor search should be large enough to have at least one similar cutout in the support set and also that the accuracy would improve generally with an increased support set size. The remote sensing application is not limited to a definite amount of classes. Thereby, it might be necessary to have large support set sizes to find appropriate neighbors. This was tested, and the accuracy was calculated for a support set size of 512, 2048, 8192 and 32768. No significant improvement could be seen.

Seeing that NNCLR could not perform appropriately on the given dataset of Denmark, experiments were conducted on the MLRSNET dataset. For this, a testset similar to the one for Denmark was drawn from the data with 500 cutouts in 46 classes, and the NNCLR model was trained with 100.000 images of the MLRSNET dataset. With this, a performance of 44.76 % could be achieved which significantly could beat the baseline performance of a ResNet50 with feature extraction in the given test environment, which only achieved 33.54 %. Also, regarding the support set size, significant effects were seen. Having a support set size of 512 achieved only an accuracy of 39.62 %. This increased to 44.76 % with a support set size of 8192. Increasing the support set size further to 32768 decreased the accuracy back to 40.48 %.

The experiments with the MLRSNET dataset show that classes are necessary for the NNCLR approach. The algorithm chooses the nearest neighbor to train. It is intended that this neighbor is of the same class and thereby pulls the elements in this one class closer together. The data of Denmark is continuous in the sense that no classes exist, and therefore no distance between classes exists. For example, conjunctive cutouts consisting of different objects can exist. This could make the approach prone to pull just all the cutouts together and bringing cutouts close, which are not similar.

Exemplary, one could think of a cutout with a house and one with a parking lot. In general, those are unsimilar, but a cutout with a parking lot and a house in combination could exist. Possibly both the house cutout and the parking lot cutout are similar to the house with a parking lot cutout. Those three would then be drawn together by the algorithm despite not all being similar. In the actual application, this transition consists of many more intermediate cutouts, but the conveyed idea is the same.

The second finding was that the support set size should only be increased limited. This is most likely reasoned by the size of the dataset. With a support set size of 32768 and a dataset of 100.000, the dataset is only three times larger than the support set size. This reduces diversity because a sample can not be trained with a different neighbor in each epoch because most likely, the same couple of cutouts would be considered neighbors in every epoch. Thereby the networks cannot learn that the cutouts should be similar to a variety of other cutouts. This effect would be solved with a larger labeled dataset enabling large support set sizes.

5.2.4 Masked Autoencoder

The Masked Autoencoder performed worst among the tested methods. But this is most likely not reasoned in the method. As explained in the foundations, the Vision Transformers could use massive amounts of data and a pretraining of a model with a large dataset. Additionally, the models scale with more data and epochs. Most likely, such attempts would have been successful on the given dataset. Nevertheless, due to the specific configurations, a pre-trained model was not easy to find and due to computational resources training with large amounts of data was infeasible. This led to a model that could not compete with others. Even other positive effects on the accuracy, like larger models, more epochs or different patch sizes, could not improve the performance systematically and significantly.

To further elaborate on this, the MNIST[51] and MLRSNET datasets were used for additional experiments to at least show the general correctness of the implementation. With MLRSNET, the accuracy was not higher than in the Denmark case of roughly over 30 %. For MNIST, a neighbor accuracy could be reached of 69.42 % and a finetuning accuracy of 85.35 %. With this dataset, also advantages of larger models and longer training could be achieved. Nevertheless, those performances are still far behind state-of-the-art performances in literature.[52] At least the hypothesis that the batch size has no significant effect on the accuracy could be seen.

[51] https://www.tensorflow.org/datasets/catalog/mnist; a dataset of handwritten numbers usually used for a proof of concept

[52] https://paperswithcode.com/sota/image-classification-on-mnist

These problems are not unique. Difficulties with this approach in training appropriate models can also be seen in the basic configuration of the implementation basis[53]. There, Cifar-10[54] is used, and the approach achieves an accuracy of 47.9 %, also being not even close in accuracy to the current state-of-the-art models.[55] Additionally, the authors already stated in the paper that the training is not trivial and can be unstable [He+21]. The final insight is that, most likely, an extensive amount of research is necessary with many more experiments and larger models, more pretraining and large datasets to be able to achieve the promising results mentioned in the paper.

5.2.5 SimCLR

The SimCLR approach performed best among the tested methods and thereby proved its usability for remote sensing nearest neighbor calculation. In figure 30, a visualization of the feature vectors of the testset extracted with the model and reduced in dimensionality with UMAP[56] [MHM18] can be seen. The colors denote the classes. It can be seen that the algorithm groups elements of the same class closer together. This becomes particularly clear if the result is compared to other models, where the separation is less significant.

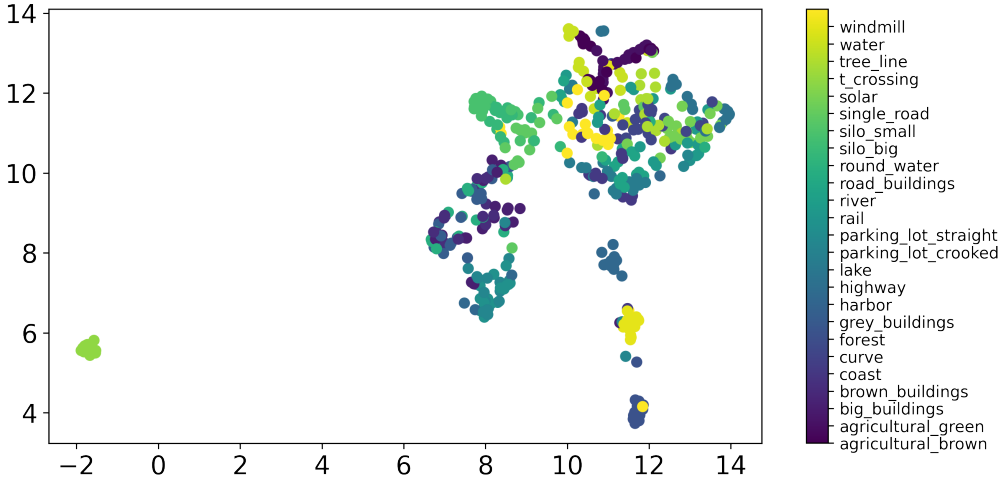


Figure 30 2-Dimensional Plot of Feature Vectors Reduced by UMAP

To achieve appropriate results, the most crucial aspect next to the general parameters is an appropriate augmentation process since the neural network is trained to represent cutouts similarly if they only differ by the different used augmentations. This

53 https://keras.io/examples/vision/masked_image_modeling/
54 https://www.cs.toronto.edu/ kriz/cifar.html
55 https://paperswithcode.com/sota/image-classification-on-cifar-10
56 A dimensionality reduction method that conserves the global structure

also leads to a problem, because thereby, some augmentations can also be malicious for the performance of the trained model for specific cutouts or in general.

One example is the color augmentation of cutouts. While the model only achieves an accuracy of 46.42 % overall on the testset without the color augmentations, some classes that are heavily reliant on the color of the cutout actually decrease in accuracy with this augmentation. This happens for *agricultural_brown* with a decrease of 13.5 % and *agricultural_green* with a decrease of 17 %. Nevertheless, other classes highly benefit from such an approach like *highway*, where the color augmentations increase the accuracy by 23 % or *river* where it is increased by 27.5 %. Herefore, this is one augmentation that benefits cutouts highly relying on the shape of objects but can decrease the quality of classes that should rely on the color.

With this in mind, the augmentations should be chosen in regard to the dataset and cannot be taken arbitrarily. For this work, the augmentations in the reference implementation relying on the results of the paper by Chen et al. [Che+20a] were used. Those are not necessarily the best for the remote sensing data at hand, but because of computational limitations, a grid search for different augmentations could not be conducted. This would be a recommendable step to enhance further the result with different augmentations or a change in their magnitude.

5.3 Final Model Scaling and Assessment

SimCLR is the best-performing method among the tested ones. To further improve the model, besides the already mentioned basic parameters, potential options derived from recommendations by Chen et al. [Che+20a] are addressed. After that, a final model is created based on all insights of this work and the performance evaluated.

5.3.1 Potential Improvement Options

To identify which improvements can benefit the final result, they are first analyzed separately. The experiments for this were mainly conducted with the Adam optimizer since degenerative behavior was witnessed in several settings with other optimizers.

More Epochs

The first tested suggestion is related to the duration of the training. The paper describes that the models improve significantly with longer training with more epochs. They benefit more from it than comparable supervised approaches. In the paper, the models are trained with several hundreds to thousands of epochs [Che+20a].

This improvement with more epochs could be reconstructed. If the training is done for 100 epochs instead of 50 epochs, the performance with the Adam optimizer rises from 52.22 % to 55.54 %. The training curve is shown in figure 31. Since no saturation can be seen, even more epochs seem helpful, as suggested in [Che+20a].

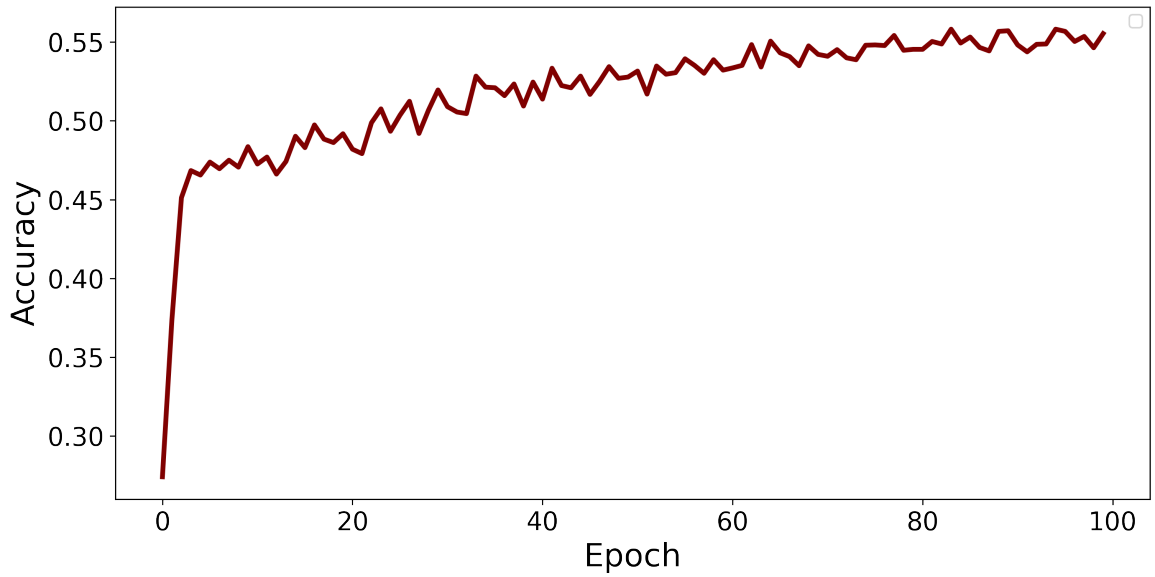


Figure 31 Accuracy per Epoch with Adam Optimizer

Larger Models

Another improvement was described by the authors for generally larger models. To try this out, the wider ResNet101 and ResNet152 architectures [He+15] were tested [Che+20a].

A significant improvement with the ResNet101 could not be identified. However, with the ResNet152, an improvement of 1.26 % could be realized.

More Data

The last regarded aspect mentioned by the authors is the use of larger training datasets [Che+20a]. In this context, a problem was detected. If larger datasets were used, the AdamW and Lars optimizer showed degenerative behavior. Therefore, this could only be tested with the Adam optimizer and the normal SGD optimizer. Of those two, the Adam optimizer performed better. Its accuracy improved by 4.18 % to 56.4 % if the amount of data was doubled from 100000 cutouts to 200000 cutouts.

Final Model

All of the experimental insights were combined in the final model. It consists of the SimCLR approach with a ResNet152 as Base Encoder. The dataset encompasses only uncorrupted cutouts with an average standard deviation of 30 and above. For the

training, 500000 cutouts were used. One epoch of this setting took twelve minutes for the 64 by 64 images with a batch size of 512. In comparison, the same model would have taken three hours for every epoch on cutouts sized 256 by 256 pixels and a batch size of 32 with the other parameters remaining equal.

This overall model was then trained for 400 epochs, which led to a final accuracy of 64.16 % after three days, beating the baseline of 59.52 % with a significant margin of roughly 5 %. The improvement can especially be made in more rare and challenging classes raising the accuracy for *windmills* by 17.5 %, for *big_silos* by 15 % and *coast* by 26.5 %. However, some classes also have a reduced accuracy in comparison to the smaller models. The detailed accuracies can be seen in the appendix in table 16.

In comparison with the other results, the accuracy improvement is very significant. For example, the model trained with the basic parameters and the Adam optimizer only achieved 52.78 %. It is also important to note that the extension of the model was highly limited by the availability of computational resources. With the number of epochs, a saturation in accuracy was reached. However, larger models and more data can certainly further improve the accuracy.

5.3.2 Evaluation of the Final Accuracy

The performance achieved by the final method is not as good as previously expected. With only around 64 %, it does not seem that the result can compete with approaches presented in literature or similar experiments with supervised learning conducted for this work. For example, a supervised training of the MLRSNET dataset with a ResNet50 architecture achieved a classification accuracy of over 90 % after 50 epochs on a testset created from MLRSNET. This seems to be much better even if the potential improvements of more computational resources, that are still available for SimCLR, are taken into account.

One reason could be that the custom dataset is much more challenging than the MLRSNET dataset, but this probably does not explain the size of the gap, since several classes are comparable with MLRSNET. However, a comparison of the nearest neighbor and classification problem can provide insight. In the given situation used for evaluating the performance of the methods, the accuracy is calculated in a scenario where 50 % of the available potential nearest neighbors are searched. Thinking of a two-step approach where first, every cutout would be classified, and second, the neighbor search would be conducted on the class labels, a neighbor would be correctly identified if two cutouts belong to the same class and if both cutouts are predicted in the same class. This means the squared classification accuracy can be expected

for this scenario to be the best value for comparison. In reverse, this means that the square root of the neighbor accuracy can be taken as an estimate for the performance in a classification setup. For the given neighbor accuracy of roughly 64 %, this would result in a comparable expected classification accuracy of 80 %. These assumptions hold according to experiments with a model trained on MLRSNET in a supervised way with a testset from MLRSNET. If, instead of the final predicted class, only the models without the head were used for comparison, the accuracy even decreased further below a neighbor accuracy of less than 60 %.

Additionally, the problem of finding just a few neighbors for a query from a set of not only a few hundred cutouts but hundreds of thousands of cutouts worked very well. The visual results showed outstanding performance as already described in subsection 5.1.6 for this few-nearest-neighbor problem.

The only thing that remains disappointing is the comparison with the baseline approach. The baseline could be beaten, but only by a few percent. This could have to do with the lack of sufficient training and the amount of data or simply a too challenging evaluation dataset where it is impossible to reach higher accuracies. Nevertheless, this is hard to test without a sufficient amount of data to approach the problem with supervised learning as a comparison. However, it shows the sophisticated generalizability of a ResNet trained on ImageNet as described in Huh, Agrawal, and Efros [HAE16].

6 Conclusion and Outlook

This work provided a thorough overview of advances in research and possible methods for a neighbor search in remote sensing images. For an exemplary unlabeled dataset of the country of Denmark, several methods showed empirical applicability. The ones of contrastive learning performed exceptionally well. However, this only applied to some, others were not compatible with the continuous dataset without distinct classes.

The model that performed best was SimCLR. After several finetuning steps, it could beat the defined baseline of a ResNet50 feature extraction with a margin of almost 5 %. This was achieved mainly by filtering the dataset for diverse patches, suitable general parameters and long training with large models and vast amounts of data.

During the experiments, it stood out that larger batch sizes had a great impact on the convergence of the model performance. In combination with the necessity for large models and long training, this shows that the methods are currently usable mainly if extensive computational resources are available. Therefore, this overview and comparison could present insights into different components of the models, but the meaningfulness regarding the maximum possible accuracy after model saturation could not be accomplished.

With the trained models, the search for limited numbers of nearest neighbors yielded a result that was perceived by a human as very good. The evaluation accuracy, on the contrary, was much worse. This demonstrates that achieving high accuracies in the all-nearest-neighbor task is much more challenging, even more challenging than in classification tasks.

Future research should be conducted to enable appropriate training even if the available computational resources are limited and large batch sizes are not feasible. The relevant aspects for this are alternative loss functions and more stable optimizers. Alternatively, it could also be possible to address the problems with a general change of the structure as described in Chen et al. [Che+22] and Yeh et al. [Yeh+21] or other contrastive approaches with particular regard to these two problems as described by He et al. [He+19]. Also, the approaches in self-supervised learning based on Vision Transformers like, for example, MAE [He+21], and DINO [Car+21], which do not have a problem with smaller batch sizes due to another loss function, have the potential to improve the performance of nearest neighbor search with additional extensive study. This will make it easier to find arbitrary objects all around the globe.

A Appendix

A.1 Additional Insights in the Dataset

class	std	class	std	class	std
agricultural_brown	8.30	lake	30.29	solar	43.99
agricultural_green	6.62	park._lot_crooked	40.74	t_crossing	30.41
big_buildings	36.44	park._lot_straight	39.42	tree_line	18.81
brown_buildings	38.96	rail	30.88	water	11.41
coast	34.17	river	26.86	windmill	28.01
curve	28.85	road_buildings	37.42		
forest	17.22	round_water	35.98		
grey_buildings	37.10	silo_big	41.03		
harbor	49.22	silo_small	43.53		
highway	26.91	single_road	25.66		

Table 10 Average Standard Distribution of the Classes

Class	**0-10**	**10-20**	**20-30**	**30-40**	**40-50**	**50-60**
agricultural_brown	0.65	0.665	0.62	0.62	0.54	0.6
agricultural_green	0.695	0.705	0.695	0.71	0.72	0.725
big_buildings	0.3	0.29	0.305	0.41	0.41	0.375
brown_buildings	0.325	0.305	0.335	0.465	0.420	0.405
coast	0.440	0.435	0.455	0.45	0.45	0.37
curve	0.405	0.4	0.375	0.385	0.405	0.41
forest	0.975	0.980	0.965	0.96	0.975	0.975
grey_buildings	0.37	0.355	0.37	0.45	0.41	0.39
harbor	0.690	0.715	0.755	0.7	0.68	0.685
highway	0.685	0.77	0.725	0.67	0.585	0.52
lake	0.46	0.41	0.425	0.465	0.455	0.36
parking_lot_crooked	0.275	0.3	0.34	0.31	0.3	0.325
parking_lot_straight	0.63	0.62	0.675	0.685	0.67	0.66
rail	0.345	0.375	0.37	0.39	0.31	0.235
river	0.48	0.490	0.535	0.565	0.495	0.455
road_buildings	0.29	0.305	0.295	0.3	0.325	0.3
round_water	0.455	0.490	0.55	0.61	0.685	0.595
silo_big	0.77	0.745	0.775	0.840	0.885	0.85
silo_small	0.395	0.375	0.415	0.485	0.5	0.58
single_road	0.440	0.38	0.415	0.41	0.4	0.420
solar	0.97	1.0	0.975	1.0	1.0	0.980
t_crossing	0.420	0.415	0.41	0.405	0.415	0.315
tree_line	0.64	0.645	0.645	0.63	0.570	0.490
water	0.940	0.92	0.965	0.96	0.96	0.955
windmill	0.245	0.23	0.27	0.325	0.275	0.285
Total	**0.5318**	**0.5327**	**0.5462**	**0.5682**	**0.5536**	**0.5302**

Table 11 Accuracy of each Class of Different Standard Deviations

A.2 Elaboration on Choice of Parameters

	Ep: 10	**Ep: 20**	**Ep: 30**	**Ep: 40**	**Ep: 50**
Batch Size: 32	43.8 %	39.98 %	41.52 %	40.70 %	30.02 %
Batch Size: 64	46.36 %	47.40 %	46.46 %	47.26 %	37.64 %
Batch Size: 128	45.98 %	47.88 %	49.56 %	47.50 %	40.88 %
Batch Size: 256	48.54 %	52.66 %	53.02 %	52.50 %	50.50 %
Batch Size: 512	51.14 %	51.58 %	54.98 %	55.58 %	56.82 %
Batch Size: 1024	52.7 %	53.52 %	54.92 %	56.64 %	56.96%

Table 12 Performance of Training with Different Batch Sizes of AdamW Optimizer

A.3 Additional Results of Model Comparison

Network	Accuracy
VGG16	0.5616
VGG19	0.5354
ResNet50	0.5876
ResNet101	0.6028
ResNet152	0.6142
EfficientNetB0	0.5974
Xception	0.3044
EfficientNetB7	0.5946

Table 13 Accuracy of Feature Extraction of Networks Without Downsampling

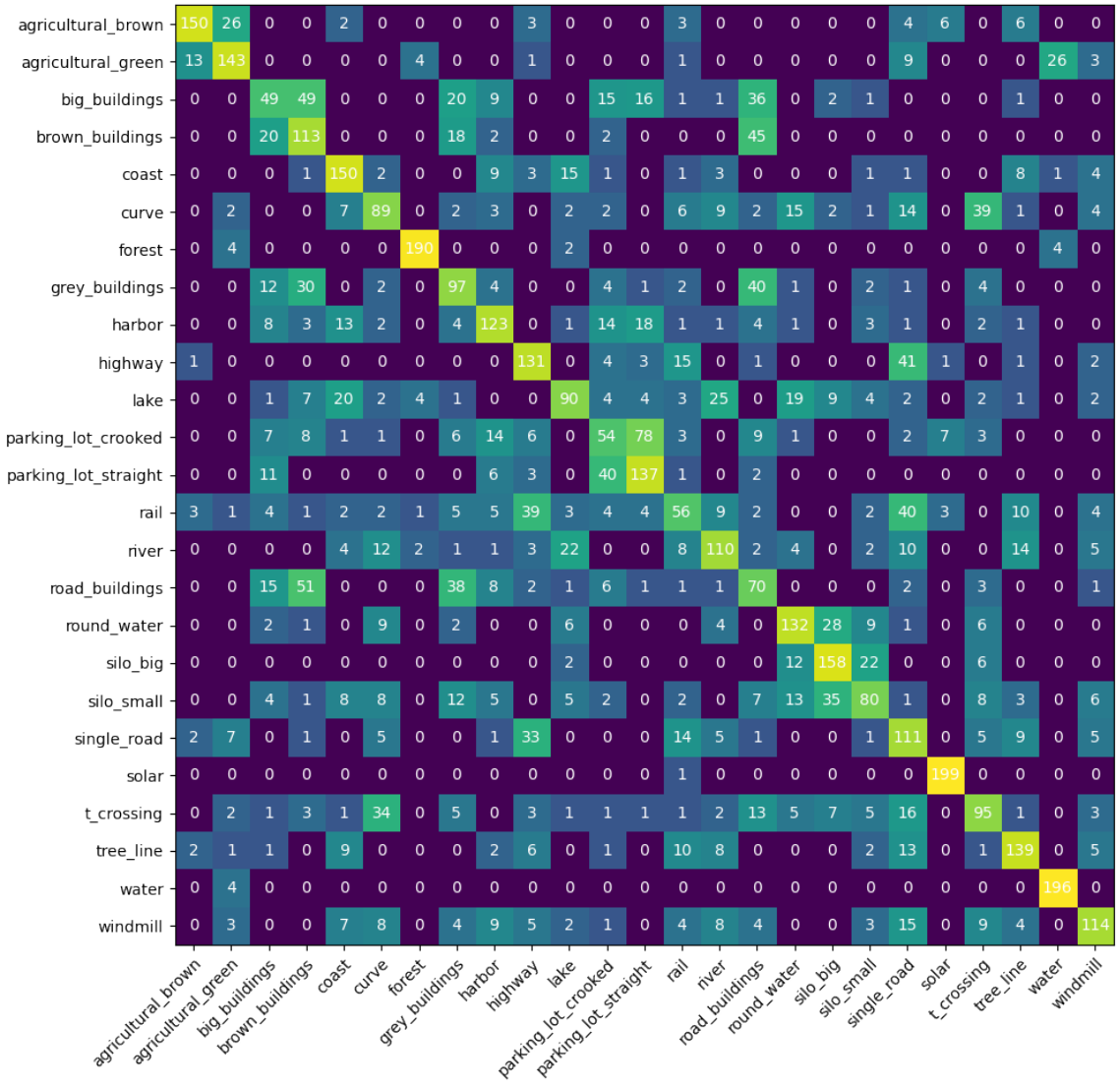


Figure 32 Classes of Cutouts Predicted as Neighbors Feature Extraction

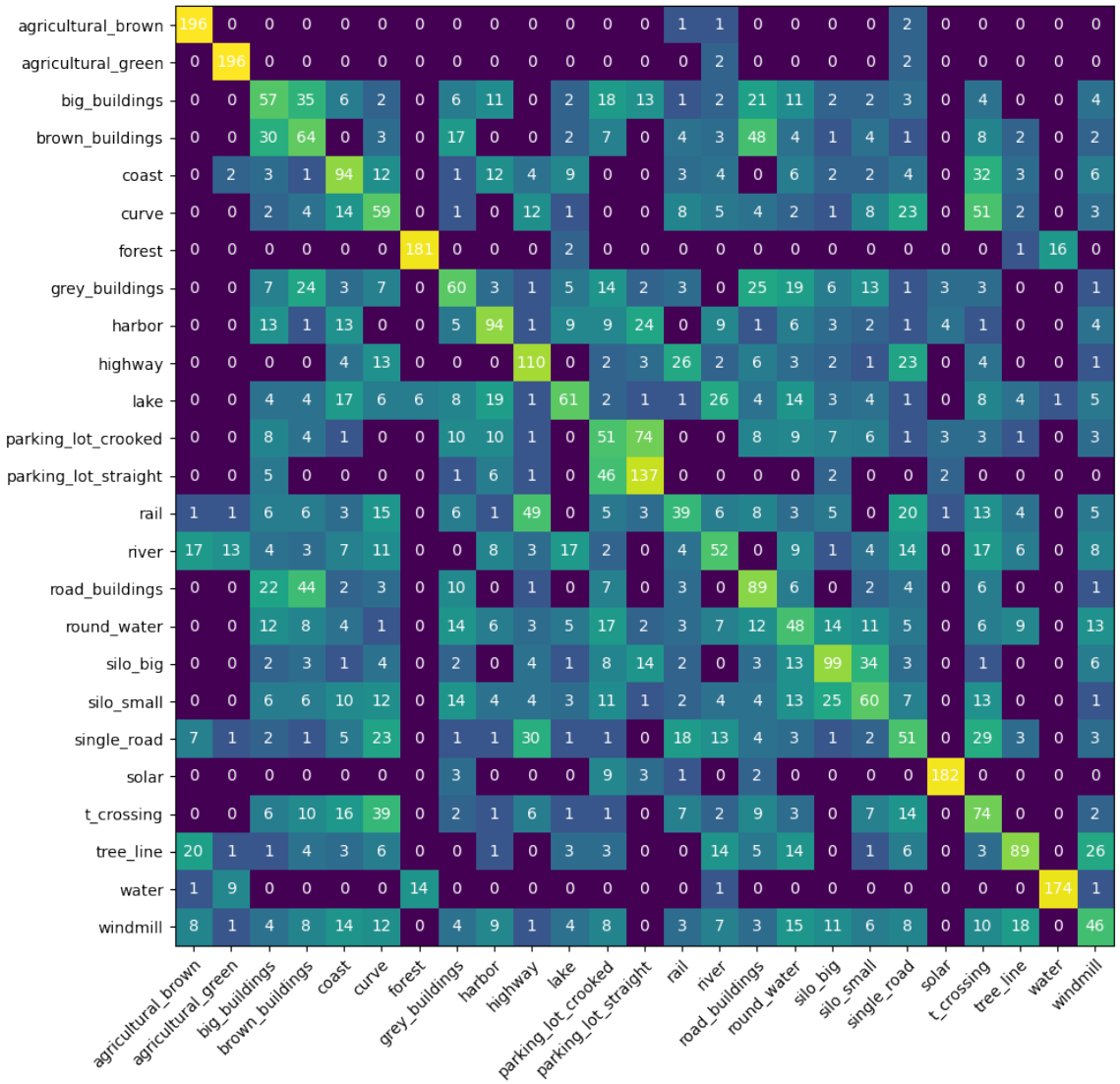


Figure 33 Classes of Cutouts Predicted as Neighbors Transfer Learning

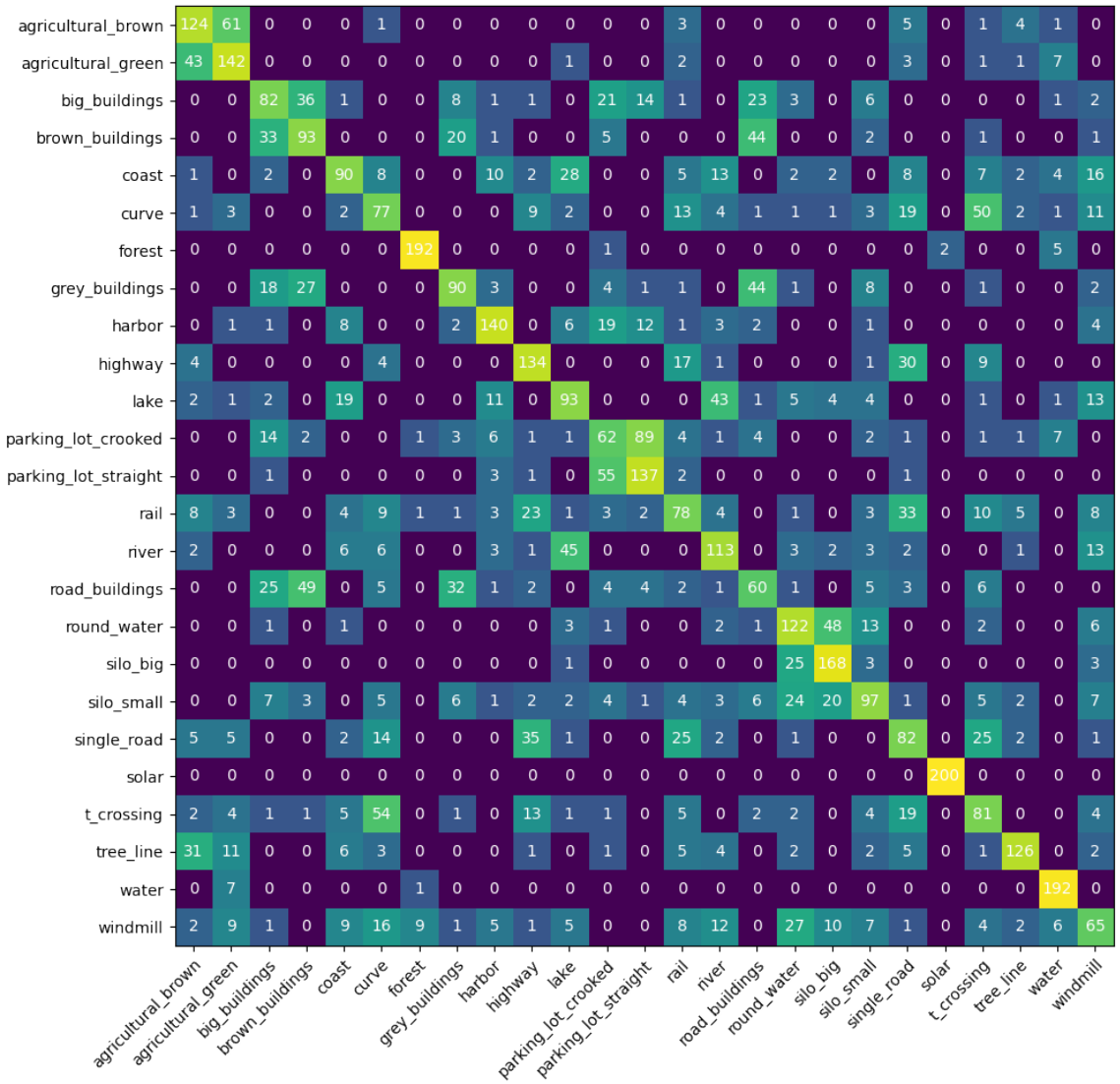


Figure 34 Classes of Cutouts Predicted as Neighbors SimCLR

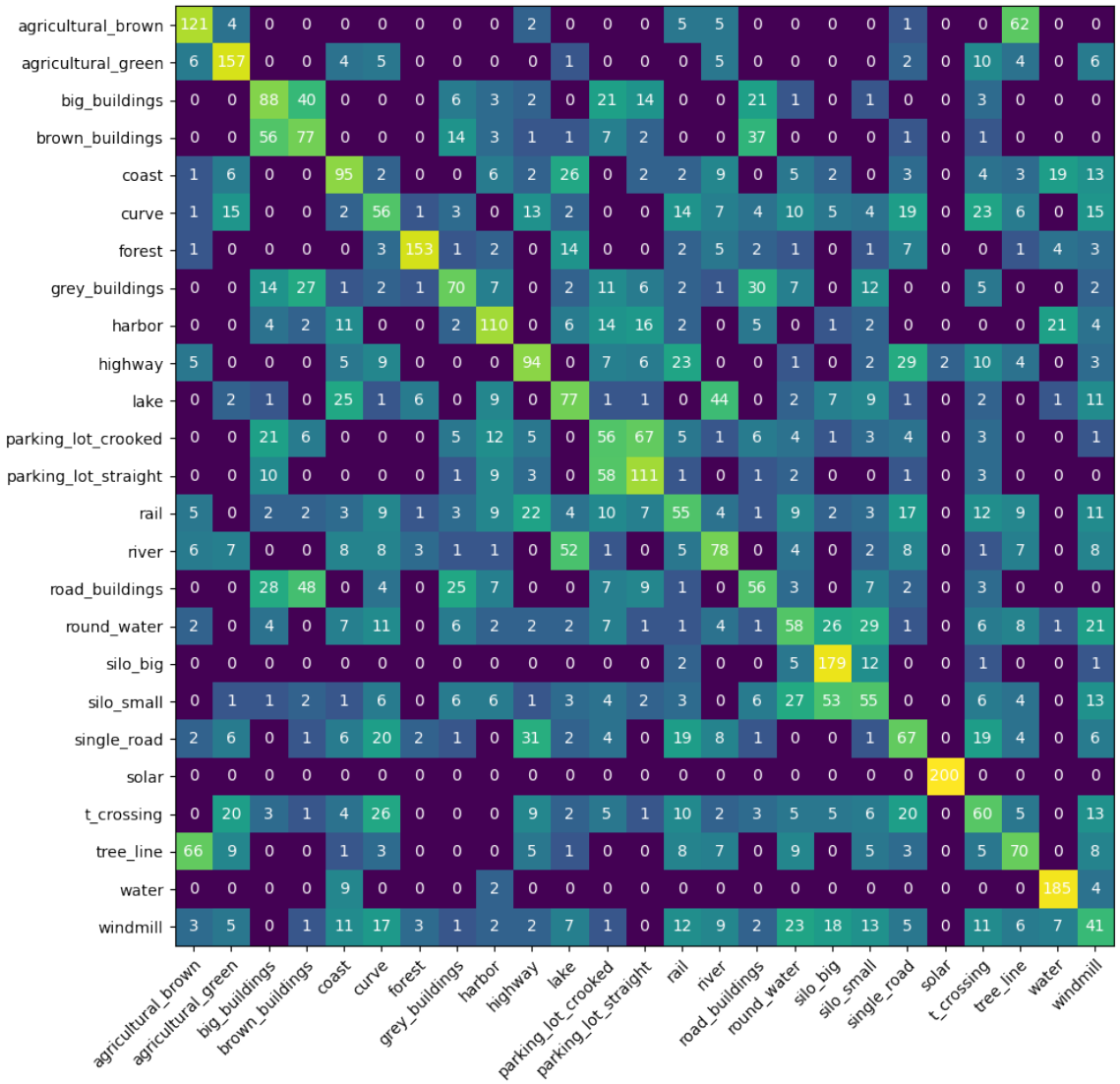


Figure 35 Classes of Cutouts Predicted as Neighbors NNCLR

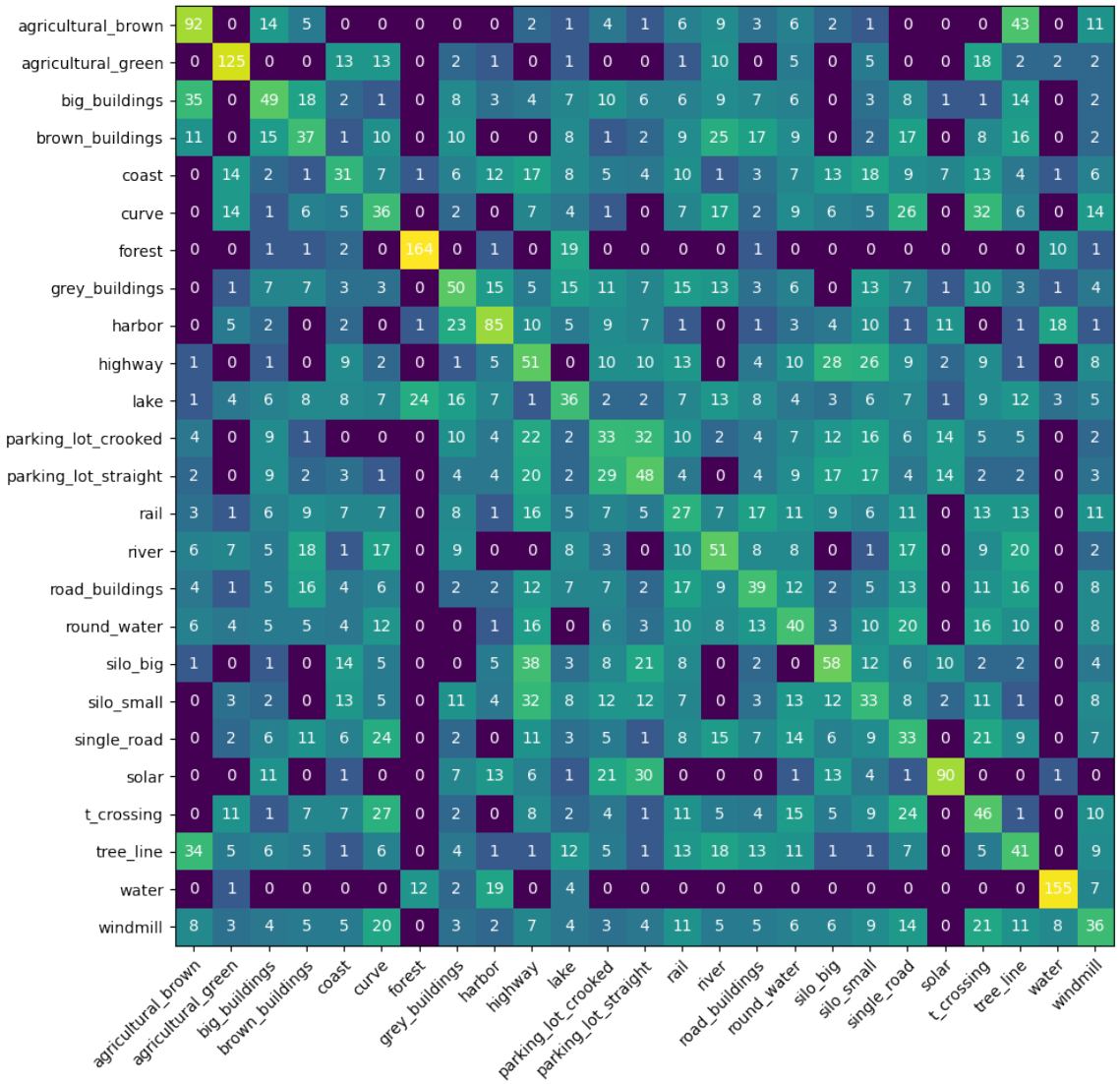


Figure 36 Classes of Cutouts Predicted as Neighbors MAE

Class	**FEEX**	**TL**	**SIMCLR**	**NNCLR**	**MAE**
agricultural_brown	0.75	0.985	0.62	0.62	0.46
agricultural_green	0.715	0.985	0.71	0.79	0.625
big_buildings	0.245	0.2855	0.41	0.435	0.2455
brown_buildings	0.565	0.32	0.465	0.385	0.185
coast	0.75	0.4753	0.45	0.475	0.155
curve	0.445	0.2955	0.385	0.26	0.18
forest	0.95	0.905	0.96	0.755	0.82
grey_buildings	0.485	0.3	0.45	0.35	0.25
harbor	0.615	0.4753	0.7	0.545	0.425
highway	0.655	0.55	0.67	0.475	0.255
lake	0.45	0.305	0.465	0.355	0.18
parking_lot_crooked	0.27	0.255	0.31	0.255	0.165
parking_lot_straight	0.685	0.685	0.685	0.555	0.24
rail	0.28	0.195	0.39	0.28	0.135
river	0.55	0.26	0.5655	0.4254	0.255
road_buildings	0.35	0.445	0.3	0.3	0.195
round_water	0.66	0.24	0.61	0.28	0.2
silo_big	0.79	0.495	0.845	0.905	0.29
silo_small	0.4	0.3	0.485	0.32	0.165
single_road	0.555	0.255	0.41	0.355	0.165
solar	0.995	0.91	1.0	1.0	0.45
t_crossing	0.475	0.37	0.405	0.305	0.23
tree_line	0.695	0.445	0.63	0.365	0.205
water	0.98	0.87	0.96	0.935	0.775
windmill	0.57	0.23	0.325	0.2152	0.18
Total	**0.5952**	**0.506**	**0.5682**	**0.4742**	**0.3208**

Table 14 Accuracy for each Class of the Models

Class	**Col. Aug.**	**No Col Aug.**
agricultural_brown	0.62	0.755
agricultural_green	0.71	0.88
big_buildings	0.41	0.39
brown_buildings	0.465	0.34
coast	0.45	0.285
curve	0.385	0.235
forest	0.96	0.87
grey_buildings	0.45	0.355
harbor	0.7	0.615
highway	0.67	0.44
lake	0.465	0.285
parking_lot_crooked	0.31	0.29
parking_lot_straight	0.685	0.53
rail	0.39	0.275
river	0.565	0.29
road_buildings	0.3	0.29
round_water	0.61	0.275
silo_big	0.84	0.755
silo_small	0.485	0.345
single_road	0.41	0.365
solar	1.0	0.935
t_crossing	0.405	0.27
tree_line	0.63	0.425
water	0.96	0.885
windmill	0.325	0.225
Total	**0.5682**	**0.4642**

Table 15 Accuracy of Each Class of Base Model and Approach Without Color Augmentation

A.4 Final Model Results

Class	Base Model	Final Model
agricultural_brown	0.62	0.82
agricultural_green	0.71	86
big_buildings	0.41	0.46
brown_buildings	0.465	0.345
coast	0.45	0.715
curve	0.385	0.33
forest	0.96	0.825
grey_buildings	0.45	0.44
harbor	0.7	0.755
highway	0.67	0.77
lake	0.465	0.425
parking_lot_crooked	0.31	0.365
parking_lot_straight	0.685	0.675
rail	0.39	0.45
river	0.565	0.59
road_buildings	0.3	0.33
round_water	0.61	0.87
silo_big	0.84	0.99
silo_small	0.485	0.54
single_road	0.41	0.48
solar	1.0	0.895
t_crossing	0.405	0.425
tree_line	0.63	0.52
water	0.96	0.905
windmill	0.325	0.5

Table 16 Accuracy of each Class of the Base Model and the Final Model

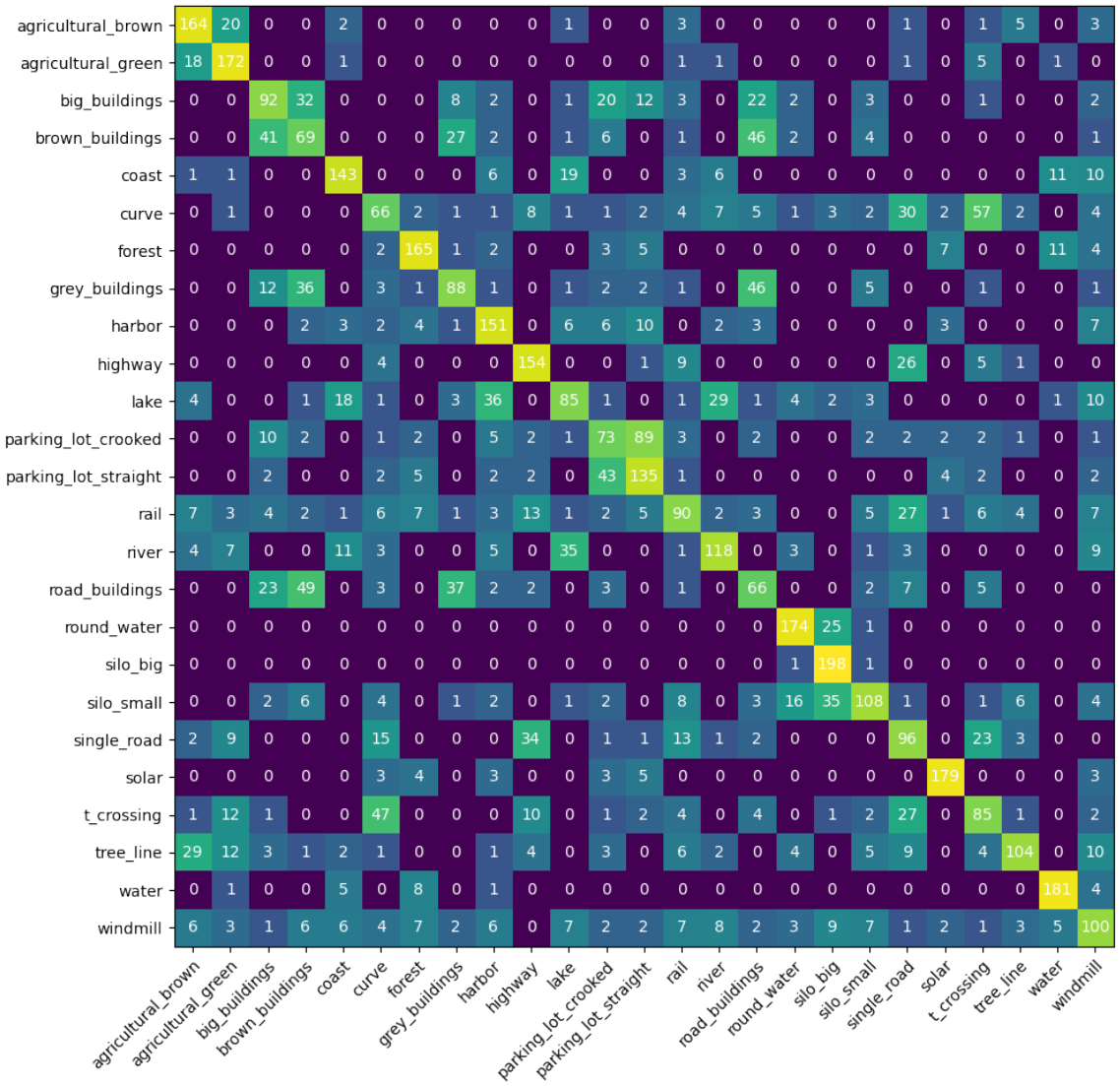


Figure 37 Classes of Cutouts Predicted as Neighbors Final Model

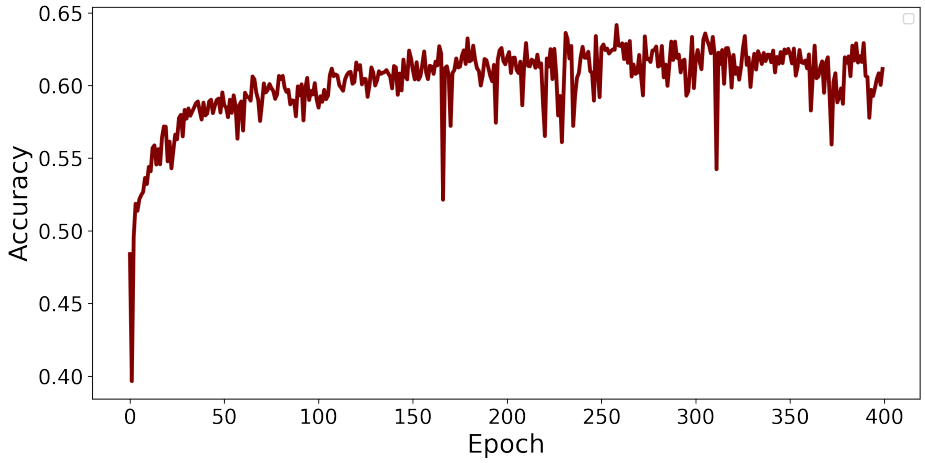


Figure 38 Accuracy of each Epoch of the Final Model

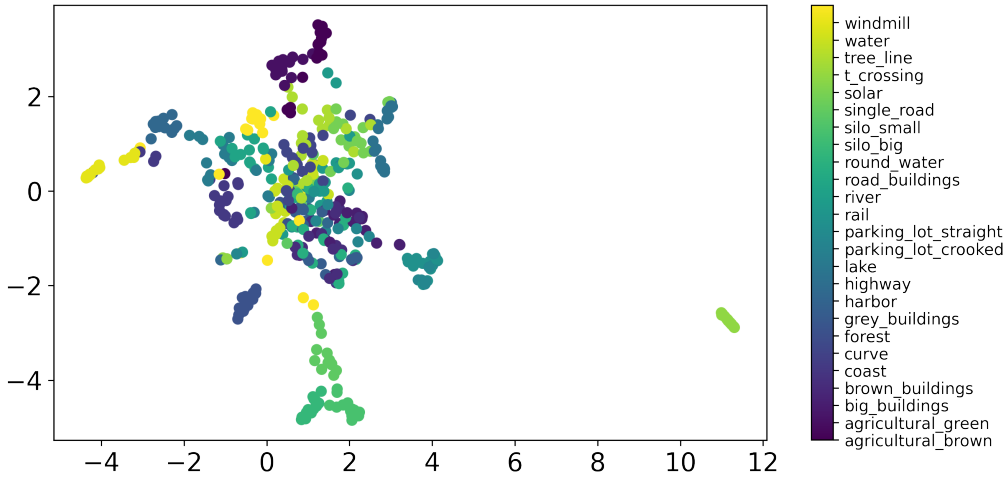


Figure 39 2-Dimensional Plot of Final Model

Figure 40 Neighbor Search of Final Model; First Element is Query Image

Bibliography

[AAK21] Sara Atito, Muhammad Awais, and Josef Kittler. "SiT: Self-supervised vIsion Transformer". In: (Apr. 2021). URL: http://arxiv.org/abs/2104.03602.

[Abd+20] Abolfazl Abdollahi et al. "Deep learning approaches applied to remote sensing datasets for road extraction: A state-of-the-art review". In: *Remote Sensing* 12 (9 2020). ISSN: 20724292. DOI: 10.3390/RS12091444.

[AC15] Jinwon An and Sungzoon Cho. "Variational Autoencoder based Anomaly Detection". In: *Special Lecture on IE* 2 (2015). ISSN: 15739031. DOI: 10.1007/BF00758335.

[AG21] Eman A. Alshari and Bharti W. Gawali. "Development of classification system for LULC using remote sensing and GIS". In: *Global Transitions Proceedings* 2 (1 2021), pp. 8–17. ISSN: 2666285X. DOI: 10.1016/j.gltp.2021.01.002. URL: https://doi.org/10.1016/j.gltp.2021.01.002.

[AK22] Abebaw Alem and Shailender Kumar. "Transfer Learning Models for Land Cover and Land Use Classification in Remote Sensing Image". In: *Applied Artificial Intelligence* 36 (1 2022). ISSN: 10876545. DOI: 10.1080/08839514.2021.2014192. URL: https://doi.org/10.1080/08839514.2021.2014192.

[Ara+18] Flavio H.D. Araujo et al. "Reverse image search for scientific data within and beyond the visible spectrum". In: *Expert Systems with Applications* 109 (2018). Hashing, Paper für Sven, pp. 35–48. ISSN: 09574174. DOI: 10.1016/j.eswa.2018.05.015. URL: https://doi.org/10.1016/j.eswa.2018.05.015.

[Ayo10] Taiwo Oladipupo Ayodele. *Machine Learning Overview*. 2010, p. 19. URL: www.intechopen.com.

[Ben75] Jon Louis Bentley. "bentley$_K$*Dtree.pdf*". In: *Communications of the ACM* (1975). ISSN: 00010782.

[BG18] Alon Brutzkus and Amir Globerson. "Why do Larger Models Generalize Better? A Theoretical Perspective via the XOR Problem". In: (Oct. 2018). URL: http://arxiv.org/abs/1810.03037.

[BKG20] Dor Bank, Noam Koenigstein, and Raja Giryes. "Autoencoders". In: (Mar. 2020). URL: http://arxiv.org/abs/2003.05991.

[Bra+20] Martin Brandt et al. “An unexpectedly large count of trees in the West African Sahara and Sahel”. In: *Nature* 587 (7832 2020), pp. 78–82. ISSN: 14764687. DOI: 10.1038/s41586-020-2824-5. URL: http://dx.doi.org/10.1038/s41586-020-2824-5.

[Car+18] Mathilde Caron et al. *Deep Clustering for Unsupervised Learning of Visual Features*. 2018.

[Car+20] Mathilde Caron et al. “Unsupervised Learning of Visual Features by Contrasting Cluster Assignments”. In: (June 2020). URL: http://arxiv.org/abs/2006.09882.

[Car+21] Mathilde Caron et al. “Emerging Properties in Self-Supervised Vision Transformers”. In: (Apr. 2021). URL: http://arxiv.org/abs/2104.14294.

[CH16] Gong Cheng and Junwei Han. *A Survey on Object Detection in Optical Remote Sensing Images*. 2016.

[Che+19] Xi Chen et al. “Variational lossy autoencoder”. In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings* (2019), pp. 1–17.

[Che+20a] Ting Chen et al. “A simple framework for contrastive learning of visual representations”. In: *37th International Conference on Machine Learning, ICML 2020* PartF16814 (Figure 1 2020), pp. 1575–1585.

[Che+20b] Ting Chen et al. “Big Self-Supervised Models are Strong Semi-Supervised Learners”. In: (June 2020). URL: http://arxiv.org/abs/2006.10029.

[Che+20c] Gong Cheng et al. “Remote Sensing Image Scene Classification Meets Deep Learning: Challenges, Methods, Benchmarks, and Opportunities”. In: (May 2020). DOI: 10.1109/JSTARS.2020.3005403. URL: http://arxiv.org/abs/2005.01094%20http://dx.doi.org/10.1109/JSTARS.2020.3005403.

[Che+22] Changyou Chen et al. *Why do We Need Large Batchsizes in Contrastive Learning? A Gradient-Bias Perspective*. 2022.

[Che21] Ta-Ying Cheng. *Supervised, Semi-Supervised, Unsupervised, and Self-Supervised Learning*. 2021. URL: https://towardsdatascience.com/supervised-semi-supervised-unsupervised-and-self-supervised-learning-7fa79aa9247c.

[CJK04] Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Ko. *Special Issue on Learning from Imbalanced Data Sets*. 2004. URL: http://purl.org/peter.turney/bibliographies/cost-.

[Cla83] Kenneth L. Clarkson. “Fast Algorithms for the All Nearest Neighbors Problem.” In: *Annual Symposium on Foundations of Computer Science (Proceedings)* (1983), pp. 226–232. ISSN: 02725428. DOI: 10.1109/sfcs.1983.16.

[CLW21] Yinbo Chen, Sifei Liu, and Xiaolong Wang. “Learning Continuous Image Representation with Local Implicit Image Function”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2021), pp. 8624–8634. ISSN: 10636919. DOI: 10.1109/CVPR46437.2021.00852.

[Dat+08] Ritendra Datta et al. “Image Retrieval: Ideas, In uences, and Trends of the New Age”. In: (2008).

[Den+09] Jia Deng et al. *ImageNet: A Large-Scale Hierarchical Image Database*. 2009. URL: http://www.image-net.org..

[Dev+18] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: (Oct. 2018). URL: http://arxiv.org/abs/1810.04805.

[Dia+20] Foivos I. Diakogiannis et al. “ResUNet-a: A deep learning framework for semantic segmentation of remotely sensed data”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 162 (February 2020), pp. 94–114. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2020.01.013. URL: https://doi.org/10.1016/j.isprsjprs.2020.01.013.

[Dim+22] Ivica Dimitrovski et al. “Current Trends in Deep Learning for Earth Observation: An Open-source Benchmark Arena for Image Classification”. In: (2022). URL: http://arxiv.org/abs/2207.07189.

[Dos+20] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: (Oct. 2020). URL: http://arxiv.org/abs/2010.11929.

[Du+19] Peijun Du et al. “Feature and Model Level Fusion of Pretrained CNN for Remote Sensing Scene Classification”. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 12 (8 2019), pp. 2600–2611. ISSN: 21511535. DOI: 10.1109/JSTARS.2018.2878037.

[DV18] Vincent Dumoulin and Francesco Visin. “A guide to convolution arithmetic for deep learning”. In: (Mar. 2018). URL: http://arxiv.org/abs/1603.07285.

[Dwi+21] Debidatta Dwibedi et al. “With a Little Help from My Friends: Nearest-Neighbor Contrastive Learning of Visual Representations”. In: (Apr. 2021). URL: http://arxiv.org/abs/2104.14548.

[EH20] Jesper E. van Engelen and Holger H. Hoos. “A survey on semi-supervised learning”. In: *Machine Learning* 109 (2 Feb. 2020), pp. 373–440. ISSN: 15730565. DOI: 10.1007/s10994-019-05855-6.

[Fer+20] Ruben Fernandez-beltran et al. “Using Probabilistic Latent Semantic Hashing”. In: 18 (2 2020), pp. 1–5.

[Fu+20] Kun Fu et al. “Rotation-aware and multi-scale convolutional neural network for object detection in remote sensing images”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 161 (May 2019 2020). good optional extension, pp. 294–308. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2020.01.025. URL: https://doi.org/10.1016/j.isprsjprs.2020.01.025.

[Gan+20] Wouter Van Gansbeke et al. “SCAN: Learning to Classify Images without Labels”. In: (May 2020). URL: http://arxiv.org/abs/2005.12320.

[Gér19] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn , Keras TensorFlow*. 2019. ISBN: 9781492032649.

[Gri+20] Jean-Bastien Grill et al. “Bootstrap your own latent: A new approach to self-supervised Learning”. In: (June 2020). URL: http://arxiv.org/abs/2006.07733.

[HAE16] Minyoung Huh, Pulkit Agrawal, and Alexei A. Efros. “What makes ImageNet good for transfer learning?” In: (Aug. 2016). URL: http://arxiv.org/abs/1608.08614.

[He+15] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: (Dec. 2015). URL: http://arxiv.org/abs/1512.03385.

[He+16] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2016-Decem (2016), pp. 770–778. ISSN: 10636919. DOI: 10.1109/CVPR.2016.90.

[He+19] Kaiming He et al. “Momentum Contrast for Unsupervised Visual Representation Learning”. In: (Nov. 2019). URL: http://arxiv.org/abs/1911.05722.

[He+21] Kaiming He et al. “Masked Autoencoders Are Scalable Vision Learners”. In: (Nov. 2021). URL: http://arxiv.org/abs/2111.06377.

[HKP12] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining. Concepts and Techniques, 3rd Edition (The Morgan Kaufmann Series in Data Management Systems)*. 2012.

[JDJ17] Jeff Johnson, Matthijs Douze, and Hervé Jégou. “Billion-scale similarity search with GPUs”. In: (Feb. 2017). URL: http://arxiv.org/abs/1702.08734.

[JT19] Longlong Jing and Yingli Tian. “Self-supervised Visual Feature Learning with Deep Neural Networks: A Survey”. In: (Feb. 2019). URL: http://arxiv.org/abs/1902.06162.

[JV96] Anil K Jain and Aditya Vailaya. *IMAGE RETRIEVAL USING COLOR AND SHAPE*. 1996, pp. 1233–1244.

[KB14] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: (Dec. 2014). URL: http://arxiv.org/abs/1412.6980.

[Kei+19] Ryan Keisler et al. “Visual search over billions of aerial and satellite images”. In: *Computer Vision and Image Understanding* 187 (2019). Transfer Learning, pp. 1–7. ISSN: 1090235X. DOI: 10.1016/j.cviu.2019.07.010.

[KG85] James M. Keller and Michael R. Gray. “A Fuzzy K-Nearest Neighbor Algorithm”. In: *IEEE Transactions on Systems, Man and Cybernetics* SMC-15 (4 1985), pp. 580–585. ISSN: 21682909. DOI: 10.1109/TSMC.1985.6313426.

[Kha+22] Salman Khan et al. “Transformers in Vision: A Survey”. In: (Jan. 2022). DOI: 10.1145/3505244. URL: http://arxiv.org/abs/2101.01169%20http://dx.doi.org/10.1145/3505244.

[Koc15] Gregory Koch. *Siamese Neural Networks for One-Shot Image Recognition*. 2015.

[KSB19] Simon Kornblith, Jonathon Shlens, and Quoc V Le Google Brain. *Do Better ImageNet Models Transfer Better?* 2019.

[LH17] Ilya Loshchilov and Frank Hutter. “Decoupled Weight Decay Regularization”. In: (Nov. 2017). URL: http://arxiv.org/abs/1711.05101.

[Li+16] Wen Li et al. “Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, and Improvement (v1.0)”. In: (Oct. 2016). URL: http://arxiv.org/abs/1610.02455.

[Li+20] Ke Li et al. “Object detection in optical remote sensing images: A survey and a new benchmark”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 159 (May 2019 2020), pp. 296–307. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2019.11.023.

[Lin+16] Tsung-Yi Lin et al. “Feature Pyramid Networks for Object Detection”. In: (Dec. 2016). URL: http://arxiv.org/abs/1612.03144.

[Liu+20] Yu Liu et al. “Anomaly detection based on machine learning in IoT-based vertical plant wall for indoor climate control”. In: *Building and Environment* 183 (July 2020), p. 107212. ISSN: 03601323. DOI: 10.1016/j.buildenv.2020.107212. URL: https://doi.org/10.1016/j.buildenv.2020.107212.

[Liu+21a] Chao Liu et al. “Deep Hash Learning for Remote Sensing Image Retrieval”. In: *IEEE Transactions on Geoscience and Remote Sensing* 59 (4 2021), pp. 3420–3443. ISSN: 15580644. DOI: 10.1109/TGRS.2020.3007533.

[Liu+21b] Xiao Liu et al. “Self-supervised Learning: Generative or Contrastive”. In: *IEEE Transactions on Knowledge and Data Engineering* (Jan. 2021). ISSN: 15582191. DOI: 10.1109/TKDE.2021.3090866.

[LMZ21] Yansheng Li, Jiayi Ma, and Yongjun Zhang. “Image retrieval from remote sensing big data: A survey”. In: *Information Fusion* 67 (October 2020 2021), pp. 94–115. ISSN: 15662535. DOI: 10.1016/j.inffus.2020.10.008. URL: https://doi.org/10.1016/j.inffus.2020.10.008.

[LRU14] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets*. 2014, p. 495.

[Ma+19] Lei Ma et al. “Deep learning in remote sensing applications: A meta-analysis and review”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 152 (March 2019), pp. 166–177. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2019.04.015. URL: https://doi.org/10.1016/j.isprsjprs.2019.04.015.

[Mas+11] Jonathan Masci et al. “Stacked convolutional auto-encoders for hierarchical feature extraction”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6791 LNCS (PART 1 2011). Unsupervised Feature Extraction, pp. 52–59. ISSN: 03029743. DOI: 10.1007/978-3-642-21735-7_7.

[MHM18] Leland McInnes, John Healy, and James Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. Umap. 2018.

[Mik+13] Tomas Mikolov et al. *Distributed Representations ofWords and Phrases and their Compositionality*. 2013. DOI: 10.1162/jmlr.2003.3.4-5.951.

[ML14] Marius Muja and David G. Lowe. “Scalable nearest neighbor algorithms for high dimensional data”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36 (11 2014), pp. 2227–2240. ISSN: 01628828. DOI: 10.1109/TPAMI.2014.2321376.

[MM96] B S Manjunathi and W Y Ma. *Texture Features for Browsing and Retrieval of Image Data*. 1996.

[Ngu+20] Huong Thi Thanh Nguyen et al. “Land use/land cover mapping using multitemporal sentinel-2 imagery and four classification methods-A case study from Dak Nong, Vietnam”. In: *Remote Sensing* 12 (9 May 2020). ISSN: 20724292. DOI: 10.3390/RS12091367.

[Noh+17] Hyeonwoo Noh et al. “Large-Scale Image Retrieval with Attentive Deep Local Features”. In: *Proceedings of the IEEE International Conference on Computer Vision* 2017-Octob (2017), pp. 3476–3485. ISSN: 15505499. DOI: 10.1109/ICCV.2017.374.

[Pat20] Kasha Patel. *How Cancún Grew into a Major Resort*. 2020. URL: https://earthobservatory.nasa.gov/images/146194/how-cancun-grew-into-a-major-resort.

[Pet+20] Biserka Petrovska et al. “Deep learning for feature extraction in remote sensing: A case-study of aerial scene classification”. In: *Sensors (Switzerland)* 20 (14 2020), pp. 1–22. ISSN: 14248220. DOI: 10.3390/s20143906.

[PNS21] Santisudha Panigrahi, Anuja Nanda, and Tripti Swarnkar. “A Survey on Transfer Learning”. In: *Smart Innovation, Systems and Technologies* 194 (10 2021). Transfer Learning, pp. 781–789. ISSN: 21903026. DOI: 10.1007/978-981-15-5971-6_83.

[Qi+20] Xiaoman Qi et al. *MLRSNet: A Multi-label High Spatial Resolution Remote Sensing Dataset for Semantic Scene Understanding*. 2020. URL: https://data.mendeley.com/datasets/7j9bv9vwsx/1.

[Rab+20] Jakaria Rabbi et al. “Small-object detection in remote sensing images with end-to-end edge-enhanced GAN and object detector network”. In: *Remote Sensing* 12 (9 2020), pp. 1–25. ISSN: 20724292. DOI: 10.3390/RS12091432.

[Rad19] Filip Radenovi. “Fine-Tuning CNN Image Retrieval with No Human Annotation”. In: 41 (7 2019). training data without manual work, pp. 1655–1668.

[Rag+21] Maithra Raghu et al. “Do Vision Transformers See Like Convolutional Neural Networks?” In: (Aug. 2021). URL: http://arxiv.org/abs/2108.08810.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “INet: Convolutional Networks for Biomedical Image Segmentation”. In: *IEEE Access* 9 (2015), pp. 16591–16603. ISSN: 21693536. DOI: 10.1109/ACCESS.2021.3053408.

[RM51] Herbert Robbins and Sutton Monro. “A STOCHASTIC APPROXIMATION METHOD”. In: *The Annals of Mathematical Statistics* (1951).

[SH07] Ruslan Salakhutdinov and Geoffrey Hinton. *Semantic Hashing*. 2007.

[Son+18] Jingkuan Song et al. *Binary Generative Adversarial Networks for Image Retrieval*. 2018. URL: www.aaai.org.

[SS00] Linda Shapiro and George Stockman. *Computer Vision*. Primary Book. 2000.

[Sun+21a] Xian Sun et al. “PBNet: Part-based convolutional neural network for complex composite object detection in remote sensing imagery”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 173 (December 2020 2021), pp. 50–65. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2020.12.015. URL: https://doi.org/10.1016/j.isprsjprs.2020.12.015.

[Sun+21b] Xian Sun et al. “Research Progress on Few-Shot Learning for Remote Sensing Image Interpretation”. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 14 (2021), pp. 2387–2402. ISSN: 21511535. DOI: 10.1109/JSTARS.2021.3052869.

[Tan+18] Chuanqi Tan et al. “A Survey on Deep Transfer Learning”. In: (Aug. 2018). URL: http://arxiv.org/abs/1808.01974.

[Tia+20] Yonglong Tian et al. “What Makes for Good Views for Contrastive Learning?” In: (May 2020). URL: http://arxiv.org/abs/2005.10243.

[Tsa+19] Grigorios Tsagkatakis et al. “Survey of deep-learning approaches for remote sensing observation enhancement”. In: *Sensors (Switzerland)* 19 (18 2019), pp. 1–39. ISSN: 14248220. DOI: 10.3390/s19183929.

[Vas+17] Ashish Vaswani et al. “Attention Is All You Need”. In: (June 2017). URL: http://arxiv.org/abs/1706.03762.

[Vou+18] Athanasios Voulodimos et al. *Deep Learning for Computer Vision: A Brief Review*. 2018. DOI: 10.1155/2018/7068349.

[Wan+14] Ji Wan et al. “Deep learning for content-based image retrieval: A comprehensive study”. In: Association for Computing Machinery, Nov. 2014, pp. 157–166. ISBN: 9781450330633. DOI: 10.1145/2647868.2654948.

[Wan+22] Di Wang et al. “An Empirical Study of Remote Sensing Pretraining”. In: *IEEE Transactions on Geoscience and Remote Sensing* 14 (8 2022). ISSN: 15580644. DOI: 10.1109/TGRS.2022.3176603.

[Xia+16] Gui-Song Xia et al. “AID: A Benchmark Dataset for Performance Evaluation of Aerial Scene Classification”. In: (Aug. 2016). DOI: 10.1109/TGRS.2017.2685945. URL: http://arxiv.org/abs/1608.05167%20http://dx.doi.org/10.1109/TGRS.2017.2685945.

[Xia+17] Gui-Song Xia et al. “DOTA: A Large-scale Dataset for Object Detection in Aerial Images”. In: (Nov. 2017). URL: http://arxiv.org/abs/1711.10398.

[YCZ20] Yanan You, Jingyi Cao, and Wenli Zhou. “A survey of change detection methods based on remote sensing images for multi-source and multi-objective scenarios”. In: *Remote Sensing* 12 (15 2020). ISSN: 20724292. DOI: 10.3390/RS12152460.

[Yeh+21] Chun-Hsiao Yeh et al. *Decoupled Contrastive Learning*. 2021.

[YGG17] Yang You, Igor Gitman, and Boris Ginsburg. “Large Batch Training of Convolutional Networks”. In: (Aug. 2017). URL: http://arxiv.org/abs/1708.03888.

[YSG21] Xiaohui Yuan, Jianfang Shi, and Lichuan Gu. “A review of deep learning methods for semantic segmentation of remote sensing imagery”. In: *Expert Systems with Applications* 169 (December 2020 2021), p. 114417. ISSN: 09574174. DOI: 10.1016/j.eswa.2020.114417. URL: https://doi.org/10.1016/j.eswa.2020.114417.

[Zha+20] Xin Zhang et al. “How well do deep learning-based methods for land cover classification and object detection perform on high resolution remote sensing imagery?” In: *Remote Sensing* 12 (3 2020), pp. 1–29. ISSN: 20724292. DOI: 10.3390/rs12030417.

Declaration of Authorship

I hereby declare that, to the best of my knowledge and belief, this thesis titled *Comparison of Remote Sensing Image Feature Extraction Methods for Nearest Neighbor Search* is my own, independent work. I confirm that each significant contribution to and quotation in this thesis that originates from the work or works of others is indicated by proper use of citation and references; this also holds for tables and graphical works.

Münster, 24.03.2023

Lennart Seeger

Consent Form

Name: Lennart Seeger
Title of Thesis: Comparison of Remote Sensing Image Feature Extraction Methods for Nearest Neighbor Search

What is plagiarism? Plagiarism is defined as submitting someone else's work or ideas as your own without a complete indication of the source. It is hereby irrelevant whether the work of others is copied word by word without acknowledgment of the source, text structures (e.g. line of argumentation or outline) are borrowed or texts are translated from a foreign language.

Use of plagiarism detection software. The examination office uses plagiarism software to check each submitted bachelor and master thesis for plagiarism. For that purpose the thesis is electronically forwarded to a software service provider where the software checks for potential matches between the submitted work and work from other sources. For future comparisons with other theses, your thesis will be permanently stored in a database. Only the School of Business and Economics of the University of Münster is allowed to access your stored thesis. The student agrees that his or her thesis may be stored and reproduced only for the purpose of plagiarism assessment. The first examiner of the thesis will be advised on the outcome of the plagiarism assessment.

Sanctions Each case of plagiarism constitutes an attempt to deceive in terms of the examination regulations and will lead to the thesis being graded as "failed". This will be communicated to the examination office where your case will be documented. In the event of a serious case of deception the examinee can be generally excluded from any further examination. This can lead to the exmatriculation of the student. Even after completion of the examination procedure and graduation from university, plagiarism can result in a withdrawal of the awarded academic degree.

I confirm that I have read and understood the information in this document. I agree to the outlined procedure for plagiarism assessment and potential sanctioning.

Münster, 24.03.2023

Lennart Seeger