Technische Universität Berlin

Faculty of Electrical Engineering and Computer Science
Dept. of Computer Engineering and Microelectronics
Remote Sensing Image Analysis Group

Image Search and Retrieval from Remote Sensing Archives based on Similarity Ranking Functions

Computer Science B.Sc.

October, 2020

Matthias Jost Wagner

Matriculation Number: 391932

Supervisor: Prof. Dr. Begüm Demir

Advisor: Tristan Kreuziger

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt habe. Sämtliche benutzten Informationsquellen sowie das Gedankengut Dritter wurden im Text als solche kenntlich gemacht und im Literaturverzeichnis angeführt. Die Arbeit wurde bisher nicht veröffentlicht und keiner Prüfungsbehörde vorgelegt.

Hereby I declare that I wrote this thesis myself with the help of no more than the mentioned literature and auxiliary means.

Berlin, 27.10.2020

Wagner ....

Matthias Jost Wagner

Acknowledgements

First of all, I would like to thank Prof. Dr. Demir for the fact that I was able to write my Bachelor's thesis in your research group Remote Sensing Image Analysis Group of the TU Berlin. Further thanks go to Tristan Kreuziger, who was always available for questions during the time writing my thesis and supported me in implementing the application of this thesis. Finally, I would like to thank my friends and family for their support during the time of my Bachelor thesis.

Abstract

The importance of Content Based Image Retrieval (CBIR) in Remote Sensing (RS) is steadily increasing with the growth of satellite image archives. Most images in Remote Sensing are labelled with several classes, in contrast to a large number of single-label images in the domain of Computer Vision (CV). In order to train deep learning models for image retrieval, error functions are used which influence the embedding function of the models during training. Error functions can be used independently of the domain in CV and RS. However, not all methods presented in publications are implemented for multi-label images. An absolute comparison of multi-label images is not always desired, so there are methods to check multi-label images for similarity and thus tolerate variations in class differences between images. Ranked List Loss (RLL) is an error function that aims to improve Triplet Loss, but has been published for single-label images. The topic of this thesis is to implement the RLL error function for multi-label images for CBIR in RS and to investigate the influence of parameters of the trained models on results by a sensitivity analysis. To distinguish multi-labels, the cosine similarity function is used, which represents the similarity of labels in an interval from 0 to 1. Labels are considered similar if the function value of the cosine similarity function exceeds a specified limit. The model trained on RLL for multi-label achieves 73% precision in image retrieval. A method of using RLL for multi-label images in RS Archives is thus introduced.

Zusammenfassung

Die Bedeutung von Content Based Image Retrieval (CBIR) in Remote Sensing (RS) steigt stetig mit der Zunahme von Satellitenbildern stetig an. Dabei sind die meisten Bilder in RS mit mehreren Klassen markiert, im Gegensatz zu einer großen Menge Single-label Bildern in der Domäne von Computer Vision (CV). Um Deep Learning Modelle auf Image Retrieval zu trainieren werden Fehlerfunktionen eingesetzt, die die Einbettungs-Funktion der Modelle beeinflussen. Fehlerfunktionen können unabhängig der Domäne in CV und RS eingesetzt werden. Es werden jedoch nicht alle Methoden in Veröffentlichungen für Multi-Label-Bilder vorgestellt. Ein absoluter Vergleich von Multi-Label-Bilder ist nicht immer gewollt, daher gibt es Methoden Multi-Label auf Ähnlichkeit zu überprüfen und so Abweichungen in Klassenunterschieden zwischen Bildern zu tolerieren. Ranked List Loss (RLL) ist eine Fehlerfunktion, die den Triplet Loss verbessern möchte, jedoch für Single-Label-Bilder veröffentlicht wurde. Thema dieser Abschlussarbeit ist es die RLL Fehlerfunktion für Multi-Label Bilder für CBIR in RS zu implementieren und den Einfluss von Parametern der trainierten Modelle auf Ergebnisse durch eine Empfindlichkeitsanalyse zu untersuchen. Zur Unterscheidung von Multi-Labels wird die Kosinus-Ähnlichkeitsfunktion eingesetzt, mit der die Ähnlichkeit von Labels in einem Intervall von 0 bis 1 wiedergegeben wird. Labels werden als ähnlich betrachtet, wenn der Funktionswert der Kosinus-Ähnlichkeitsfunktion einen festgelegten Grenzwert überschreitet. Das auf RLL für Multi-Label trainierte Modell erreicht in der Bildabfrage eine Genauigkeit von 73%. Eine Methode RLL für Multi-Label Bilder für RS Archive einzusetzen ist somit eingeführt.

Contents

List of Acronyms

RS	Remote Sensing
CBIR	Content Based Image Retrival
RLL	Ranked List Loss
LMNN	Large Margin Nearest Neighbor
NCA	Neighbourhood Components Analysis

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Remote Sensing (RS) research is busy adapting state-of-the art methods to its own domain and there is a lot of potential. Using different loss functions to improve results in the field of deep metric learning gained traction over the last years. Especially image retrieval and classification are always on the search for a loss function, that is able to create a mapping into feature space with more accuracy and diversity regarding input data. With loss functions being an essential part in deep metric learning, many different approaches have been proposed and implemented. Many of the used functions are based on triplet loss [11][23]. On the topic of deep metric learning Wang et al. proposed a loss function called 'Ranked List Loss' which aims to overcome two limitations of existing ranking-motivated structured losses citeWang.08.03.2019. Usually, most data points are not used to build the similarity structure. Missing out on non-trivial data points is making it therefore less informative. This is because the inclusion of all data points is computationally expensive, which creates the goal of including as many non-trivial data points as possible without slowing down the process too much. Another goal is, to try to minimize the distance between positive pairs in the feature space as much as possible. As a result, the distribution in individual classes is extremely close. To solve the first limitation, the ranking loss uses all non-trivial data points for its structure in order to learn to differentiate more accurately. On the second one, Wang et al. (2019) [24] proposes to learn a hypersphere for each class respectively to widen the intra-class distribution. Their paper implemented the function in a convolutional neural network designed for image retrieval in computer vision. Images of CV and RS differ in various ways. Most of the CV datasets are single labelled in contrast to remote sensing images which are often multi labelled. The images in RS include Lidar and multispectral images, whereas CV works with RGB images most of the time. Furthermore, the composition of the images differs greatly. CV images are often composed of foreground and background, shifting the focus on the objective in the foreground. In RS that concept does not usually exist, images are mostly from the bird's eye view and all objects in an image are on the same plane.

1.2 Objective

The goal of this thesis is to implement a ranked list loss function [24] on a Convolutional Neural Network (CNN) that utilizes Soft Pairwise Similarity [27] and adapts RLL for multi label application in CBIR in RS. The trained model is then evaluated.

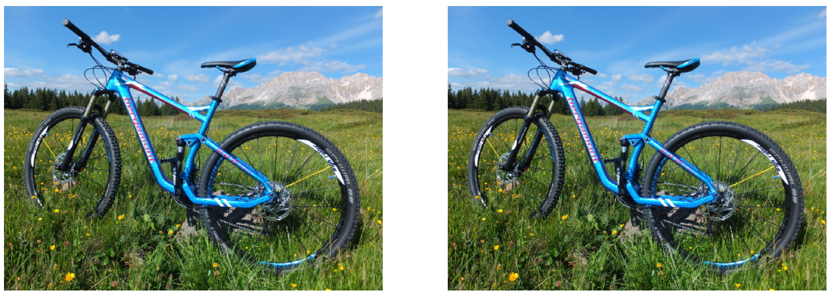
1.3 Outline

First, in Chapter 2, an overview of CBIR in RS is given. In Section 2.1.2 the Triplet Loss and its origin are presented. Three different approaches that are based on Triplet Loss conclude the related work. With Chapter 3 the Ranked List Loss [24] is introduced and explained. The difference between single label and multi label data is discussed and changes to the existing RLL for multi label use are introduced. Chapter 4 introduces the BigEarthNet data set. Further information about the ResNet model is provided and other aspects of the training setup are presented. Additionally the design of experiments and parameters for the training are listed. The Sensitivity Analysis of Experiments, in Chapter 5, describes the results of the conducted experiments. The different parameters and their impact on the training process and model performance are examined. In the end, Chapter 6 provides a summary of the thesis and an outlook on future work.

2 Related Work

2.1 Content Based Image Retrieval (CBIR)

The need to retrieve images from archives arose as soon as the first images were produced and stored away. At first, simple index cards with a shelf and box number and their content written out were enough to find images. But soon, index cards were digitized and with time images too. These first CBIR systems functioned like text retrieval system. A given search query was compared to all image labels. Images whose labels matched the search query were then retrieved. This process in itself functions rather well but has its downsides. The first is, that every image added to the archive has to be manually labelled with the content from the image, so that it can be later found within the archive. Secondly, selecting effective labels, to cover as much information from the images as possible, is difficult to achieve. There are many synonyms for tags in labels and to decide what a particular part of image represents, is often subjective, as can be seen in Figure 2.1. In addition, the person who labels the image has to decide which part of the image is important enough to be included in the label. And last, the retrieval efficiency of such a system is low [6]. Figure 2.1 shows the same image twice but with labels by different users. The lack of overlapping and partly differing labels indicates a problem for future use in CBIR systems.


(a) mountainbike, bicycle, blue, blue mountainbike, pasture, pastureland, mountain, alp, forest, woods, tire, bicycle seat, bicycle saddle, bike saddle, dandelion

(b) mountainbike, blue, meadow, mountain, forest, tire

Figure 2.1: Image of a blue bike labelled by two different user [2].

For a user who wants to search an image the keyword selection is just as difficult as choosing the correct labels for image archiving [5]. Since semantic similarity is subjective, another method to retrieve images had to be established. Automatic image retrieval analyses an image for its most important features and maps the image vector into an embedding space. This part is called feature extraction. Images can then be compared by measuring the distance to each other in the embedding space. Similar images have a short distance to each other, while dissimilar images are farther apart [5]. For the user, the process to retrieve an image is simple. An image, that has similiar content as what is searched, has to be provided to the system. The CBIR system then extracts the features of this query image, compares it to the archive of images in the embeddings space and retrieves images that are most similar to the images content. A representation of such a system is described in Figure 2.2.

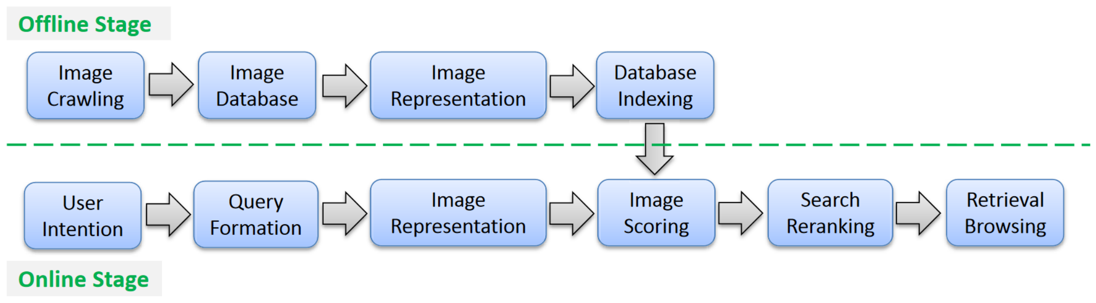


Figure 2.2: Framework for image retrieval systems [29].

How the system maps the input images, directly influences the performance of image retrieval. There are many different approaches that each try to optimize the image retrieval with different embedding functions. Some of these approaches are discussed in the section 'Similarity Ranking Functions' 2.1.2.

2.1.1 CBIR in Remote Sensing

Over the last years, the number of high-resolution RS images steadily increased [13]. This is caused by further development in Earth Observation Systems (EOS) and an increasing interest in RS. The management of large high-resolution data produced by EOS is an objective of CBIR for RS. CBIR in RS differs from other domains in that most images are not only RGB images but taken with frequencies that are not on the visible spectrum. Image capturing ranges from sonar systems on ships to Lidar on satellite and aircrafts. The objective of these images is not only to capture the visible part of the earth, but also other physical characteristics, which would otherwise not be captured. With more information, other data formats are needed to store the images. For training of a retrieval system, more information is available, which can influence the learning and retrieval itself. Examples of RS applications are among others the mapping of large forest fires [22][3], monitoring of agriculture [15][18], urban development [16], weather forecasting [20], and application in the military [7]. Cities are growing faster every year and with nearly half of the global population living in cities using only 5% of terrestrial surface [16] the importance of urban planning is undeniable. The demographic character of cities is influenced by its urban development and can be formed, if additional data such as RS images are at hand. CBIR in RS can help identifying growth areas that are in need of better infrastructure.

City districts can be categorized by quality of their buildings and with this information the cities budget can be adjusted. The military uses RS for area surveillance and detection of toxic gases in the atmosphere [7]. CBIR can also be used in weather forecasting to identify weather formations and predict possible future events. Future temperature, relative humidity, rainfall, wind speed and atmospheric pressure can be predicted by utilizing CBIR on weather satellite images [20].

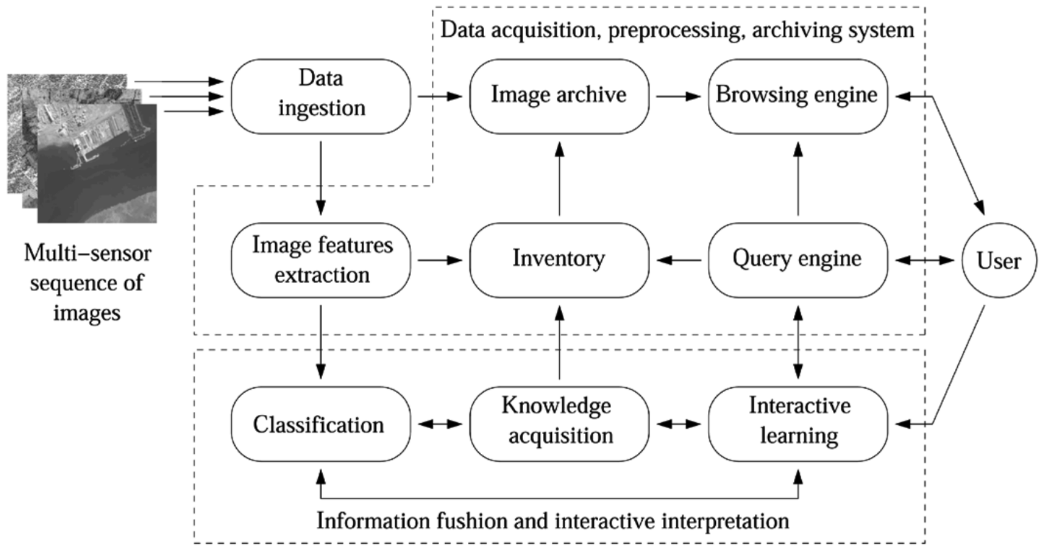

Data acquisition, preprocessing, archiving system
Data ingestion
Image archive
Browsing engine
Multi−sensor sequence of images
Image features extraction
Inventory
Query engine
User
Classification
Knowledge acquisition
Interactive learning
Information fushion and interactive interpretation

Figure 2.3: Framework for CBIR in RS [4].

A framework for CBIR in RS consists of multiple components. A user should be able to submit a query and retrieve images deemed important by the system. The system analyses the image and extracts its features. Query features are compared to a database of features and similar features are retrieved. These similar features are linked to the original images and the corresponding images are retrieved and presented to the user. A framework for CBIR in RS is depicted in Figure 2.3.

2.1.2 Similarity Ranking Functions

Deep learning models use loss functions to train neural networks. There are many different loss functions that are of interest for image retrieval in RS. The following is an introduction to the standard of such a loss and its variations. A model takes an image as input and outputs vectors in the embedded space, also called feature space. Applying the embedding function on all images of a dataset gives the correlating points to each image in the embedding space. The similarity of two images or rather their points in the feature space is determined by the Euclidean distance between them. For single labelled images one would try to cluster images for each class in the feature space so that similar images are in the same proximity in the feature space. The same applies to images with multiple labels, but balancing these clusters and finding the optimal setting for overlapping classes and similarities is undoubtedly more difficult than with labels for individual classes. Loss functions determine how the model learns and shape, how the embedding function maps the input into the feature space. Therefore, it is in our interest to model loss functions that optimize the mapping of the embedding function. The similarity of images in image retrieval is mostly determined by their Euclidean distance in the feature space. The objective of the loss function should be to maximize the distance of dissimilar images and reduce the distance between similar images in the feature space. The idea to minimize the distance between image pairs that are similar and widen the distance to dissimilar images using triplets has been around for a while. Today the proposal of the large margin nearest neighbour loss by Weinberger and Saul (2009) [25] is widely recognised as the prototype of the triplet loss [28]. The following explanation of Large Margin Nearest Neighbour (LMNN) is based on the proposal by Weinberger and Saul (2009) [25].

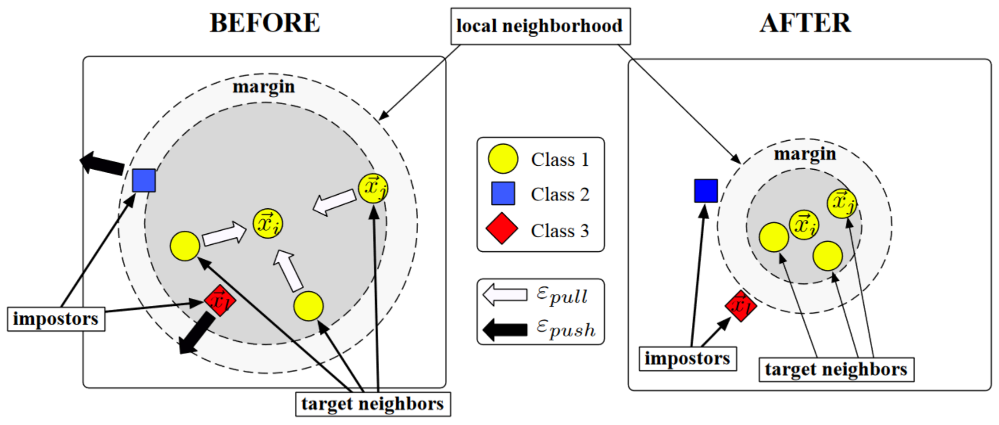


Figure 2.4: Large Margin Nearest Neighbor: Untrained feature vectors (left) and trained feature vectors after training (right) [28].

The distance between input and the target neighbour farthest away, plus a margin m creates a radius. Data points with different labels than the input are not allowed to breach this radius. If data points with different labels than the input are in the perimeter, they are impostors. $\mathbf{L}$ is the linear embedding function of the input into the feature space.

$$\|\mathbf{L}(\vec{x}_i - \vec{x}_l)\|^2 \leq \left\|\mathbf{L}(\vec{x}_i - \vec{x}_j)\right\|^2 + m \tag{2.1}$$

The input data point is $\vec{x}_i$, while $\vec{x}_l$ is the impostor data point and $\vec{x}_j$ is a target neighbour of $\vec{x}_i$. The margin m is established between the impostors and the target neighbours. The loss consists out of two terms. One, to pull target neighbours in the feature space closer to the input, thus penalizing large distances. The other term, $\varepsilon_{\text{push}}$, pushes impostors farther away from the input data point, penalizing small distances.

$$\varepsilon_{\text{pull}}(\mathbf{L}) = \sum_{j \rightsquigarrow y_i} \left\|\mathbf{L}(\vec{x}_i - \vec{x}_j)\right\|^2 \tag{2.2}$$

It is important to note that $\varepsilon_{\text{pull}}$ does not penalize all data points with similar labels, only target neighbors. For kNN classification to be accurate, data points in feature space do not have to be closely clustered [25].

$$\varepsilon_{\text{push}}(\mathbf{L}) = \sum_{i,j \rightsquigarrow il} \sum_{T} (1 - y_{il}) \left[1 + \left\|\mathbf{L}(\vec{x}_i - \vec{x}_j)\right\|^2 - \|\mathbf{L}(\vec{x}_i - \vec{x}_l)\|^2\right]_+ \tag{2.3}$$

$[z]_+ = \max(z, 0)$ is the standard hinge function. The value of y_{il} is 1 if $y_i = y_l$, otherwise 0. It follows that only dissimilar data points are penalized by $\varepsilon_{\text{push}}$. At last both terms are joined to form the loss $\varepsilon(\mathbf{L})$.

$$\varepsilon(\mathbf{L}) = (1 - \mu)\varepsilon_{pull}(\mathbf{L}) + \mu\varepsilon_{push}(\mathbf{L}) \tag{2.4}$$

The weighting parameter $\mu \in [0, 1]$ balances both terms. Depending on which value μ holds, target neighbours are getting pulled closer to the input, or impostors are pushed farther away. The same balancing is used in RLL 3.9. Many of Triplet Loss variations are based around features, such as pulling similar data pairs closer together, pushing dissimilar data pairs farther apart and establishing a margin between the input and dissimilar data points.

2.1.3 Triplet Loss

Schroff et al. (2015)[17] published in 'A Unified Embedding for Face Recognition and Clustering' a formally defined margin based triplet ranking loss. The following introduction into Triplet Loss is based on their publication. Weinberger and Saul (2009) are referenced in the context of nearest-neighbour classification as the motivation for the proposed triplet loss. The triplet loss function uses three images as an input using, one as an anchor and the other two as a positive and negative. Positive images have similar features as the anchor and negative images should be dissimilar to the anchor. The goal is to minimize the distance between the anchor and positive image and to maximize the distance between the anchor and negative image in the embedding space. This approach allows to work with many classes and still obtain good performance. To train the images on the triplet loss function, a CNN architecture with normalized output, that feeds into the triplet loss function, was used [17].

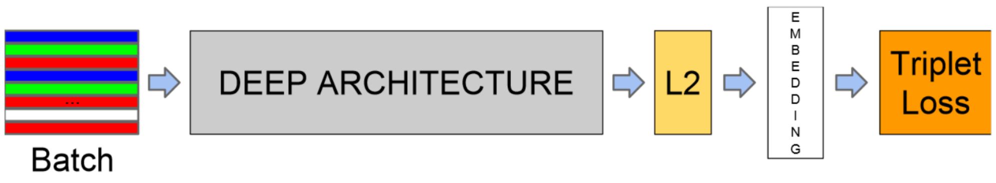


Figure 2.5: Structure for Triplet Loss training.[17]

A similar structure as seen in Figure 2.5 is used in the experiments in 'Data Set Description and Design of Experiments' 4 to train on the RLL function.

L_2 Norm

L_2 normalization is used to keep the coefficients of the model low. The L2 norm calculates the distance of the vector coordinate from the origin of the vector space, as shown below.

$$\|x\|_2 := \sqrt{x_1^2 + \cdots + x_n^2} \tag{2.5}$$

Triplet loss uses triplets of data points for its loss, the expected learning behaviour is shown in Figure 2.6.

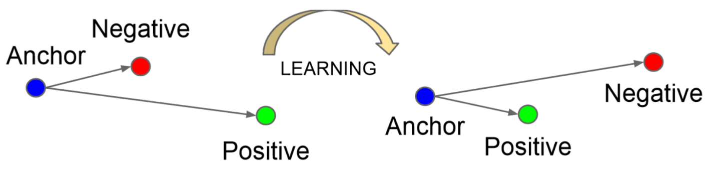


Figure 2.6: Triplet Loss: Image representation in embedding space before (left) and after training (right).[17]

To achieve the learning that is depicted in Figure 2.6, all data points have to fulfil the constraint 2.6 that the distance for each positive pair plus a margin α has to be smaller than their respective negative pair. The anchor image x_i^a, is closer to all positive images x_i^p that contain the same content as x_i^a than to any negative image x_i^n that does not resemble the anchor x_i^a for all images, while pushing negative images by a margin α away. $\mathscr{T}$ is the set of all possible triplets with cardinality N.

$$\left\|f(x_i^a) - f\left(x_i^p\right)\right\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2 \tag{2.6}$$

$$\forall \left(f(x_i^a), f\left(x_i^p\right), f(x_i^n)\right) \in \mathscr{T} \tag{2.7}$$

From this condition follows the loss to be minimized.

$$\sum_i^N \left[\left\|f(x_i^a) - f\left(x_i^p\right)\right\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha\right]_+ \tag{2.8}$$

$[x]_+ = \max(x, 0)$ is the standard hinge function.
Considering each data point as input then the distance between input and positive data points is minimized and distance between input and negative data points are be pushed away, so that it satisfies the constraint 2.6.

Triplet Selection

Using Triplet Loss with every triplet from $\mathscr{T}$ would result in slow convergence. Some triplets easily satisfy the restraint 2.6. Therefore, the network would not have to make much of a change and training is wasted. That is why it is important to choose hard triplets, that violate constraint 2.6, so that the network can be efficiently improved. Given x_i^a, a hard positive data point x_i^p from the training batch should be chosen, such that the distance in feature space to x_i^a is maximized. To maximize the learning effect we want to find the pair, that violates the constraints as much as possible [17]. The hard positive data point x_i^p is chosen, that its distance to the anchor x_i^a is maximized.

$$\operatorname{argmax}_{x_i^p} \left\|f(x_i^a) - f\left(x_i^p\right)\right\|_2^2 \tag{2.9}$$

A hard negative x_i^n should be chosen, so that the distance in feature space to x_i^a is minimized.

$$\operatorname{argmin}_{x_i^n} \|f(x_i^a) - f(x_i^n)\|_2^2 \tag{2.10}$$

It is an expensive task to calculate a triplet, that satisfies both conditions for every image of the dataset. Two options are presented with the last one utilized. The first, using the latest checkpoint of the model, every n steps $\operatorname{argmin}_{x_i^n}$ and $\operatorname{argmax}_{x_i^p}$ are calculated for a subset of the dataset. The other option calculates triplets online while training. Using the size of smaller batches when training the model, $\operatorname{argmin}_{x_i^n}$ and $\operatorname{argmax}_{x_i^p}$ are recalculated for each training batch [17]. With smaller batches one has to ensure that enough positive pairs of each class are in the batch included. If there are too few or even no positive pairs the triplet loss would be imbalanced. Schroff et al. (2015) [17] found that using all anchor positives resulted in better results than using hard anchor-positives. With all anchor positives the model was more stable and converged slightly faster in the beginning of training. Since selecting hard negative pairs can result in local minima during training, the worst case scenario is a collapsed model, Schroff et al. (2015) [17] used semi-hard negative pairs. Semi-hard negative pairs x_i^n are farther away from x_i^a than x_i^p, but their squared distance is still close to the anchor-positive distance. Even though they are semi-hard pairs, x_i^n lie inside the margin α. Equation 2.11 describes this constraint for semi-hard pairs.

$$\left\|f\left(x_i^a\right)-f\left(x_i^p\right)\right\|_2^2<\left\|f\left(x_i^a\right)-f\left(x_i^n\right)\right\|_2^2 \tag{2.11}$$

As described in the section above, the triplet selection is an important part for model training to achieve a fast convergence rate. Regarding performance triplet loss set a new standard for loss functions and feature mapping. But there are still some aspects that are to improve. Since triplet loss pulls similar data points as close as possible the intra-class variance is not assured. Further the selection of hard-negative pairs can lead to local minima in training. The following section lists three alternatives that are derived from triplet loss.

2.1.4 Ranking Motivated Loss

N-Pair-MC

This overview of N-Pair-MC is based on the publication ”Improved Deep Metric Learning with Multi-class N-pair Loss Objective” by Sohn (2016)[10]. Given a query N-pair-mc aims to learn by identifying one positive example from N-1 negative examples, where every class is represented. The loss takes inspiration from the (N+1)-tuplet loss[10]. The (N+1)-tuplet loss is build as shown in Equation 2.12.

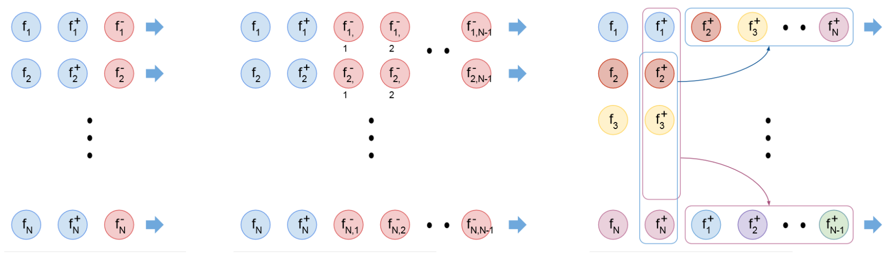
$$\mathscr{L}\left(\left\{x,x^+,\{x_i\}_{i=1}^{N-1}\right\};f\right)=\log\left(1+\sum_{i=1}^{N-1}\exp\left(f^\top f_i-f^\top f^+\right)\right) \tag{2.12}$$

(N+1) tuplet loss maximizes the distance between query and negative examples and minimizes it between query and a positive, which is identified from the negative examples. It is beneficial for the loss if the negative examples contain all classes. But with a larger number of classes, the difficulty of standard optimization increases [10]. If N=2 the (N+1) tuplet loss is similar to the standard triplet loss:

$$\mathscr{L}_{(2+1)\text{ -tuplet}}\left(\left\{x,x^+,x_i\right\};f\right)=\log\left(1+\exp\left(f^\top f_i-f^\top f^+\right)\right) \tag{2.13}$$

$$\mathscr{L}_{\text{triplet}}\left(\left\{x,x^+,x_i\right\};f\right)=\max\left(0,f^\top f_i-f^\top f^+\right) \tag{2.14}$$

The advantage of N-pair-mc over (N+1) tuplet loss or triplet loss is that N-pair-mc requires only 2N, triplet loss 3N and (N+1)-tuplet loss (N+1)N passes to evaluate embedding vector [10], see Figure 2.7. N-pair-mc improves the triplet-loss by pushing multiple negative examples from different classes away [10].



(a) Triplet loss (b) $(N+1)$-tuplet loss (c) N-pair-mc loss

Figure 2.7: Batch construction of N-pair-mc loss in comparison to Triplet Loss and (N+1)-tuplet loss.[10]

The N-pair-mc loss takes clear inspiration by (N+1) tuplet loss $\log\left(1+\sum_{i=1}^{N-1}\exp\left(f^{\top}f_i - f^{\top}f^{+}\right)\right)$.

$$\mathscr{L}_{N\text{-pair-me}}\left(\left\{\left(x_i,x_i^{+}\right)\right\}_{i=1}^{N};f\right)=\frac{1}{N}\sum_{i=1}^{N}\log\left(1+\sum_{j\neq i}\exp\left(f_i^{\top}f_j^{+}-f_i^{\top}f_i^{+}\right)\right) \tag{2.15}$$

N-Pair-MC utilizes negative examples from multiple classes but does not take advantage of a margin like triplet loss does.

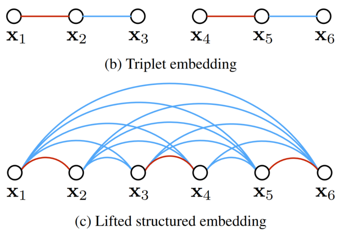
Lifted Struct

This introduction into Lifted Struct is based on the publication by Song et al. 2016[19]. Similar to Triplet loss Lifted Struct tries to pull one positive pair as close as possible, but utilizes all negative data points comparing them to the positive pair and pushes them as far as a margin α away.

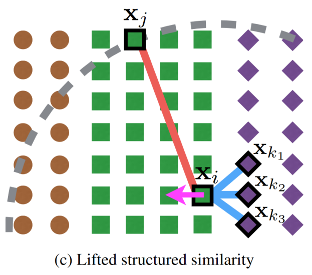
$$\tilde{J}_{i,j}=\log\left(\sum_{(i,k)\in\mathscr{N}}\exp\left\{\alpha-D_{i,k}\right\}+\sum_{(j,l)\in\mathscr{N}}\exp\left\{\alpha-D_{j,t}\right\}\right)+D_{i,j} \tag{2.16}$$

$$\tilde{J}=\frac{1}{2|\mathscr{P}|}\sum_{(i,j)\in\mathscr{P}}\max\left(0,\tilde{J}_{i,j}\right)^2 \tag{2.17}$$

D is a dense pairwise squared distance matrix that is computed over all batch elements in training.



(a) Construction for pair wise edges with training training batch. Red edge represent a similar relation between the data points and blue edge represent a dissimilar relation.



(b) Training of Lifted Struct. The gray dashed border represents the margin. The arrow indicates the direction of the negative gradient for positive training data points. A red edge represents similarity and a blue edge dissimilarity.

Figure 2.8: Relation of data points and training for Lifted Struct. [19]

Instead of forming triplets for each datapoint and utilizing only two data points, Lifted Struct considers every data point. The difference is shown in Figure 2.8.

Proxy NCA

The following explanation of Proxy NCA is based on the publication ”No Fuss Distance Metric Learning using Proxies” by Movshovitz-Attias et al. (2017) [14]. Proxy-NCA loss uses NCA loss with proxies of the data set. That way the real data points are each represented by a proxy. Advantages over triplet loss are that less triplets have to be generated and fewer resources are used. With that, larger datasets can be utilized and more information be processed. NCA maximizes the number of correct expected classified data points [8]. Since the Proxy NCA does not calculate the loss based on the real dataset, but proxies, over fitting of the model should not be expected. During training Goldberger et al. (2005) did not experience any over fitting, and specify that the larger the expected number of points correctly classified, the better their model performs during testing [8].

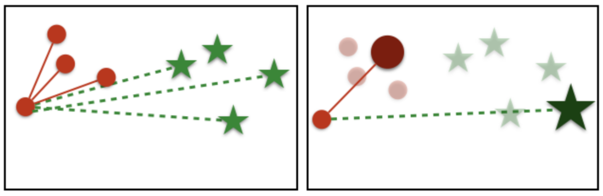


Figure 2.9: Proxy NCA choose proxies for cluster in the data set and reduces the number of triplets that can be created and therefore the number of comparisons that have to be calculated [14].

A proxy $p(u)$ for a data point u is chosen such that for every data point u there is a proxy with minimal distance to u in the positive set P of pairs.

$$p(x) = \arg\min_{p \in P} d(x,p) \tag{2.18}$$

The proxy approximation error is the worst approximation with the largest distance d, over all data points [14].

$$\varepsilon = \max_{x} d(x, p(x)) \tag{2.19}$$

With the proxies representing the training data set, the number of triplets generated and used for the loss is greatly reduced [14], as can be seen in Figure 2.9. Z is the negative set with p(z) as the proxy of negative points.

$$L(\mathbf{a},\mathbf{u},\mathbf{Z}) = -\log\left(\frac{\exp(-d(\mathbf{a},p(\mathbf{u}))))}{\sum_{\mathbf{z}\in\mathbf{Z}}\exp(-d(\mathbf{a},p(\mathbf{z})))}\right) \tag{2.20}$$

3 Proposed Method

3.1 Introduction into Ranked List Loss

The following introduction into RLL and explanation of RLL are based on the publication by Wang et al. (2019) [24]. The introduction of the RLL function seeks to solve problems of Triplet Loss. RLL wants to distance dissimilar images (negative) in the feature space from the anchor and bring similar images (positive) and the anchor closer together. The positive and negative set should have at least one image to take advantage of the weighting. A trained model should be able to rank images based on a query image from most similar to most dissimilar. To achieve a margin between these sets, RLL uses several parameters. The boundary α is the distance to divide the query and the negative set. The margin m is the distance between the positive boundary and the negative boundary, therefore is $\alpha - m$ the boundary for the positive set. RLL sorts all data points between the positive and negative boundary into either set to establish a result for the question whether they are similar to the query or not. The RLL sorts all images in the batch into two sets of positive and negative images based on the similarity to the anchor. The algorithm lists the positive data points first and then the negative ones, forcing a margin between the data points in the feature space. An advantage over triplet loss is, that all non-trivial data points are utilized. Additionally, Triplet Loss compromises the intra class variance by pulling positive data points very close to each other. RLL pulls positive data into a hypersphere with diameter $\alpha - m$ to tackle that issue. Wang et al. compare the proposed function with state-of-the-art approaches and conclude that the proposed approach with RLL achieves state-of-the-art performance [24].

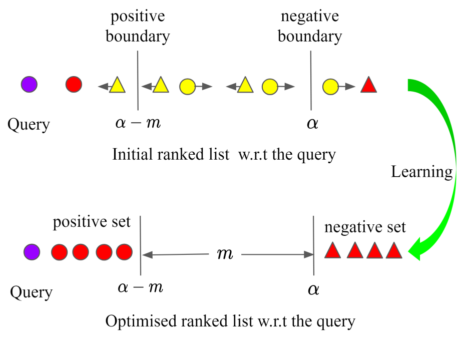


Figure 3.1: Shapes represent different classes, while purple represents the query, red data points that are either in the positive dataset or the negative dataset. Yellow objects are data points that need to be pushed farther away or pulled closer [24].

The structure of the RLL algorithm is depicted in Figure 3.2:
For each data point from the batch, a positive and negative set is created (Example Mining). Then the weights for the positive and negative set are calculated. After the 'Example Weighting' is finished the loss for each data point is calculated and in the end, all losses accumulated return the final loss [24].

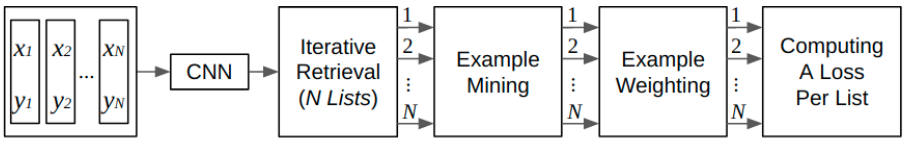


Figure 3.2: Structure of the RLL algorithm [24].

3.2 Ranked List Loss

3.2.1 Pairwise constraint

Wang et al. (2019) [24] chose pairwise margin loss by Wu et al. (2017) [26] as the basic pairwise constraint.

$$L_m(x_i, x_j; f) = (1 - y_{ij})[\alpha - d_{ij}]_+ + y_{ij}[d_{ij} - (\alpha - m)]_+ \tag{3.1}$$

The query image is represented as x_i and images from either the positive set or the negative set as x_j. Labels for x_i and x_j are y_i and y_j. If the labels are equal L_m for positive pairs is calculated. On the other hand, if the labels are not equal $y_i j$ is set to 0 and L_m for negative pairs is calculated.

$$\begin{aligned} y_{ij} &= 1, \text{ if } y_i = y_j \\ y_{ij} &= 0, \text{ if } y_i \neq y_j \end{aligned} \tag{3.2}$$

$d_{ij} = \left\| f(x_i) - f(x_j) \right\|_2$ is the Euclidean distance between two points in feature space with f being the embedding function.

The positive images are pulled closer to the anchor by a defined threshold $\alpha - m$, while simultaneously establishing a margin m between the positive and negative set. Therefore L_m computes $[d_{ij} - (\alpha - m)]_+$ for positive images and $[\alpha - d_{ij}]_+$ for negative images.

3.2.2 The Positive and Negative Set

The positive image set contains all images from the image subset that have the same label as the anchor. The negative image set contains images from the image subset that have not the same label as the anchor. To avoid trivial data points, which would not challenge the model to classify the trivial data point easily into one of the sets only images for the positive set that satisfy the condition $d_{ij} > (\alpha - m)$ are included. Likewise for the negative set only images that satisfy the condition $d_{ij} < \alpha$ will be included. For the query x_i^c follows:

$$\mathrm{P}^*_{c,i} = \left\{x_c^j \mid i \neq j, d_{ij} > (\alpha - m)\right\} \tag{3.3}$$

$$\mathrm{N}^*_{c,i} = \left\{x_j^k \mid k \neq c, d_{ij} < \alpha\right\} \tag{3.4}$$

Wang et al. (2019) [24] call this step 'Mining informative pairs'. Mining informative pairs can help to speed convergence and improve generalization performance if properly designed [24].

3.2.3 Weighting Pairs

With a large number of non-trivial images in each set with a different impact on the overall loss, each pair is weighted based on their loss value. For the positive and negative set the loss value will be multiplied by a factor T which stands for temperature and then the exponential function is applied. The value correlates with how much each pair violates its constraint. Both sets have different temperatures T_p and T_n. Through the temperatures, the degree of the slope of the function can be regulated. For $T_n = 0$ every data point from the negative set is treated equally. With $T_n = +\infty$ the example mining becomes extremely difficult.

$$w_{ij} = \exp\left(T_n \times (\alpha - d_{ij})\right), x_j^k \in \mathrm{N}^*_{c,i} \tag{3.5}$$

The weighting of positive image pairs is controlled through the parameter T_p. For $T_p > 0$ positive pairs with a large distance are heavier weighted. If $T_p < 0$, positive pairs that are closer to each other are assigned more weight. That is often applied, if there are many positive pairs to sustain local similarity manifold structure.

$$w_{ij} = \exp\left(T_p \times (d_{ij} - (\alpha - m))\right), x_j^c \in \mathrm{P}^*_{c,i} \tag{3.6}$$

3.2.4 Overall Optimization Objective

To learn a class hypersphere for positive data pairs, the loss for positive pairs is minimized.

$$L_P\left(x_i^c; f\right) = \sum_{x_j^c \in \left|\mathrm{P}_{c,i}^*\right|} \frac{w_{ij}}{\sum_{x_j^c \in \left|\mathrm{P}_{c,i}^*\right|} w_{ij}} L_m\left(x_i^c, x_j^c; f\right) \tag{3.7}$$

To push negative datapoints further away beyond the boundary α, the loss for negative pairs is minimized.

$$L_N\left(x_i^c; f\right) = \sum_{x_j^k \in \left|\mathrm{N}_{c,i}^*\right|} \frac{w_{ij}}{\sum_{x_j^k \in \left|\mathrm{N}_{c,i}^*\right|} w_{ij}} L_m\left(x_i^c, x_j^k; f\right) \tag{3.8}$$

Both minimization tasks are joined in the overall RLL L_{RLL}.

$$L_{RLL}\left(x_i^c; f\right) = (1-\lambda) L_P\left(x_i^c; f\right) + \lambda L_N\left(x_i^c; f\right) \tag{3.9}$$

With λ giving the opportunity to control the balance between the impact of the loss of positive and negative images. By default λ is set to 0.5 so that both sets contribute equally towards the loss. According to a study conducted by Wang et al. (2019) [24] it is an improper practice to pull positive pairs as close as possible. Therefore RLL pulls pairs into a hypersphere with a diameter of $\alpha - m$. This solves one of the problems from Triplet loss and preserves the intra-class variance.

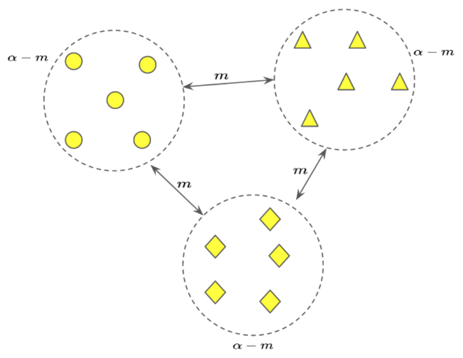


Figure 3.3: Hyperspheres in the embedding space. Different shapes represent different classes. The distance between each is the margin m as a minimum. With hyperspheres the intraclass distribution can be preserved. The diameter of each class hypersphere is $\alpha - m$ [24].

.

3.2.5 Calculating RLL

For each image in the batch the RLL 3.9 is computed and their average results in the final loss.

$$L_{RLL}(X;f) = \frac{1}{N} \sum_{\forall c, \forall i} L_{RLL}(x_i^c;f) \tag{3.10}$$

With $N = \sum_c N_c$ being the batch size.

3.2.6 RLL Algorithm

The algorithm of RLL on one mini-batch is as follows defined:

Algorithm 1 Ranked List Loss on one mini-batch

```
1: Mini-Batch Settings: The batch size $N$, the number of classes $C$, the number of images per class $N_c$.
2: Parameters: The distance constraint $\alpha$ on negative points, the margin between positive and negative examples $m$, the weighting temperature $T_n, T_p$.
3: Input: $\mathrm{X} = \{(\mathrm{x}_i, y_i)\}_{i=1}^N = \left\{\{\mathrm{x}_i^c\}_{i=1}^{N_c}\right\}_{c=1}^C$, the embedding function $f$, the learning rate $\beta$.
4: Output: Updated $f$.
5: Step 1: Feedforward all images $\{\mathrm{x}_i\}_{i=1}^N$ into $f$ to obtain the images' embeddings $\{f(\mathrm{x}_i)\}_{i=1}^N$.
6: Step 2: Iterative retrieval and loss computation.
7:     foreach $f(\mathrm{x}_i^c) \in \left\{\{f(\mathrm{x}_i^c)\}_{i=1}^{N_c}\right\}_{c=1}^C$ do
8:         Mine non-trivial positive set $\mathbf{P}_{c,i}^*$ 3.3.
9:         Mine non-trivial negative set $\mathrm{N}_{c,i}^*$ 3.4.
10:        Weight negative examples using Eq. 3.5.
11:        Weight positive examples using Eq. 3.6.
12:        Compute $L_{\mathrm{P}}(\mathrm{x}_i^c;f)$ using Eq. 3.7.
13:        Compute $L_{\mathrm{N}}(\mathrm{x}_i^c;f)$ using Eq. 3.8.
14:        Compute $L_{\mathrm{RLL}}(\mathrm{x}_i^c;f)$ using Eq. 3.9.
15:    end for
16:    Compute $L_{\mathrm{RLL}}(\mathbf{X};f)$ using Eq. 3.10.
17: Step 3: Gradient computation and back-propagation to update the parameters of $f$:
18:    $\nabla_f = \partial L_{\mathrm{RLL}}(\mathrm{X};f)/\partial f$
19:    $f = f - \beta \cdot \nabla_f$
```

The Algorithm is taken from Wang et al. (2019) [24].

3.3 Differences in single label and multi label

Single label data usually defines a number for each class of its data set. Each data point of the data set is then assigned a number as a label. Multi label data with n classes uses n-dimensional vectors. Each class corresponds to a dimension of the label vector. The classes an image is assigned to, are characterised as a 1, classes where the data point does not belong to, are characterised as a 0 in the label vector. Accuracy in classification is determined by the similarity of the generated label by the embedding function and the real label. A similar approach is taken in the domain of image retrieval. Images are mapped by the embedding function into the feature space The cluster and embedded images that are closest to the input image are similar images that are retrieved. To determine the similarity of multi labels, an element wise comparison of the labels with each other works. But with increasing classes the comparison of labels and their subclasses becomes increasingly difficult. That is why it is necessary to determine the overall similarity of the label vectors. Figure 4.3 shows multi label images that are similar to each other. The labels overlap but are too different, that a simple comparison may not suffice.

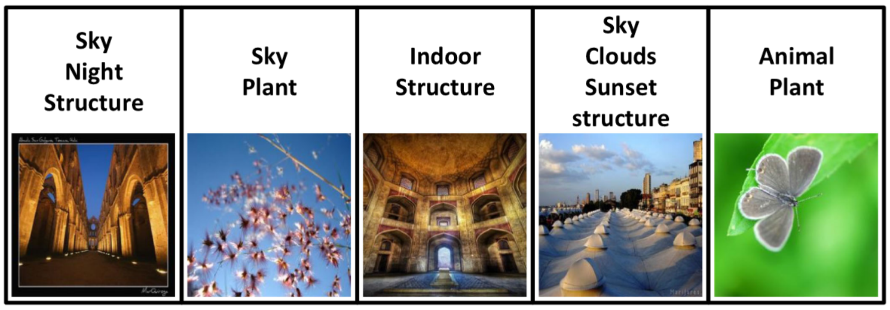


Figure 3.4: Similar images with different labels. [27]

To achieve an estimation of similarity the cosine similarity function produces good results. The cosine similarity calculates the similarity between two vectors and equals the cosine of the angle between the vectors. The calculated result is a value between zero and one. Whether or not two vectors are similar is then defined by a threshold. The first to introduce this function into comparison for labels were Zhang et al. (2020), calling it Soft Pairwise Similarity [27].

$$\text{softSimilarity}\,(x,y) = \frac{x \cdot y}{||x||\,||y||} \tag{3.11}$$

Most loss functions do not discriminate between single label and multi label inputs. If they do labels are often compared, to sort inputs into different sets to work with, as in the RLL approach for multi label. To utilize the function effectively the calculated similarity has to cross a certain threshold to be considered similar. This threshold can be chosen arbitrarily, e.g. to 70%, but a selection process may prove better results.

3.4 Changes in RLL for multi label adaption

The proposed method of RLL for multi label adaption is heavily motivated by Wang et al. (2019) [24]. The main objective of change concerns the distinction of labels. A simple comparison, whether or not two labels of multi labelled RS images are equal, is not enough. Through the use of Soft Pairwise Similarity the similarity of two labels is determined. A threshold t_{sim} determines if two images are equal or not. The following details the changes for multi label adaption in RLL. For the pairwise constraint, the conditions for y_{ij} have to be adjusted, because a simple comparison between labels is not sufficient. y_{ij} takes the value 1 if the Soft Pairwise Similarity exceeds the threshold t_{sim}, the image pair is deemed similar. If the Soft Pairwise Similarity does not meet the threshold criteria y_{ij} is assigned the value 0.

$$L_m(x_i, x_j; f) = (1 - y_{ij})[\alpha - d_{ij}]_+ + y_{ij}[d_{ij} - (\alpha - m)]_+ \tag{3.12}$$

$$y_{ij} = 1, \text{if} \quad \text{softSimilarity}(y_i, y_j) \geq t_{\mathrm{sim}} \tag{3.13}$$

$$y_{ij} = 0, \text{if} \quad \text{soft Similarity}(y_i, y_j) < t_{\mathrm{sim}} \tag{3.14}$$

Since images are no longer separated by their class, cosine similarity is used to determine the set an image is assigned to. For images, that violate the positive constraint, to be added to the positive set softSimilarity (y_i, y_j) must exceed the threshold t_{sim}. Images that violate the negative constraint are only added to the negative set if the similarity with the input falls below t_{sim}.

$$P_i^* = \{x_j \mid i \neq j, d_{ij} > (\alpha - m), \text{ softSimilarity } (y_i, y_j) \geq t_{sim}\} \tag{3.15}$$

$$\mathrm{N}_i^* = \{x_j \mid i \neq j, d_{ij} < \alpha, \text{ softSimilarity } (y_i, y_j) < t_{sim}\} \tag{3.16}$$

The remaining part of the algorithm remains unchanged in the sense, that all connections to classes are removed and the equations rely on the images from the positive and negative set.

4 Data Set Description and Design of Experiments

4.1 Description of the Data Set

The BigEarthNet was created in the need of a more convenient training source in the context of image retrieval and scene classification. Out of this necessity the dataset is larger than most other existing archives in RS. The image patches are multi labelled and are provided by the CORINE Land Cover database of the year 2018 (CLC 2018) [21]. The Corine Land Cover (CLC) exists since the 1980s to provide consistent land cover land use (LCLU) maps of Europe. Today images of 39 European countries are in the database. The dataset consists out of 590.326 Sentinel-2 image patches with three different bands: 10m, 20m and 60m. The 10m bands are 120 x 120 pixel-, 20m for 60 x 60 pixel- and 60m for 20 x 20 pixel-images. All images in the BigEarthNet data set are from ten European countries (Austria, Belgium, Finland, Ireland, Kosovo, Lithuania, Luxembourg, Portugal, Serbia, Switzerland) [21]. Images were taken in all seasons, with fewer than the average of images in winter. Hereby images are excluded that show cloud coverage as seen in Figure 4.2. The image count of the BigEarthNet data set is as follows: spring - 175937, and summer - 126913, autumn - 143557, winter - 72877.

BigEarthNet-19 Labels

New labels for BigEarthNet were introduce to support better training of deep learning models, examples can be seen in Figure 4.1. From the original label set 11 classes were removed and 22 classes were merged into 9 new classes. Ten classes of the original label set remain the same for the new label set [21].

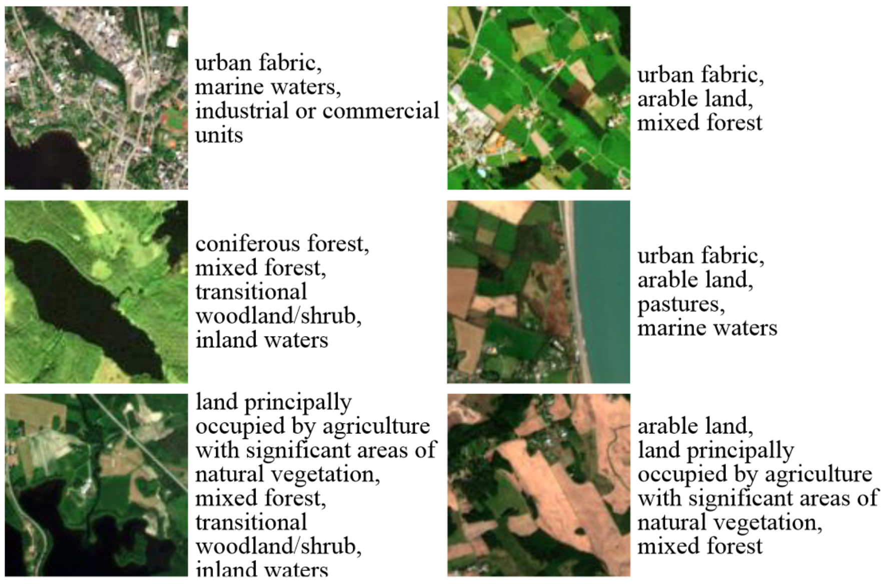

urban fabric,
marine waters,
industrial or commercial units
urban fabric,
arable land,
mixed forest
coniferous forest,
mixed forest,
transitional woodland/shrub,
inland waters
urban fabric,
arable land,
pastures,
marine waters
land principally occupied by agriculture with significant areas of natural vegetation,
mixed forest,
transitional woodland/shrub,
inland waters
arable land,
land principally occupied by agriculture with significant areas of natural vegetation,
mixed forest

Figure 4.1: Updated labels for the BigEarthNet data set.

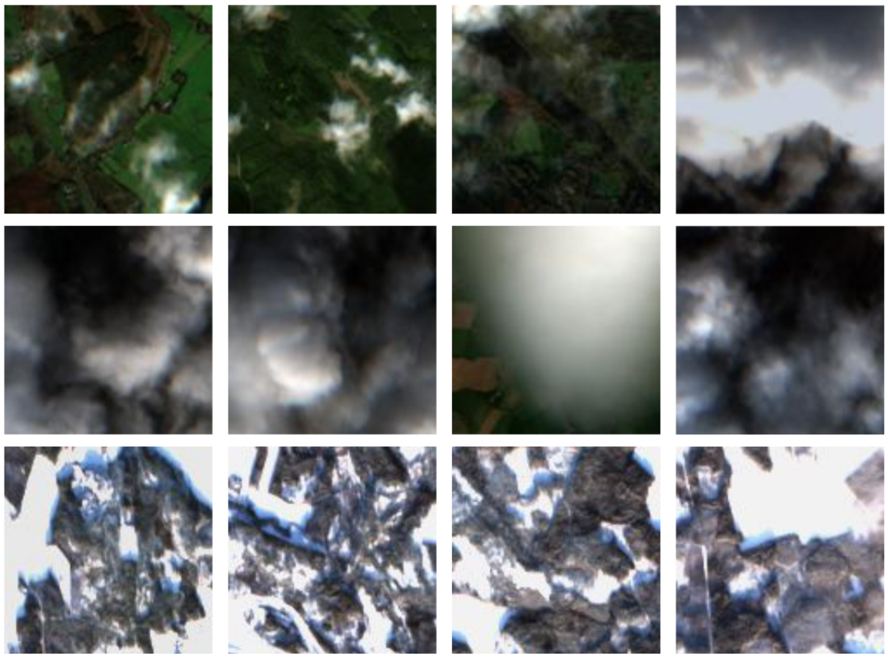


Figure 4.2: BigEarthNet images with cloud and snow coverage.

4.2 Design of Experiments

Ireland dataset

With BigEarthNet being a large dataset, the time to train a model increases. Consequently, the training of the model was undertaken with a smaller subset of images from Ireland. The training set consists out of 8197 images, the validation set out of 3839 and the test set consists out of 3858 images. A smaller subset allows for shorter training, while still capturing information of similar data form the whole dataset.

PyTorch

The implementation of RLL by Wang et al. (2019) [24] described RLL and the deviation described in the previous chapter is done in PyTorch. PyTorch is an open source learning library developed by Facebook's AI Research lab. The library finds application in computer vision and natural language processing [1]. On top of PyTorch, PyTorch Lightning is used. PyTorch Lightning abstracts code and creates a uniform framework for researchers to work with. So that the code becomes more accessible to other researchers, through clearly dividing research code and training code.

PyTorch Custom Loss Function

To backpropagate and calculate the gradients, PyTorch stores every operation a tensor is part of in the tensor and creates a traversable tree. To calculate how the parameters of a model have to change, the tree is traversed and gradients are calculated. A custom loss function in PyTorch can be created, by only using PyTorch functions for operations. The gradient is then calculated automatically by PyTorch.

High Performance Computing Cluster TU Berlin

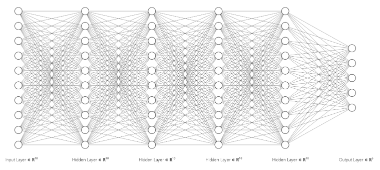
The HPC cluster is used for academic research by members of the TU Berlin. The cluster provides, among other computational nodes, 20 GPU nodes and 132 MPP nodes, with Linux as the operating system. Each GPU node consists out of 2x NVIDIA Tesla P100 16GB HBM2, 2x CPU INTEL Xeon E5-2630v4 and 16 x DDR4-2400 32GB RAM. The MPP nodes are built from 2x CPU INTEL Xeon E5-2630v4 and 8 x DDR4-2400 32GB ECC RAM.

ResNet

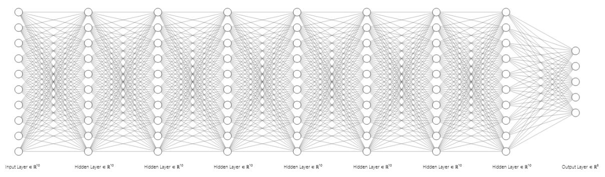
A pretrained model ResNet50 was provided by Sumbul et al. (2019) [21], 50 referring to the number of layers of the model. The model is trained on BigEarthNet with the new class nomenclature with 19 classes. To be able to use the model for image retrieval the last layer has to be removed and a new fully connected layer is inserted with an output of the size 2048.
The following is based on the publication by He et al. (2015) [9]. Convolutional Neural Networks in image recognition have gained immense progress in recent years. With increasing computational power the depth of neural networks increased as well. But increasing depth of networks does not equal an improvement of the embedding function of the network itself as the findings from He et al. (2015) [9] shows. ResNet was created to overcome the degradation problem and vanishing gradient problem. ResNet achieves training a multitude of layers, while still maintaining performance.

Degradation Problem

Let m and n be numbers of layers for neural networks where $m > n$. Network A contains n layers and network B contains m layers. Both networks are trained equally 4.3. The network B should perform as well as network A since the first n layers of network B could be replaced by the layers of network A and the rest of the layers are the identity mapping. Therefore, network A and network B should generate the same representation, which is not the case as presented in Figure 4.4. With more layers in a deep neural network, the effect of a higher training and testing error is observed.



(a) Network A with n layers



(b) Network B with m layers.

Figure 4.3: Networks with different layer sizes.

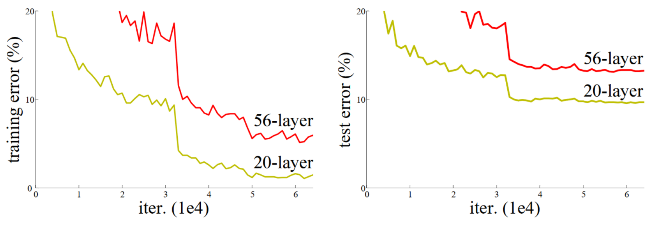


Figure 4.4: Networks with 20-layers and 56-layers, trained on CIFAR-10 dataset showing an unexpected phenomena. In theory, the same error is expected for both networks [9].

Vanishing gradient problem

The vanishing gradient problem applies to neural networks, that are based on gradient learning and back propagation. Weights of the neural network are updated through back propagation. The vanishing gradient problem describes the case, that the gradient holds such a small value that the updated weights value effectively does not change, leading in the worst case to a stop of the training process.

The curse of dimensionality

The curse of dimensionality describes the problem that with increasing dimensions the Euclidean distance between vectors in these dimensions decreases. Regarding image retrieval, it means that images get more similar to each other, which is the opposite of what is wanted.

Residual blocks

Usually in a neural network, each layer sends its input to the next layer and so on. In residual blocks each layer sends its input to the next layer, additionally skips the next two layers and inserts the input again, as can be seen in Figure 4.5.

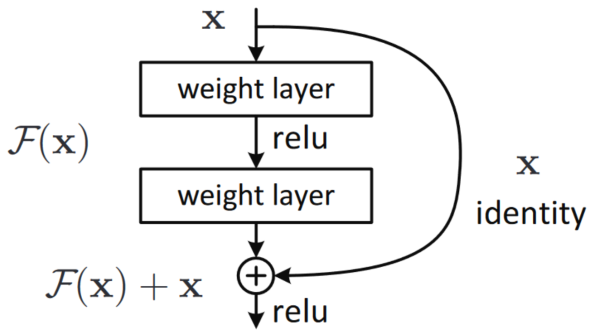

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 4.5: A building block for residual learning [9].

Using this technique the described problems can be prevented and detailed features of images can be learned with a multitude of layers.

Parameters

Parameters that influence the loss function are α, m, T_p, T_n, similarity, batch size and embedding size of the model.

Batch size

The larger the batch size, the more non-trivial data points can be utilized to calculate the loss. Thus, it results in a more stable and accurate loss value regarding the parameters of the function. The GPUs have a memory capacity of 16GB, which puts a restraint on the batch size, the models can train on.

Embedding size

The last layer defines the granularity with which the model is able to map images in the feature space. It also defines the computing power it takes to calculate the loss. A larger vector results in more calculations and comparisons, therefore an ideal output size is to be determined that does not contain unnecessary information, while also not disregarding information.

Alpha and margin *m*

Due to the L2 normalization the Euclidean distance between any two point in the embedding space ranges from 0 to 2. Therefore α is not allowed to be greater than two. The distance between any two hypersphere boundaries is no less than m [24]. The boundary α influences how far negative examples should be pushed away and the margin m how close positive examples should be pulled. Furthermore these parameters have an impact on which data points violate constraints 3.3, 3.4 and are then added to the positive or negative set.

Weighting temperatures

The weighting temperature T_p and T_n influence the differentiation between data points. Data points that already violate the constraints of RLL are weighted exponentially heavier with adjusted temperature. Larger weights have a greater impact on the loss, and lighter weights a smaller impact.

Similarity

The parameter similarity decides which images are considered non-trivial regarding the query image. A low value indicates that many images are deemed similar and learning of the model should be slower because dissimilar images are easy categorized into the negative set. A high value for similarity equals only non-trivial images are chosen for the positive set. Preliminary the value 0.7 is chosen.

4.2.1 Design of Experiments

The training of the ResNet model was conducted under changes of the listed Parameters. All tests are run on a NVIDIA Tesla P100 16GB from the HPC cluster of the TU Berlin. The optimizers that are used are the Stochastic Gradient Descent (SGD) and Adam [12]. Pre-trained ResNet weights are provided by Sumbul et al. and are used to train the models. The weights were trained for classification, so it will be interesting, how the results differ to a trained model without pre-trained weights.
The precision of a model is calculated by retrieving similar images by distance for an input from a batch sample X. The retrieved images are sorted ascending by their distance and the top k images X_k are chosen into a list, with k $\in \mathbb{N}$. The number of images X_p from

$$X_k$$

that are deemed similar (Soft Pairwise Similarity), by crossing the similarity threshold, are divided by k. The mean over all lists results in the precision.

$$\text{Precision } = \frac{|X_k \cap X_p|}{|X_k|} \tag{4.1}$$

5 Experimental Results

This chapter describes the results of the conducted experiments. Different parameters and their impact on the training process are examined.

5.1 Sensitivity Analysis of Experiments

Boundary α

Wang et al. (2019) [24] choose α depending on m as $\alpha = 1 + \frac{m}{2}$. That way, a margin m can be selected and the boundary α is automatically decided. Therefore, the first trained models used this formula for the value of α by selecting and changing m, as can be seen in table 5.1.

Margin m

The margin m is the first parameter that was changed for training because it is one of the most important ones regarding the loss function. Positive data points are pulled into a hypersphere of diameter $\alpha - m$. For the first model iterations, m influences the value of α, which is the distance data pairs from the negative set were to be pushed away. Considering the normalization layer of the model a distance between two data points lies within the interval [0,2]. The parameters to train the model further are $m = 1.0$ and $\alpha = 1.5$. These values scored the highest precision and are expected to not over fit the model as a large margin of 1.5 and the related α value 1.75 would possibly do. The lower precision scores of 52.05% is attributed to a small margin that does not divide embedded images far enough. Wang et al. (2019) [24] emphasize the distancing of negative pairs and choose to not weight the positive pairs, $T_p = 0$. The negative temperature for weighting negative pairs is chosen as $T_n = 10$. All models are trained with an embedding size of 2048. The selection of data points for positive and negative data set were calculated with a similarity threshold of 0.7. The model was trained on a batch size of 265 for 100 epochs with weights from the pre-trained model provided by Sumbul et al. (2019) [21].

m	α	Precision for Top 10 images
0.5	1.25	52.05%
0.75	1.35	60.11%
1.0	1.5	65.26%
1.25	1.625	61.31%
1.5	1.75	65.2%

Table 5.1: Precision results for different margins and their respective α

With α = 1.5 as the chosen boundary changes of the margin m could be of interest. As seen in table 5.2 the precision does not change considerably compared to the achieved precision of 65.26%. The values of 0.25 and 0.5 for margin m are disregarded, considering that the diameter of a class hypersphere is calculated as $\alpha - m$. With a smaller margin m the diameter of a classes hypersphere increases. Choosing a large diameter for a class hypersphere could create a disadvantage that embedded images are not pulled close enough.

m	α	Precision for Top 10 images
0.25	1.5	65.03%
0.5	1.5	65.27%

Table 5.2: Precision results for different margins and their respective α

Weighting Temperature

From the start, the negative temperature was set to 10, in order to weight the negative pairs and emphasize the distancing of negative data pairs. Since the first model iterations were not trained with the weighting of positive data pairs, the next step was to look at changes created by the positive Temperature. Other parameters are chosen as in section 'Margin m' 5.1 described. An increase of T_p gives more importance to data pairs that violate constraints and the model is able to learn more effectively. Therefore an increase in precision is expected. Since T_p=15 does not improve the models precision, T_p = 10 is chosen for further training. The last row of table 5.3 shows the result for $T_p = 15$ and $T_n = 15$. The lower precision of 61% indicates that chosen data pairs who are heavier weighted, are not all selected properly for the negative and positive set. If that was the case and only data pairs that improve the model are heavier weighted, the precision should not decrease.

T_n	T_p	Precision for Top 10 images
10	5	66.84%
10	10	68.78%
10	15	68.08%
15	15	61%

Table 5.3: Precision results for different weighting temperatures

Similarity

The similarity parameter is essential for the selection of data points for the positive and negative set (Eq. 3.15 and Eq. 3.16). Choosing a lower threshold like 0.5 would lead to data points in the positive set that are only distantly similar to the input data. With 'wrong' positive pairs in the positive set, the objective of pulling real positive data pairs closer would fail. Images that should not be pulled closer are pulled closer. The results from table 5.4 may be due to too few data points that are considered for the positive and negative set. If there are not enough data points the model can learn from the embedding function does not improve.

similarity	m	α	T_p	T_n	Precision for Top 10 images
0.7	1.0	1.5	10	10	68.78%
0.8	1.0	1.5	10	10	66.06%
0.9	1.0	1.5	10	10	57.68%

Table 5.4: Precision results for different similarity

Learning Rate

After training a model on batch size=512 and epochs=50 an increase of the training loss can be observed in Figure 5.1 (orange graph). Such an increase of training loss indicates a learning rate that is not small enough, leading away from an optimized minimum.

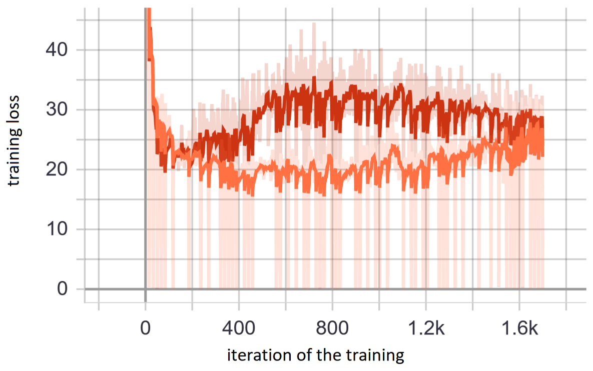


Figure 5.1: Training loss of model with learning rate=0.001 (orange) and lr=0.0001 (red)

This insight leads to a decreased learning rate of 0.0001 with immediately better results, as seen in table 5.5. Judging from the training loss, the loss is still converging and the mapping of the model in the embedding space can still be improved. Since the optimizer Adam handles learning rate better than SGD the following models are trained with Adam.

learning rate	Precision for Top 10 images
0.001	66.19%
0.0001	72.12%

Table 5.5: Precision results for different learning rates using SGD

Optimizer Adam

The optimizer Adam utilizes an adaptive learning rate. Considering the issue of the learning rate, Adam should be able to converge with a starting leaning rate of 0.001 and lower the learning rate as the training progresses. With Adam delivering promising results, an insight in how the model performs without pre-trained weights is shown in table 5.6. The model is trained for 50 epochs on a batch size of 512. That the precision at k=1 for the pre-trained model is lower than precision at k=5 is unexpected. Four out of the eight calculated precisions for each test batch at k=1 scored slightly below the results of precision at k=5. The other four results were higher at k=1 than k=5. This anomaly can be explained by considering that the best result is sometimes wrong and the second and its following results correct. If 0 is a wrong retrieved image and 1 stands for a correct retrieval, then the precision at k=1 for (0,1,1,1,1) is 0%, but a precision at k=5 is 80%. This would explain the lower precision of the pre-trained model for precision at k=1.

Optimizer	pretrained	α	m	T_p	T_n	Precision Top 1	Precision Top 5	Precision Top 10
Adam	yes	1.5	1.0	10	10	73.27%	73.40%	73.11%
Adam	no	1.5	1.0	10	10	59,80%	58.99%	58.56%

Table 5.6: Precision results for models trained with Optimizer Adam

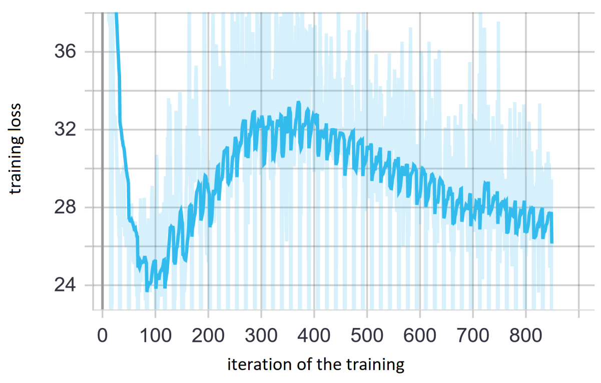


Figure 5.2: Loss for model trained with Adam and its adaptive learning rate on 50 epochs.

Figure 5.2 shows the training loss of the Adam model with pre-trained weights. It can be seen that the loss is still converging, which leads to the conclusion that this model has still room for improvement.

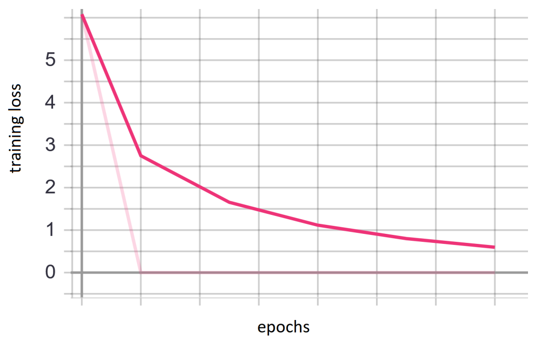


Figure 5.3: Loss for model trained with Adam without pre-trained weights

The lower percentage of the model without pre-trained weight can be investigated by considering the selection of data pairs for the positive and negative set. Since the model has no trained weights the constraints of Eq. 3.15 and Eq. 3.16 may not be violated by the data points leading to sparse or empty sets. Which could result in difficult learning for the model. Similar results to the ones of the pre-trained model might be achieved with a longer training period.

Final model

The final model trained using Adam with a starting learning rate of 0.001. The usual embedding size of 2048 and a batch size of 512 were used. The model is trained for 100 epochs with $\alpha = 1.5$ and $m = 1.0$. The temperatures T_p and T_p are each initialized with the value 10.
The model achieves a precision of 73.16% for k=10.

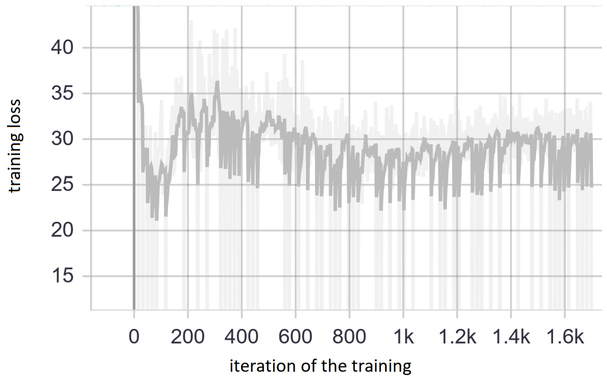


Figure 5.4: Loss for model trained with Adam on 100 epochs

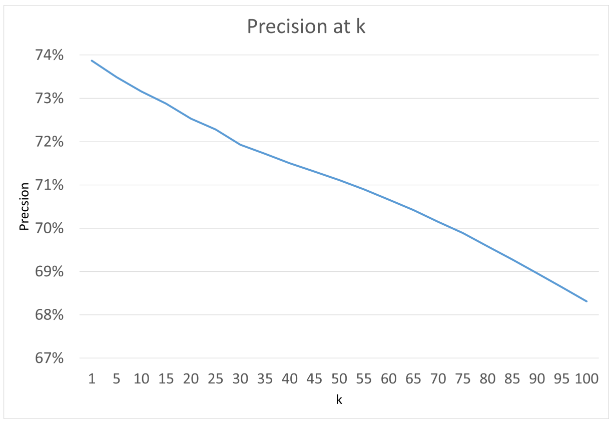


Figure 5.5: Precision at k for model trained with Adam on 100 epochs

Figure 5.5 shows the precision over k until 100, showing a linear descend starting with 73.87% and ending ate k=100 with 68.31% in precision.
Considering the previews trained models a precision of 73% seems to be the upper boundary for these parameters.

6 Conclusion and Future Work

Utilizing the Soft Pairwise Similarity function by Zhang et al. (2020) [27], a multi label implementation of RLL [24] for CBIR in RS was realised. The RLL for multi label in RS implementation was trained to achieve a precision of 73%. The sensitivity analysis shows the impact the parameters have on the precision of the model.

However, trained models had their difficulty further discriminating between different classes to achieve a higher precision. This derives presumably from the selection process for positive and negative data points as described in section 5.1. To further improve this fine line of selecting a threshold that considers labels, which are different but not too dissimilar to the input data, is subject to further study and improvements.

Future work could continue studying the similarity of multi labels and improve the Soft Pairwise Similarity Function. Soft Pairwise Similarity does not discriminate data points by classes that resemble each other, but their position in space. Therefore, an implementation that examines, which classes of labels are equal and hold semantic similar information may further improve the comparison of labels.

Bibliography

[1] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: (2019).

[2] red bike.de. *Blue Bike*.

[3] Emilio Chuvieco and Russell G. Congalton. “Application of remote sensing and geographic information systems to forest fire hazard mapping”. In: *Remote Sensing of Environment* 29.2 (1989), pp. 147–159. ISSN: 0034-4257. DOI: `10.1016/0034-4257(89)90023-0`. URL: `http://www.sciencedirect.com/science/article/pii/0034425789900230`.

[4] H. Daschiel and M. Datcu. “Information mining in remote sensing image archives: system evaluation”. In: *IEEE Transactions on Geoscience and Remote Sensing* 43.1 (2005), pp. 188–199. ISSN: 0196-2892. DOI: `10.1109/TGRS.2004.838374`.

[5] Ritendra Datta, Jia Li, and James Z. Wang. “Content-based image retrieval: approaches and trends of the new age”. In: *Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval*. 2005, pp. 253–262.

[6] Du Peijun et al. “Study on content-based remote sensing image retrieval”. In: *Harmony between man & nature*. Piscataway, NJ: IEEE Operations Center, 2005, pp. 707–710. ISBN: 0-7803-9050-4. DOI: `10.1109/IGARSS.2005.1525204`.

[7] R. D. Hudson and J. W. Hudson. “The military applications of remote sensing by infrared”. In: *Proceedings of the IEEE* 63.1 (1975), pp. 104–128. ISSN: 0018-9219. DOI: `10.1109/PROC.1975.9711`.

[8] Jacob Goldberger et al. “Neighbourhood Components Analysis”. In: 2005, pp. 513–520. URL: `https://papers.nips.cc/paper/2566-neighbourhood-components-analysis`.

[9] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015).

[10] Kihyuk Sohn. “Improved Deep Metric Learning with Multi-class N-pair Loss Objective”. In: 2016, pp. 1857–1865. URL: `http://papers.nips.cc/paper/6200-improved-deep-metric-learning-with-multiclass`.

[11] Sungyeon Kim et al. “Deep Metric Learning Beyond Binary Supervision”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[12] Diederik P. Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[13] Yansheng Li et al. "Content-Based High-Resolution Remote Sensing Image Retrieval via Unsupervised Feature Learning and Collaborative Affinity Metric Fusion". In: *Remote Sensing* 8.9 (2016), p. 709. DOI: `10.3390/rs8090709`. URL: `https://www.mdpi.com/2072-4292/8/9/709/htm`.

[14] Yair Movshovitz-Attias et al. "No Fuss Distance Metric Learning using Proxies". In: (). URL: `https://arxiv.org/pdf/1703.07464`.

[15] David J. Mulla. "Twenty five years of remote sensing in precision agriculture: Key advances and remaining knowledge gaps". In: *Biosystems Engineering* 114.4 (2013), pp. 358–371. ISSN: 1537-5110. DOI: `10.1016/j.biosystemseng.2012.08.009`. URL: `http://www.sciencedirect.com/science/article/pii/S1537511012001419`.

[16] Maik Netzband, William L. Stefanov, and Charles L. Redman. "Remote Sensing as a Tool for Urban Planning and Sustainability". In: (), pp. 1–23. DOI: `10.1007/978-3-540-68009-3{\textunderscore}1`.

[17] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "FaceNet: A Unified Embedding for Face Recognition and Clustering". In: (2015), pp. 815–823. DOI: `10.1109/CVPR.2015.7298682`. URL: `https://arxiv.org/pdf/1503.03832`.

[18] Santhosh K. Seelan et al. "Remote sensing applications for precision agriculture: A learning community approach". In: *Remote Sensing of Environment* 88.1-2 (2003), pp. 157–169. ISSN: 0034-4257. DOI: `10.1016/j.rse.2003.04.007`. URL: `http://www.sciencedirect.com/science/article/pii/S0034425703002360`.

[19] Hyun Oh Song et al. "Deep Metric Learning via Lifted Structured Feature Embedding". In: (). DOI: `10.1109/cvpr.2016.434`.

[20] Senduru Srinivasulu and P. Sakthivel. "Extracting spatial semantics in association rules for weather forecasting image". In: *Trendz in Information Sciences & Computing(TISC2010)*. IEEE, 12/17/2010 - 12/19/2010, pp. 54–57. ISBN: 978-1-4244-9007-3. DOI: `10.1109/TISC.2010.5714608`.

[21] Gencer Sumbul et al. "Bigearthnet: A Large-Scale Benchmark Archive for Remote Sensing Image Understanding". In: *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*. 2019, pp. 5901–5904. ISBN: 2153-7003. DOI: `10.1109/IGARSS.2019.8900532`.

[22] F. Sunar and C. Özkan. "Forest fire analysis with remote sensing data". In: *International Journal of Remote Sensing* 22.12 (2001), pp. 2265–2277. ISSN: 0143-1161. DOI: `10.1080/01431160118510`.

[23] Jiang Wang et al. "Learning fine-grained image similarity with deep ranking". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1386–1393.

[24] Xinshao Wang et al. "Ranked List Loss for Deep Metric Learning". In: (). URL: `https://arxiv.org/pdf/1903.03238`.

[25] Kilian Q. Weinberger and Lawrence K. Saul. “Distance Metric Learning for Large Margin Nearest Neighbor Classification”. In: *J. Mach. Learn. Res.* 10 (2009), pp. 207–244. ISSN: 1532-4435.

[26] Chao-Yuan Wu et al. “Sampling Matters in Deep Embedding Learning”. In: (). URL: `https://arxiv.org/pdf/1706.07567`.

[27] Z. Zhang et al. “Improved Deep Hashing With Soft Pairwise Similarity for Multi-Label Image Retrieval”. In: *IEEE Transactions on Multimedia* 22.2 (2020), pp. 540–553. DOI: `10.1109/TMM.2019.2929957`.

[28] Shizhou Zhang et al. “Person Re-Identification With Triplet Focal Loss”. In: *IEEE Access* 6 (2018), pp. 78092–78099. ISSN: 2169-3536. DOI: `10.1109/ACCESS.2018.2884743`.

[29] Wengang Zhou, Houqiang Li, and Qi Tian. “Recent Advance in Content-based Image Retrieval: A Literature Survey”. In: (). URL: `https://arxiv.org/pdf/1706.06064`.

Appendix

The code used in the experiments can be found on the GitLab of the TU Berlin, `https://gitlab.tubit.tu-berlin.de/rsim/ranked-list-loss`.