

Author / Eingereicht von
Thomas Lintner
11703785

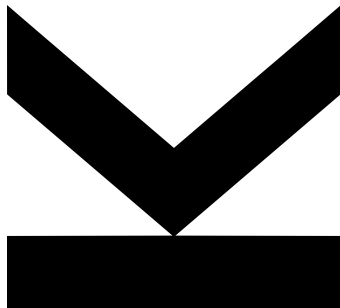
Submission / Angefertigt am
Institute of
Computational
Perception

Thesis Supervisor /
Evaluator / Erstbetreuer /
Beurteiler
DI Dr. Univ.-Prof. Dr.
Gerhard Widmer

Co-Supervisor /
Mitbetreuung
Dr. Arthur Flexer

June 2022

Semantic and Visual Organization of a Fine Arts Museum Collection



Master Thesis

to obtain the academic degree of

Diplom-Ingenieur

in the Master's Program

Computer Science

Kurzfassung

In dieser Masterarbeit werden Kunstwerke der Belvedere Kunstsammlung miteinander in Beziehung gestellt indem Informationen, welche von Textquellen sowie Bildern extrahiert wurden mit Hilfe von künstlicher Intelligenz verarbeitet werden. Darauf aufbauend wurden Algorithmen erstellt die verschiedene auf Ähnlichkeiten bezogene Aufgaben erfüllen. So können diese zum Beispiel semantische Pfade durch die Kunstsammlung erstellen, Vorschläge für unbesetzte Stellen in einer Serie von Kunstwerken liefern oder aber auch die Kunstsammlung anhand von Ähnlichkeiten zu einem bestimmten Kunstwerk oder Begriff sortieren bzw. filtern. Im Zuge dieser Arbeit wurden neurale Netzwerke eingesetzt um Objekte in Kunstwerken zu entdecken und um Informationen aus textuellen Quellen, wie zum Beispiel den Annotationen durch Kunstexperten, zu extrahieren. Ein Fokus lag weiters auf dem Erstellen von Metriken zur Evaluierung einzelner Resultate, aber auch zum quantitativen Überprüfen des Gesamterfolgs dieser Arbeit.

Abstract

In this thesis artworks of the Belvedere Collection are brought into context with each other by processing information extracted from textual as well as visual elements with the use of artificial intelligence. This is used as the foundation of various similarity-related tasks, such as the generation of semantic paths through the art collection, the filling of empty spots in a series of artworks or simply sorting or filtering artworks based on their similarity to a given artwork or search term. In the course of this work neural networks were used for detecting objects in artworks as well as for extracting information from textual sources like the annotations of art experts. Lastly, the creation of metrics for evaluating individual results as well as quantitatively determining the success of this thesis was of great importance.

Contents

1. Introduction	1
2. Related Works	3
2.1. Language Processing	3
2.1.1. Embeddings	3
2.1.2. Word2Vec	4
2.1.3. fastText	6
2.1.4. Implementation Details	8
2.2. Problems with Text Tags	8
2.3. Image Processing	9
2.3.1. Research	10
2.3.2. Approach for this thesis	13
3. The Data	15
3.1. Relevant Columns in the Dataset	15
3.2. Extraction of Tags	21
3.2.1. Lemmatization	23
3.2.2. Out of Vocabulary Words	23
3.3. Word Splitting	24
3.3.1. Statistics and Selection	35
3.4. Countering Bias towards longer Tags	37
3.5. Box Annotation Dataset	40
3.6. Tree and Person Annotation Dataset	41
4. Implementation	42
4.1. Similarity Calculations and Caching	42
4.1.1. General Process of calculating Similarities	42
4.1.2. Normalization of Similarities	42
4.1.3. Framework	45
4.2. Filter Cache	51
4.3. Path-Generation Algorithm	51
4.3.1. Prior work on Path-Generation	51
4.3.2. The Path-Generation Algorithm for this Thesis	52
4.3.3. Alternative Path-Generation Algorithm	54

Contents

4.4.	Algorithm for Filling empty Spots	56
4.5.	Algorithms for the Extraction of Objects from Artworks	57
4.5.1.	Enhancing Images	58
4.5.2.	Modelclass, Characteristics and Training	59
4.5.3.	Data Preparation and Augmentations	60
4.5.4.	Making predictions	63
4.6.	Evaluation of Object Detection	68
5.	Results	71
5.1.	Path-Generation	71
5.1.1.	Semantic Matching	71
5.1.2.	Visual Matching	74
5.1.3.	Semantic and Visual Matching combined	75
5.2.	Using the Iconclass System for creating a Measure of Success	78
5.3.	Quantitative Analysis	82
6.	User Interface	87
7.	Outlook	97
8.	Conclusion	99
A.	Appendix	100
A.1.	Acknowledgements	100
A.2.	Disclaimer	100
A.3.	License of Data	100
A.4.	Acknowledgements	101

List of Figures

2.1. Example of the advantage of object recognition	14
3.1. Example of iconclass hierarchy (as seen on http://www.iconclass.org , last visited on 03. Dec. 2021) illustrated by the author	16
3.2. Number of times a unique value from the column “Object Class” is assigned to an artwork	19
3.3. Number of times a unique value from the column “Year Estimate” is assigned to an artwork	19
3.4. Number of times a unique value from the column “Year Estimate” is assigned to an artwork, sorted by year	20
3.5. Number of times a unique value from the column “Material Technique” is assigned to an artwork	20
3.6. Simplified overview of the word splitting algorithm	28
3.7. Word Splitting: Clarification for step “Find words”	29
3.8. Word Splitting: Step of changing ending and case of words	30
3.9. Word Splitting: Letter substitution	30
3.10. Word Splitting: Handling the problem of not knowing whether a letter is part of the current or next word	31
3.11. Word Splitting: Words and Word-endings	32
3.12. Word Splitting: Recombine words step	33
3.13. Word Splitting: Recombine words special considerations	34
3.14. Number of artworks with specific number of tags, considering tags extracted from Descriptions	35
3.15. Number of artworks with specific number of tags, considering tags extracted from Titles	35
3.16. Number of artworks with specific number of tags, considering tags extracted from Expert Annotations	35
3.17. Number of artworks with specific number of tags, considering tags extracted from Titles and Expert Annotations	35
3.18. Number of times a specific tag extracted from image titles occurs	37
3.19. Number of times a specific tag extracted from image expert tags occurs	37
4.1. Calculation of Image Similarity	44
4.2. AutoSimilarityCache - Required Methods and Interface Methods	46

List of Figures

4.3. AutoSimilarityCache - Caching Scheme	49
4.4. Python algorithm for calculation of shifting weight pairs	53
4.5. Pseudo Algorithm of alternative path generation algorithm	56
4.6. Artwork “Alm mit Rindern” by Joseph Heicke in its original form.	58
4.7. Artwork “Alm mit Rindern” by Joseph Heicke in a processed form.	58
4.8. Importance of Augmentations	62
4.9. Bounding boxes filtered with nms=0	63
4.10. Bounding boxes filtered with nms=0.5	64
4.11. Unchanged Coco Model Confusion Matrix for predictions of persons with ideal parameters	66
4.12. Confusion Matrix: Tree Test Dataset, No Augmentations	67
4.13. Confusion Matrix: Person Test Dataset, No Augmentations	67
4.14. Confusion Matrix: Tree Test Dataset, with Augmentations	67
4.15. Confusion Matrix: Person Test Dataset, With Augmentations	67
4.16. Confusion Matrix: Tree Test Dataset, No Augmentations; Ideal threshold values	67
4.17. Confusion Matrix: Person Test Dataset, No Augmentations; Ideal threshold values	67
4.18. Confusion Matrix: Tree Test Dataset, with Augmentations; Ideal threshold values	68
4.19. Confusion Matrix: Person Test Dataset, With Augmentations; Ideal thresh- old values	68
4.20. F1 scores of model trained with x percent of dataset - Tree Test Dataset . .	70
4.21. F1 scores of model trained with x percent of dataset - Person Test Dataset .	70
5.1. Example 1 of a Path resulting from semantic Matching only	72
5.2. Example 2 of a Path resulting from semantic Matching only	72
5.3. Detected Objects for the artwork “Der Karrengaul” by Emanuel Franz Hegenbarth; Predictions made with Augmented model, nms=0.1, score threshold=0.25	74
5.4. Example 1 of a Path resulting from semantic and visual Matching	76
5.5. Example 2 of a Path resulting from semantic and visual Matching	76
5.6. Iconclass comparison using text labels extracted from expert tags and title tags	81
5.7. Illustrative example of the metric: Ideally the order would be strictly as- cending, the actual result however can be different.	83
5.8. An algorithm calculates how many times neighboring elements have to be swapped in order to transform the actual result into the ideal one.	84
5.9. Relative Difference of Errors for Paths of Length x in Percent	85
5.10. Quantitative Analysis of Generating Paths of different lengths	86
6.1. The start menu of the User Interface	89

List of Figures

6.2.	The info menu	90
6.3.	The Filter Menu	91
6.4.	View after selecting the artwork “Abend am Mittenwaldsee” by Mathilde Sitta-Allé in the “Show most similar artworks” window	92
6.5.	Entry mask for “Create Semantic Path” window	93
6.6.	Results of a semantic path in the GUI	93
6.7.	Entry mask 1 of “Fill Spots” window	94
6.8.	Entry mask 2 of “Fill Spots” window	94
6.9.	Example input of “Fill Spots” in the GUI (only a part of the input is visible)	95
6.10.	Example results of “Fill Spots” in the GUI	96

List of Tables

3.1. Tags assigned by the language model given the title of a chosen artwork as input	21
3.2. Tags assigned by the language model given the title of a chosen artwork used in a sentence as input	22
3.3. Examples of results of the word splitting algorithm	34
4.1. Percentages of positive samples in datasets and splits for the classes “Baum” (=tree) and “Person” (=person)	60
4.2. Percentages of positive samples in the folds of the multilabel stratified 3 fold cross validation used for training and validation for the classes “Baum” (=tree) and “Person” (=person)	60
4.3. Ideal values for non maximum suppression and score thresholds in object predictions	65
4.4. “Confusion Matrices” of iconclass annotations and final object labels . . .	69

1. Introduction

Data science has seen giant leaps in the last decades and such it is no wonder that it finds applications in the field of art as well. Collections of art are bound by physical aspects no longer, as collections too can now be digital. Without such limitations, their size can quickly grow into the thousands - too much for any person to keep track of. Therefore tasks such as finding the perfect artwork for a specific slot in an exhibition become an increasingly difficult process. Considering said slot however, it stands to reason, that the artworks that would be considered for filling it, are either in a contextual or visual proximity to the images around it. For this reason, it would be enriching for the curators to have a tool at their disposal that creates a preselection according to these aspects as well as on other input. In order to enable a creative “flow” and easy experimentation, the program must produce results with very little response time. It must also be flexible enough to allow various inputs from the user. Furthermore, the large amount of unique labels (=tags) that must be handled is quite challenging. Almost 4000 unique tags were extracted from the titles and almost 3400 unique tags were extracted from expert annotations. Since there is no overlap between those two sets, this totals to about 7350 unique tags. The reason why specifically the numbers of **unique** tags is mentioned, is that these will drive the complexity up in later processing.

Through adapting the algorithms created in this thesis, it would be feasible to generate art exhibitions automatically. However, the beauty in art lies in the more subtle aspects that cannot be expressed by an algorithm or formula. If there was a way to achieve this creativity and depth with a computer program, one would also have to ask the more philosophical question of whether the value of an artwork is given by the characteristics of the resulting object alone or whether its value is also enriched by the story of the artist, the time in which the artist lived, etc. Therefore the ambitions of this project stop short of writing a program that by itself takes care of creating an entire exhibition on its own.

1. Introduction

The aim for this project instead is to provide the curators responsible for creating the exhibitions with a useful tool.

In the course of this thesis neural networks will be used for detecting objects in artworks as well as for extracting information from textual sources. In the end metrics will be created that allow to evaluate the processes introduced in this work. Special consideration is also given to the cleaning and enhancement of the data.

2. Related Works

2.1. Language Processing

The most important operation in the application is finding artworks that are similar. In a first step that means deciding whether the words describing a pair of artworks are similar. For a human this task would be trivial as it is obvious that e.g. the word “cow” fits the word “alps” better than the word “chair”. If a human is asked to put these similarities into numbers however, it would be a far more difficult task. Nevertheless this is what needs to be done in order to being able to calculate with similarities in a computer program. Luckily, this does not have to involve careful considerations by a human and in this chapter it will be explained how it can be done by computers instead.

2.1.1. Embeddings

Since words are made of letters and these letters have arbitrary numbers assigned to them based on an encoding, calculating with these would not be very meaningful. Many natural language processing systems of the past therefore assigned simple indexes to the words, which has the problem of losing any notion of similarity between the words [1]. Modern systems more commonly use a vector of numbers to represent each word instead. Every word is then represented as e.g. 300 numbers that can be positive or negative. With these it is then relatively simple to calculate: The similarity of two words is just the cosine distance between the two vectors. If it is low, it means that the representations of the words are close to each other in the embedding space. It follows that they are to be considered similar. This leads to the much more concrete question of how to find the numbers for said vector, which will be answered in the next sections.

2. Related Works

2.1.2. Word2Vec

This section focuses on the works of Mikolov et al. [1] and Mikolov et al. [2] in which the models were introduced and improved upon.

The used network is based on a so called “Feedforward Neural Net Language Model” (NNLM). This NNLM takes a 1-of-(Size of the Vocabulary) coding of a fixed number of previous words as input. Then the input passes through a projection layer into a hidden layer, which computes the probability distribution over all words. Since the non-linear hidden layer is quite complex, the authors of [1] propose a new architecture that shares the projection layer for all words and therefore does not need the non-linear hidden layer. Finally, a hierarchical softmax is used as output. It is implemented as a binary Huffman tree, since this provides two times speed up during evaluation (with vocabulary size = 1 million) [1] and also a speedup in training [2] by assigning short binary codes to frequent words [1].

The authors [1] trained this architecture in two different ways resulting in two different Models:

- Continuous-Bag-of-Words Model (CBOW)
 - Goal: Learn to predict one word based on a set containing a given number of surrounding words
- Continuous Skip-gram Model
 - Goal: Learn to predict words within a certain range of an input word within the same sentence

In [1] it was found that the accuracy of the word vectors, which are learned by neural networks, depend on their dimensionality and on the amount of training data. Through the above described approach it was possible to create word vectors of significantly greater length [1] and in Mikolov et al. [2] it is stated for the Skip-gram model that it is possible to use significantly more data to train on. This is attributed to the low time complexity of the model.¹

¹Note of the author: Since the CBOW model is similar to the Skip-gram model, the last statement likely applies to that model as well.

2. Related Works

Characteristics of resulting Word Vectors

The resulting word vectors can be used for answering questions, as illustrated by these examples taken from the aforementioned papers [1, 2]:

- The word that is similar to “small” in the same sense as “biggest” is similar to “big” is found by looking for the most similar vector to the result of performing the following arithmetic operations on the learned vector representations of the following words [1]:

$$\text{“biggest”} - \text{“big”} + \text{“small”}$$

- The capital of Germany, which is Berlin, is found by looking for the most similar vector to the result of performing the following arithmetic operations on the learned vector representations of the following words [2]:

$$\text{“Germany”} + \text{“capital”}$$

Although this functionality is not used explicitly in this thesis, it is very important to the success of this work. An example of where this will be important is when comparing the titles of two artworks with the titles “Painting of the capital of Germany” and “Painting of Berlin”². Here the result should be that these two artworks are considered to be almost identical based on their titles.

Another feature of these vectors that should come in handy is that the individual vectors representing words can be “somewhat meaningfully combined using just simple vector addition”[2]. This is vital since the pieces of text that are processed in this thesis are mostly longer than just a single word.

Further improvements

In this section only a few chosen improvements made in [2] are shortly discussed.

In [2] the authors worked out a way to find phrases that consist of multiple words and represent them with their own vector instead of being represented by simply combining the meanings of the words they are made up from.

²Not actual examples of keywords from the dataset

2. Related Works

Another addition that was made is to subsample words, such that the most frequent words (e.g. articles) are often discarded, which improved the accuracy of the vector representations especially for rare words and lead to a very significant speedup during training [2].

2.1.3. fastText

This section focuses in large parts on the paper from Bojanowski et al. [3] in which the fastText model was first introduced.

Base Model

In section 2.1.2 the Word2Vec model was described on which the model in this section builds upon. Specifically, the Continuous Skip-gram model is used together with negative sampling and subsampling of frequent words³. A characteristic that might leave room for improvement in the Word2Vec model is that it assigns a vector to each word while completely ignoring its morphology [3]. For example, it has no influence that the word “railroad” is made up from the two words “rail” and “road”.

Modeling morphology

In the “fastText” model, words are assigned their vectors based on a set of character n-grams.

A so called “n-gram” is just a piece of a specific length that can be extracted from a word. In the paper [3] the n-grams of length 3 of the word “<where>” are given as an example. Note that boundary symbols were added to denote the start and end of a word. This is necessary to distinguish words from n-grams - The representation of the word “her” (→ “<her>”) must be different from the vector representation of the n-gram “her”⁴. Going back to the example, the n-grams of length 3 are: “<wh”, “whe”, “her”, “ere”, “re>”. For the actual implementation the authors use all n-grams that are of sizes $3 \leq n \leq 6$ and add

³see section 2.1.2

⁴Example from the paper [3]

2. Related Works

the word itself as a “special sequence”. The actual result therefore includes the sequence “<where>”.

Summing up the representations of the elements of this set yields the representation of the word they make up. This means that even for words that were never seen before a vector can be computed. While this feature is certainly useful in most cases, it is not desirable in this thesis. This discussion however will be postponed to section 3.2.2.

Using information of the characters that make up a word also means that infrequent words can be modeled better [3].

fastText for the German Language

It is easy to see that a language model does not work equally well for all languages. Consequently, the amount of improvement that the fastText language model brings is different for every language as well.

The authors [3] observe that using n-grams is more important for some languages than it is for others. This is true for German, which is the relevant language for this thesis. One reason for this is that there are many compound words in German and the units of which they are made up from can be captured by the n-grams [3]. For German long n-grams are required [3]. Another reason is that German has 4 cases [3]. This means that there are many words, that essentially mean the same thing, but are written only a little bit differently.

The authors [3] also highlight the improvement for the German language in their model over the baseline. This is despite the fact that in the task of answering semantic questions, the performance degraded as also shown in [3].

Interestingly, the experiments in Grave et al. [4] came to the conclusion that using data from crawling the web⁵, does not increase the performance for the German language. They attribute this partly to the dataset that was used for testing. However, they also note that using the data from the crawl leads to a higher coverage which can lead to better performance in other fields in which a language model might be applied in.

⁵In addition to using Data from Wikipedia

2. Related Works

2.1.4. Implementation Details

Because of the reasons stated in 2.1.3 word vectors from a fastText model are used for this thesis. Specifically, the fastText vectors that are contained in the model called “de_core_news_lg”⁶ from the library “spacy” are used.

Although Bojanowski et al. [3] recommend retraining the model on textual data relevant to the application, this cannot be done for this thesis, since the data used for this thesis consists of mostly text fragments.

2.2. Problems with Text Tags

The same problems were encountered in this thesis as were described by Flexer [5]:

- There are some keywords describing pictorial organization, that do not add any context
 - e.g. “Dreifigurig” (=“three-figured”), which only means that there are three figures seen in the artwork, but there is no added context like who or what these figures are etc.
 - These tags cannot be filtered effectively, however their influence can be reduced by increasing the number of tags (generated using text or visual information)
- There are some tags that are very general
 - e.g. “Landschaft” (=“landscape”)
 - This problem appears to be reduced by taking into account the expected similarities when calculating the similarities of pairs, as is shown in section 4.1 and by the matching according to equation 4.1.
- There are some keywords which lack semantic context

⁶https://spacy.io/models/de#de_core_news_lg

2. Related Works

- e.g. the word “Akt”, which, in the context of art, is the German word for “nude painting”, but much more commonly it is used as the German word for “file”. It is reasonable to believe that a context insensitive language model, which is a language model that maps 1 word to exactly 1 representation, recognizes the letter combination “Akt” as the latter.
- A solution could be using context sensitive language models. These are language models that assign a vector to a word based on the context it appears in. This means that the same word can have multiple different vector representations. However, [6] suggests to have whole sentences or paragraphs as input for these networks⁷. As can be seen in section 3.2 there are problems caused by incomplete sentences even with the context insensitive language model, consequently it is reasonable to assume that using a context sensitive language models would not bring a benefit to this application. It would even have the disadvantage that word vectors could not be cached, since words would not map to the same vector every time. The language model would also have to be loaded constantly to generate a new vector every time one is needed, which would also add a lot of processing time and memory usage. Therefore, the decision was made against using such a model. It is worth noting that a BERT network is used in Banar et al. [7] who created a program to automatically assign Iconclass⁸ codes to artworks. Specifically, the multi-lingual version of the BERT network was used in that work, since they process text data from multiple languages.

2.3. Image Processing

In sections 5.1.1 and 5.3 it will be seen that the matching based on the extracted text tags alone gives very satisfactory results. It has to be kept in mind however, that, as also described in those sections, the minimum requirement to consider an artwork as a potential match was that the artwork has at least a specific number of sources of tags that are considered in the matching process. This measure was taken so that the hypothesis whether an artwork is similar to another or not is based on more than one indicator alone

⁷specifically the source says this is about BERT networks

⁸This is a labeling system that will be discussed in section 3.1

2. Related Works

and consequently to increase the probability of correctness. However, this came at the cost of discarding a significant number of artworks straight away. It is therefore necessary to gather more data on the artworks. Since all textual data was already considered, this leaves the images associated to the artworks themselves.

In Carneiro et al. [8], the term “artistic image understanding” is defined as “a process that receives an artistic image and outputs a set of global, local and pose annotations” [8]. While in this thesis pose annotations would likely not be very beneficial, ideally the global (“[...] set of artistic keywords describing the contents of the image”[8]) and local (“[...] set of bounding boxes that localize certain visual classes [...]”) annotations are produced. While the former is covered by processing the text fragments extracted from the dataset, the latter requires analysis of the images themselves. In other words, processing the contents of the images themselves is a necessary step in order to achieve this “artistic image understanding” and therefore should be done in any case.

A problem with the dataset at hand is that some images only show the actual artworks (e.g. oil paintings), but others are photographs of the artworks (e.g. for statues). In the cases of photographed artworks the actual artworks are in the center of the images and they are surrounded by some background. The performance of the model could therefore vary between the different material techniques. One apparent approach would be to train two models: One for photographs with background and another for paintings, drawings, etc. However, there are too few photographed artworks in the dataset to meaningfully train or fine-tune a separate neural network.

2.3.1. Research

There have been several attempts to use the visual information in the artworks in order to label images.

Frome et al.

Frome et al. [9] have implemented a system that focuses on zero-shot prediction of labels that were not seen by the visual model before. They use labeled image data and information extracted from unannotated text and learned word embeddings from

2. Related Works

wikipedia documents by using a skip-gram model (see section 2.1.2). In addition, they also retrained a network for object recognition to predict the word embeddings that belong to an image. The visual model is trained on the dataset for the 1000-class ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 and employs the architecture that won the challenge in 2012. An interesting detail is that Frome et al. [9] remove the Softmax layer of the visual model and use a projection layer to predict the word embeddings generated by the language model.

In their experiments the authors test their approach in two different ways:

- “Hit@k” which is a metric based on the correct result being in the top k predictions
- “Hierarchical precision@k” which is a metric based on the ImageNet label hierarchy. The authors describe it as follows: “[This] metric will give the model credit for predicting labels that are in the neighborhood of the correct label in the ImageNet hierarchy (for $k > 1$)”[9].

The results are that although the baseline model trained with a traditional softmax layer performs better according to the “hit@k” metric, the proposed model wins according to the “hierarchical precision@k” metric. The authors interpret this as the models ability of “leveraging the semantic knowledge captured by the language model to make reasonable predictions.”[9]

Finally the proposed model significantly outperforms the softmax baseline model in the task of predicting labels for classes that it has never before visually observed.

Kim et al.

While Frome et al. [9] trained the visual model to predict the vector representation of the words describing an image, Kim et al. [10] use the visual model to find co-occurrence patterns. The authors use an Inception-V3 network, pretrained on ImageNet and fine-tune the last fully connected layer, which they adapt to output a probability mass function over 2048 words, extracted from the titles of the artworks. The network then learns the “associative patterns between the output words”[10].

The authors validated their results with a small survey among art students and also two different measures. While the survey indicated that the approach works, the authors

2. Related Works

recognize that the system has “limitations regarding certain types of paintings especially in complex depictions or compositions.”[10]

Milani et al.

Milani et al. [11] describe three applications: semi-automatic labeling of images, different clusterings of artworks and detecting the regions where the core elements of iconography class are to be found which can be meaningful for further analysis.

They apply a convolution neural network classifier, more specifically a ResNet50, which is pretrained on the ImageNet dataset, whereas the last two layers have been changed to a 1 x 1 convolution layer. This last layer has one channel per class and is used as the classifier. All the layers except for the first few are retrained.

The authors [11] also note a problem that frequently co-occurring classes may cause some confusion in the network. Another interesting part of this paper is that the authors use Class Activation Maps to visualize the parts of images that were important for a label assignment.

Experiments showed that the classes were predicted with a mean average precision per class of 72.73%, which is a very respectable result. However, in this work only 10 different iconclass labels were considered.

Banar et al.

The authors [7] note that “textual features strongly outperform visual features”, but also note that the visual features can still be beneficial. They note that a CNN architecture might “devote too much of its representational capacity to image fragments containing redundant information” [7] and thus they employ a different strategy: First a Faster R-CNN is used to extract the region of interest in the form of bounding boxes and detect and classify objects within them. A position-wise fully connected layer then combines them into a feature matrix, that is then passed to the self-attention layer. This layer encodes relationships between the regions, which the authors see as very important, given that there is no fixed order of the image regions.

2. Related Works

An interesting aspect of their multi-modal⁹ approach is that they combine word embeddings and the vectors obtained from the visual information by concatenating them and then downsizing the obtained vector again by using a fully connected layer.

The results show that while assigning Iconclass codes based on visual features alone is not promising, the best performance can be achieved by using textual and visual features together.

2.3.2. Approach for this thesis

The most promising strategy of extracting the context from images seems to be to use object detection like in Banar et al. [7]. The reason for this is illustrated best with an example: Assume that the image shown in figure 2.1 is assigned the label “triangle”. If the visual processing pipeline is given the whole image as input then the neural network will have difficulties to make out that the reason for this assignment is the two areas in which the triangles are shown. If the artist has drawn multiple similar artworks and “triangle”-tags appear in most of them, the network could learn to associate the style of this artist with the label “triangle”, which would be wrong. Another reason for using object detection is that it is to be expected that object detection will do a fairly good job on the artworks in the used dataset, as most of the art is rather realistic looking.

Therefore, a pretrained neural network for object detection and recognition will be used in this thesis. The objects that are found on an image are then used as text labels. This is an alternative to a joint embedding space and makes use of the rich context that the language model has learned from a very significant number of resources. This procedure should help to counteract the problem that image recognition alone cannot understand cultural meaning behind image elements, which Flexer recognizes in [12].

⁹meaning that images and texts are used together

2. Related Works

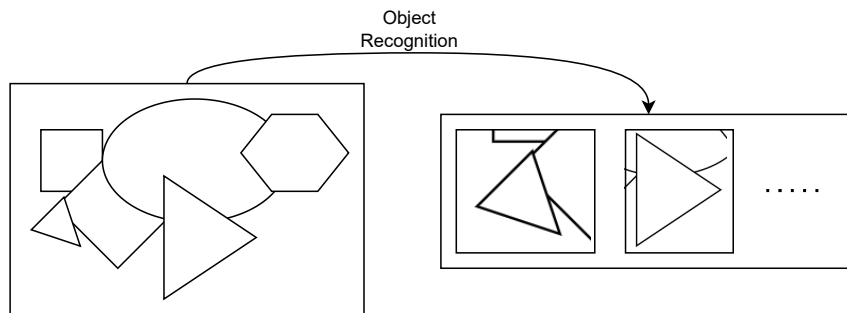


Figure 2.1.: Example of the advantage of object recognition

3. The Data

For this thesis a dataset of the art collection of the Belvedere museum in Vienna, Austria was used. It was provided by Dr. Arthur Flexer in the course of the “Dust and Data”¹⁰ project.

The artworks were downloaded using the links provided in the dataset, but not every artwork was still available. The 240 database entries for which no artworks were available, were discarded, leaving 4135 rows.

3.1. Relevant Columns in the Dataset

The most interesting columns of the dataset for this project are the title column, containing the titles of the artwork and the expert-tags column, which contains labels based on classification systems like “Iconclass”¹¹. As the name suggests, the latter were assigned to the specific artworks by art experts.

Iconclass is a system that is used by experts to label artworks. It has 10 main categories, each spanning a tree structure, as implied by figure 3.1.

¹⁰<http://www.dustanddata.at/>

¹¹<http://www.iconclass.org/help/outline>

3. The Data

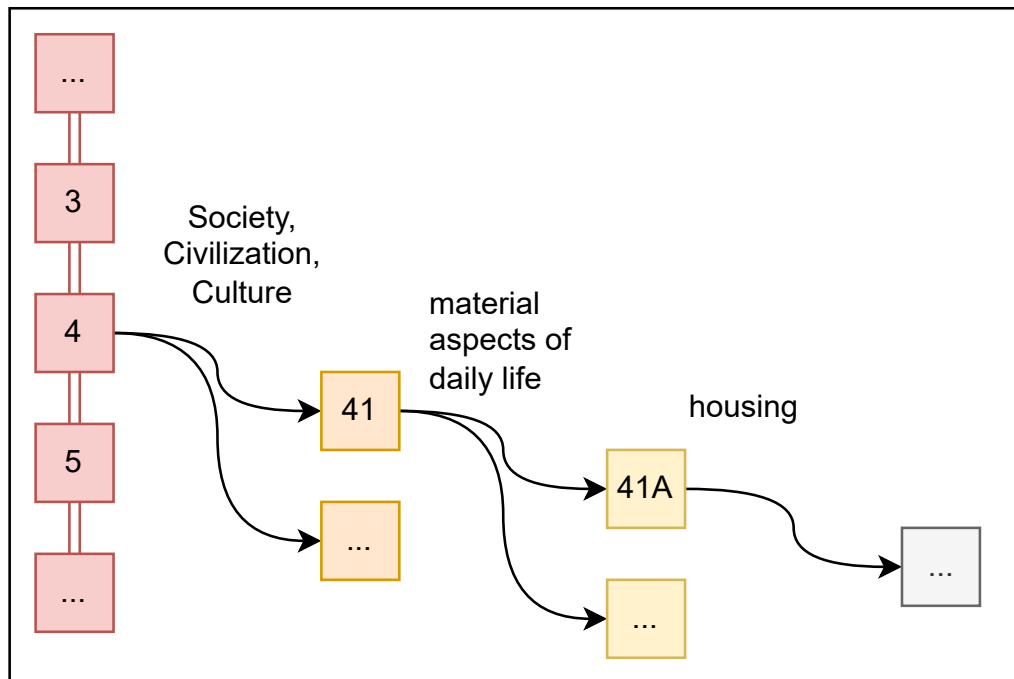


Figure 3.1.: Example of iconclass hierarchy (as seen on <http://www.iconclass.org>, last visited on 03. Dec. 2021) illustrated by the author

Other interesting columns are the “Year” column, which contains the year an artwork is created in, the “Material Technique” column that contains the material the artwork is made with and the “Object Class” column, containing the type of artwork. Examples of the columns are given below.

Examples of entries of column:

Material Technique:

- | | |
|---|--|
| - Öl auf Karton (= oil on cardboard) | - Terrakotta (= terracotta) |
| - Öl auf Leinwand (= oil on canvas) | |
| - Öl auf Holz (= oil on wood) | - Messer aus Holz (= knife made from wood) |
| - Öl auf Leinwand auf Karton (= oil on canvas on cardboard) | |
- ⇒ 519 unique values in total

3. The Data

Object Class:

- | | |
|--|---|
| - <i>Mappenwerk</i> (= portfolio) | - <i>Buchseite</i> (= page from a book) |
| - <i>Gemälde</i> (= painting) | |
| - <i>Zeichnung</i> (= drawing) | - <i>Kopfbüste</i> (= head bust) |
| - <i>Skulpturengruppe</i> (2-figurig) (= Group of sculptures with 2 figures) | - <i>Statuette</i> (= statuette) |
| - <i>Skulpturengruppe</i> (3-figurig) (= Group of sculptures with 3 figures) | ⇒ 91 unique values in total |

Year:

- | | |
|---|---|
| - <i>undatiert</i> (= undated) | - 1909-1912, 1924 <i>überarbeitet</i> (= 1909-1912, revised 1924) |
| - 1786 (?) | |
| - 1916/17 | - <i>spätestens</i> 1759 (= 1759 at the latest) |
| - 2. Hälfte des 19. Jahrhunderts (= 2 nd half of the 19 th century) | - <i>wohl</i> 1702 oder bald danach (= probably 1702 or soon after) |

The "Year" column is in a quite unusable state, therefore a parser was written that converts, for example the value "2. Hälfte des 19. Jahrhunderts" to 1875, which equals to the average of the second half of that century. Consequently only integers are then still left. The column also contains a lot of missing (= "undatiert") values, which were substituted by using the epoch the artwork was created in and substituting missing values by what we estimated to be the average year of the epoch based on the Wikipedia articles concerning the respective epoch. Since the epochs of Renaissance, Baroque and Romanesque art are quite long, the year estimates are likely quite a bit off. Hence, if the artists are known, the average year of the lifetime of the artist (which is gathered from the Belvedere Website <https://sammlung.belvedere.at/>) is used instead. This information is known for 1 row of Romanesque art where the year was still missing and 7 rows for the epoch of Baroque. For those values that are missing the year and the epoch, the average year of the lifetime of the artist is used as well. The rows that are then still missing can be associated to an epoch

3. The Data

again by considering the collections the artworks are a part of. The results are then saved in the column "YearEstimate", which is visualized in figure 3.3. The substituted years might be quite inaccurate, however, a better prediction would require a more profound knowledge of art history. This is not a problem however, since the values are only relevant for figures 3.3 and 3.4.

In figure 3.2 it can be seen how many unique values exist that appear x times in the dataset, where x is one of the values represented by the x-axis. This figure shows that most values appear rarely while a very significant fraction of the art collection is from the class "Gemälde" (= painting), followed by the classes "Zeichnung" (= drawing) and "Druck" (= print). In similar fashion figure 3.3 shows the distribution of years. There are 3 unique years that stick out: 1900, 1860 and 1835. The reason why there exist surprisingly many artworks for these dates is connected to the cleaning of the column containing the creation date: 1900 is the result when an artwork is labeled with e.g. "19. Jahrhundert" (= 19th century), 1860 is the substitute year estimate for all artworks that lack a creation date and were made during the epoch of "Realismus" (= realism). Similarly artworks without a creation date that were made in the epochs of "Biedermeier" or "Biedermeier-Realismus" (=Biedermeier realism) were labeled with the year 1835. This also explains the spikes in figure 3.4, where the years are plotted chronologically.

The most used "Material Technique" values are, as shown in figure 3.5, "Öl auf Leinwand" (= oil on canvas), "Öl auf Holz" (= oil on wood) and "Öl auf Karton" (= oil on cardboard). This comes as no surprise as it could already be seen earlier in figure 3.2 that a significant number of artworks in the collection are paintings.

3. The Data

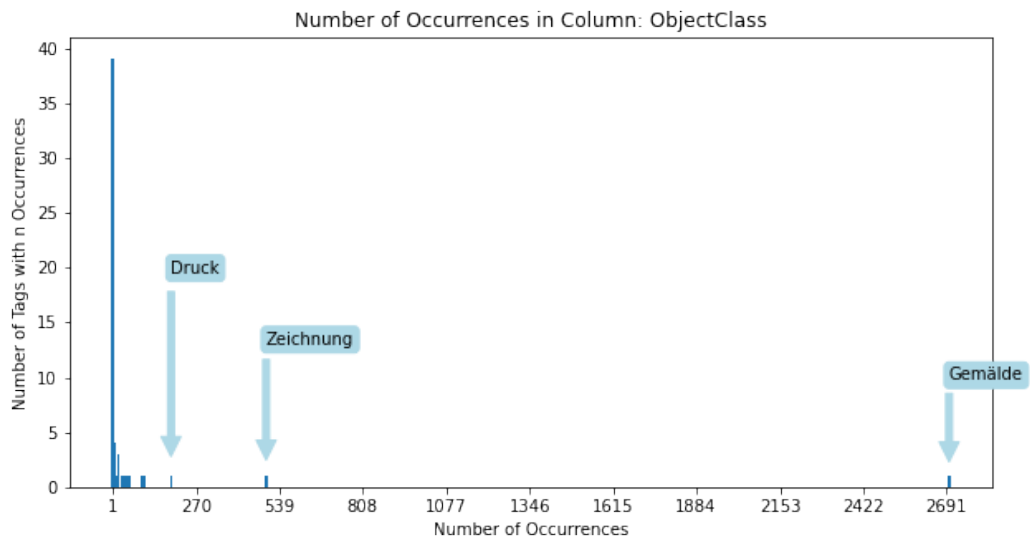


Figure 3.2.: Number of times a unique value from the column “Object Class” is assigned to an artwork

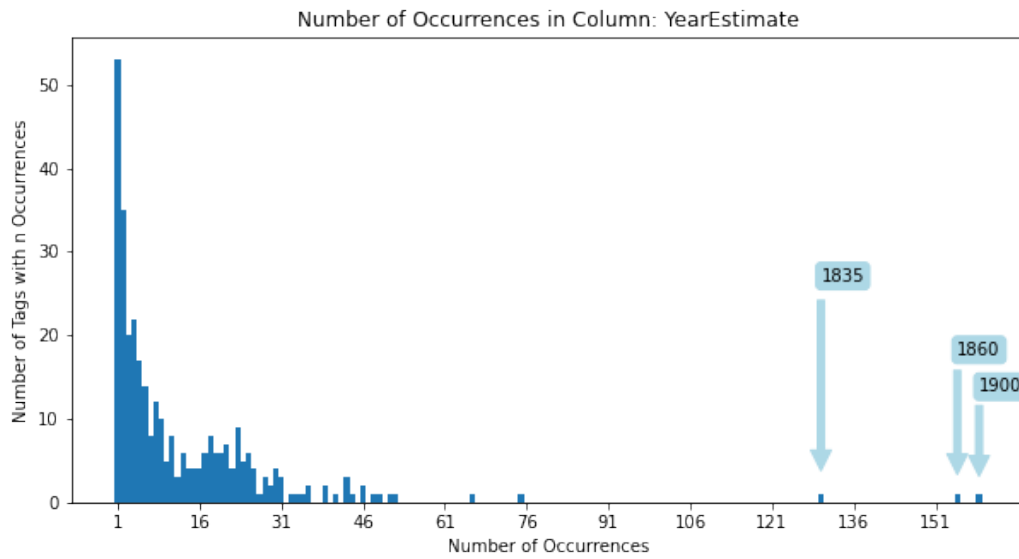


Figure 3.3.: Number of times a unique value from the column “Year Estimate” is assigned to an artwork

3. The Data

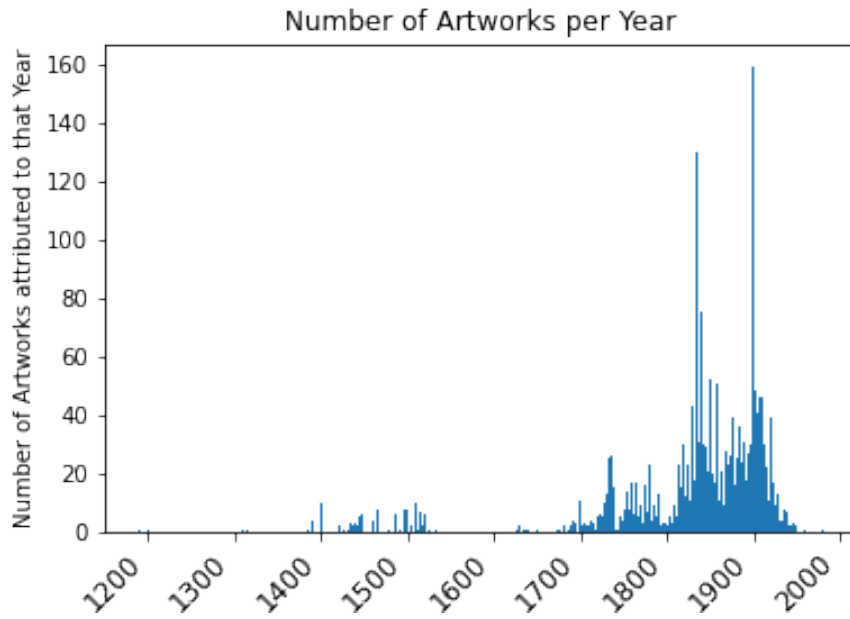


Figure 3.4.: Number of times a unique value from the column “Year Estimate” is assigned to an artwork, sorted by year

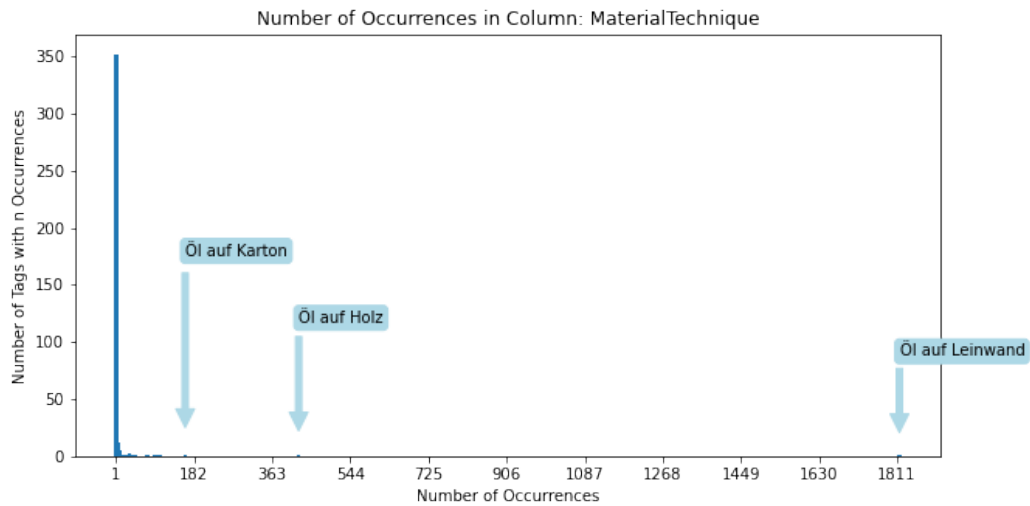


Figure 3.5.: Number of times a unique value from the column “Material Technique” is assigned to an artwork

Some artworks also contain descriptions. However, many of them do not talk about the artwork itself, but instead about the artist or are references (e.g. to inventory numbers). It

3. The Data

was found that some descriptions end with a citation ([Name, Year]) and these always concern the artwork itself. By filtering irrelevant ones out, 375 descriptions are left of the former 889.

3.2. Extraction of Tags

With the help of the methods from the natural language processing library “spacy” (see section 2.1.4), noun chunks are extracted.

A noun chunk is “a noun plus the words describing the noun” [13]. The detection of these noun chunks is not flawless, given the circumstance that the framework would normally be used with whole sentences as input, while in this thesis only short combination of words (e.g. the title of an artwork) can be given as such.

An example is the title “Abend am Mittenwaldsee”, (=“Evening at the Mittenwald lake”) for which spacy recognizes the words as following¹²:

Input	Output
Abend	Adverb
am	preposition with article
Mittenwaldsee	Noun, singular or mass

Table 3.1.: Tags assigned by the language model given the title of a chosen artwork as input

In table 3.1 it can be seen that the word “Abend” was wrongly labeled as an adverb. If the title is used in a sentence however, the same word is correctly labeled as “Noun, singular or mass”, as can be seen in table 3.2, which follows the similar procedure as was used for table 3.1, but with the input of “Wir verbringen einen Abend am Mittenwaldsee.” (=“We spend an evening at the Mittenwald lake.”).

¹²According to the output produced by the “to_json” method from which the value of the key “Tag” is used.

3. The Data

Input	Output
Wir	non-reflexive personal pronoun
verbringen	finite verb, full
einen	definite or indefinite article
Abend	Noun, singular or mass
am	preposition with article
Mittenwaldsee	Noun, singular or mass
.	sentence-final punctuation mark

Table 3.2.: Tags assigned by the language model given the title of a chosen artwork used in a sentence as input

Given this information, the question arises of whether noun chunks (or just nouns) should be extracted at all. An alternative would be using the whole phrase and perhaps removing the stop words. While this would be possible, it can be seen in section 4.1 that the time complexity of the application is driven by the number of unique tags. It is much more likely that a noun chunk is found in the data of multiple different artworks than it is to find a whole phrase used on multiple occasions. An additional drawback would be that the tags would then contain much more noise in the form of words that do not add any context. Therefore, this procedure reduces the number of unique tags and greatly increases efficiency, at the cost of losing some of the information.

Stop-words¹³ are filtered out, as well as a short collection of words that were added manually, such as articles. From these tags it was attempted to filter out entities that are persons, as information such as the name of an artist generally does not add a lot of context for the language model. However this worked only with limited success, as the language model generally needs longer inputs to perform entity recognition.

After this, some of the most common tags were removed manually. These were for example the tags containing the word "Inv.", which is used for referencing inventory. Words like "Bild" (= "image") were also filtered out as they do not add any context.

¹³= very common words that do not add a lot of context (e.g. the word "is")

3. The Data

3.2.1. Lemmatization

In the following the “similarity of two words” refers to the cosine similarity of their vector representations. This similarity ranges from -1 (=low) to 1(=high). There still exists a problem with different forms of the same word, which have a much lower similarity than one would expect. For example, the words “Mann” (=“man”) and “Männer”(=“men”) have a similarity of 0.637 and “geht” (=“walks”) to “gegangen” (=“walked”) has only a similarity of 0.551. To put these numbers into context, the similarity of “Messer” (=“knife”) to “Gabel” (=“fork”) equals to 0.627 and the similarity of “Eisen” (=“iron”) to “Stahl” (=“steel”) is 0.608.

Essentially this means that words that mean the same things are matching about as well as words that are just strongly associated with each other, but are actually different things. This can be solved by using lemmatization, which is the process of bringing words to their base form. So the words “geht” and “gegangen” will then both be “gehen” and therefore have a similarity of 1.

This processing is also provided by the natural language processing tools mentioned above.

It is worth noting that this process introduces some errors as well. For example the word “Herde” (=“herd”) got converted into the word “Herd”, which means “stove”.¹⁴ To counteract this, in this thesis a word is only lemmatized if its lemma is a more frequent word than the original word (according to the language model).

3.2.2. Out of Vocabulary Words

The language model that is used came with a large table of word vectors. However, there are still many words which are not covered by it. These are mainly compound words, which the German language has plenty of. This would in theory be easily solved, since fastText is able to generate vectors for previously unseen words as already described in section 2.1.3. However, it has to be considered that the fastText model would produce a representation of an unknown word based on the n-grams of a specific range of lengths that are contained in that word [3]. Since the n-grams were learned on the German language

¹⁴The words “Herde” and “Paar” were excluded from being lemmatized, since the errors were noticed.

3. The Data

unknown foreign words or names would therefore have a wrong vector representation. Letter sequences like “InvNr” (short for Inventory Number) would then also not be filtered out, as they should. Since word vectors can be constructed based on character n-grams, it is reasonable to assume that the same process would work for compound words when using not n-grams, but whole words instead. The word “Autofahrer” (=“car driver”) would then be approximated by the representations of “Auto” (=“car”) and “Fahrer” (=“driver”). The examples below show that this assumption is reasonable:

The match between the words “Autofahrer” and “Auto Fahrer” is 0.762. Between “Waschmaschine” and “Wäsche Maschine” it is 0.779. To put these numbers into context the similarity of two pairs of synonyms is shown: The similarity between “Schachtel” and “Kiste” is 0.693 and the one between “Wand” and “Mauer” is 0.573.

For all the reasons stated above it is reasonable to believe that such an approach can give a good representation of unknown words, without changing the language model. The algorithm of splitting words is described in section 3.3.

In the end there are only 9 out of 4135 artworks left without any tags generated from the titles of artworks¹⁵ or their associated expert annotations.

3.3. Word Splitting

In this section the algorithm used in section 3.2.2 is described. The motivation for its creation was also given in section 3.2.2.

To better understand the decisions in the design of the algorithm, a few of the challenges are listed below:

General Problems:

- Parsing the word letter by letter means that uncertainties need to be handled
 - Consider the “s” in the words “Landesangelegenheit” (\approx matters concerning the “Land”, a political zone) and “Landestreifen” (=landing strip). Both words are compound words made up from 2 words each. When parsing the two

¹⁵Even though every artwork has a title, some of these remain out of vocabulary even after applying the mentioned algorithm. Therefore some artworks do not have any title tags associated with them.

3. The Data

words from left to right, the algorithm eventually reaches the letter “s”. Up to this point the two words were completely identical, however, in the first word the “s” belongs to the first unit (which is “Landes”), but in the second word the “s” belongs to the second unit (which is “Streifen”).

German Language specific problems:

- Sometimes the letter “s” needs to be swapped with the letter “e” in order for a unit, which makes up a compound word, to be a correct word itself.
 - e.g. “Gebirgs” from “Gebirgspass” should actually be “Gebirge”
- Sometimes the letter “e” needs to be added to a unit that is part of a compound word in order for it to be a correct word itself.
 - e.g. “Hochschul” from “Hochschulautonomie” should actually be “Hochschule”
- Some words can be combined to form longer words in multiple valid ways.
 - e.g. Should “Abend”, “Land”, “Schaft” result in “Abendland” and “Schaft” or “Abend” and “Landschaft”

It has to be said at this point that for this algorithm naturally not all of the German grammar was considered, since the aim of developing it was not general applicability, but to improve the data cleaning process of the dataset at hand.

The algorithm prefers having no results over uncertain ones (e.g. extremely rare words) and has a set of rules in place of when to determine if an input is invalid and therefore return an empty result list. This also filters out letter combinations like “InvNr”.

It is also worth noting that the resulting words of the split will still be treated as one tag in later processes.

In figure 3.6 an overview of the algorithm is given. In figures 3.8, 3.7, 3.9, 3.10, 3.11, 3.12 and 3.13 further details are given. Note that some of the examples used are illustrative and not actual examples. This is in order to showcase the concepts in a manner that is as simple as possible.

In the overview given in figure 3.6 it can be seen that words are parsed from left to right. If the already parsed part of the word is in the set of the 150,000 most frequent words

3. The Data

that were learned by the language model, it is seen as valid. The reason for this threshold is that otherwise too many “words” would be considered that are not actual words, but still are found in the language model. Presumably the model picked these up from data generated by crawling the web.

In the next step shown in that figure single words are recombined with endings that have previously been cut off and finally whole words are recombined again to the largest unit(s) that are valid words.

Note the boxes on the right of the figure. Here it is shown ...

- ... what word endings are allowed to be dropped (box “acceptable to miss”)
- ... what is considered as endings that should not be dropped (box “endings”)
- ... which of the words in the set of most frequent words should be ignored. There are some English words to be found for example. This again is presumably caused by using data from crawling the web. (box “filter out”)
- ... which valid words should be filtered from the final result as they do not add enough context (e.g. numbers) (box “filter from final result”)

Figure 3.8 further explains the step “Change Ending and Case” seen in figure 3.6. Note that it is tested here whether it makes sense to append the letter “e” to a word. The reason for this is that in German, the letter “e” is often cut off from words when they are used to form a compound word. An example is the combination of the words “Hochschule” and “Autonomie”, which results in “Hochschulautonomie”.

Figure 3.7 shows how in the first step of parsing a shorter word can be preferred over a longer one based on its rank.

Figure 3.9 shows how some words used in compound words need a letter swap to be valid words on their own. An example for this is the combination of the words “Gebirge” and “Kamm”, which results in “Gebirgskamm”.

Figure 3.10 shows how the problem can be solved that, when parsing a word from left to right, it is uncertain whether a letter belongs to the current word or to the next. This can be illustrated by parsing the words “Landestreifen” (= landing strip) and “Landesangelegenheiten” (\approx matters concerning the “Land”, a political zone). While in the former

3. *The Data*

compound word the “s” belongs to the first word (“Landes”), in the latter the “s” is part of the second word (“Streifen”).

Figure 3.11 corrects an oversimplification shown in figure 3.6. For simplicity it was shown in figure 3.6 that the algorithm continues to look for the next word as soon as one word is found and recombines word endings in a second step. However, it is much safer to check whether the found word has an ending that has not been reached by the parser yet before moving on to looking for the next word. Since endings are sometimes only single letters, not every ending found this way is correct and thus the same procedure as explained in figure 3.10 is applied to make sure that all the options are considered.

In figure 3.12 it is shown how fragments can be combined into valid words again.

Finally, in figure 3.13 special examples of the recombination of found parts are shown.

3. The Data

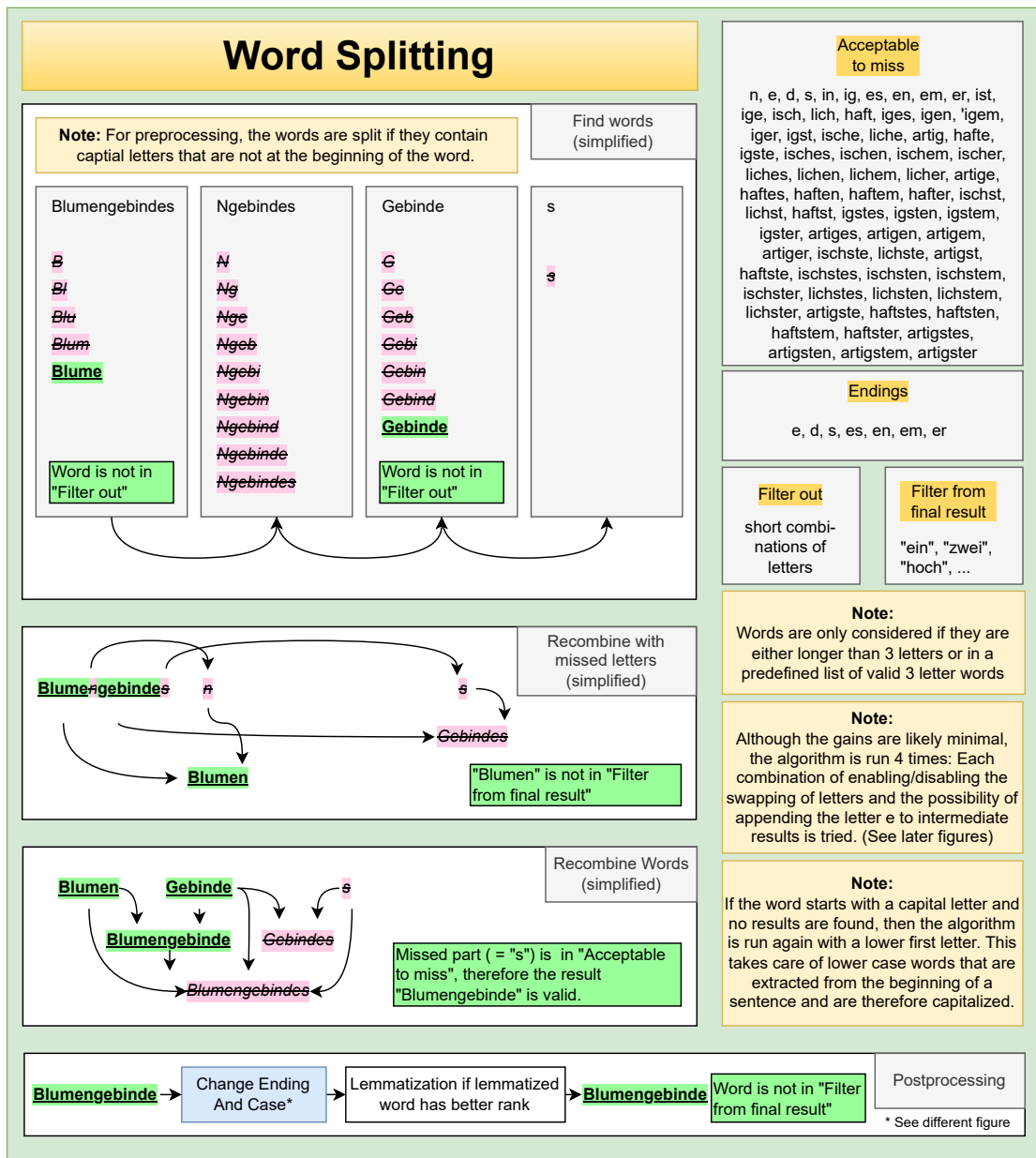


Figure 3.6.: Simplified overview of the word splitting algorithm

3. The Data

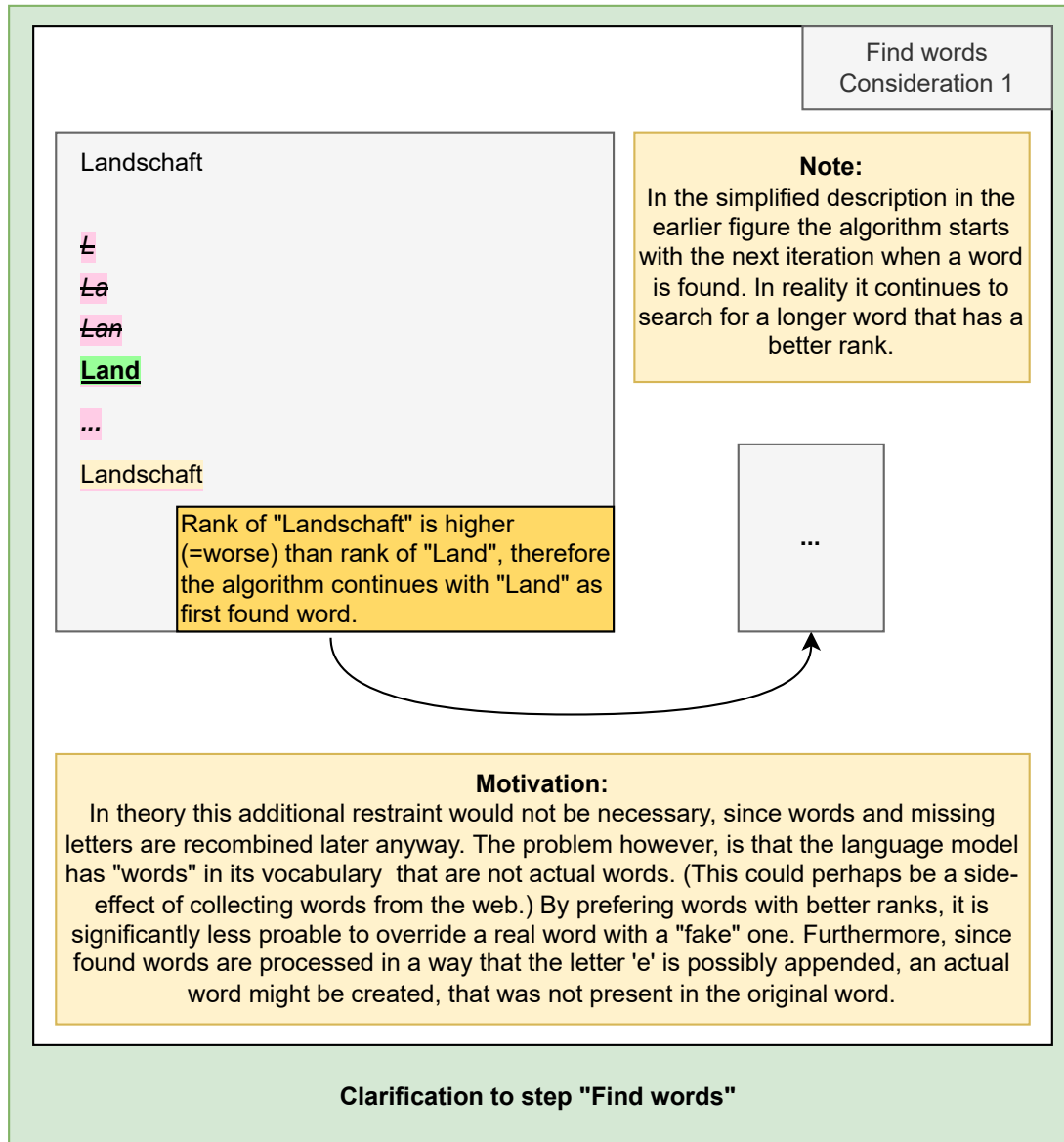


Figure 3.7.: Word Splitting: Clarification for step "Find words"

3. The Data

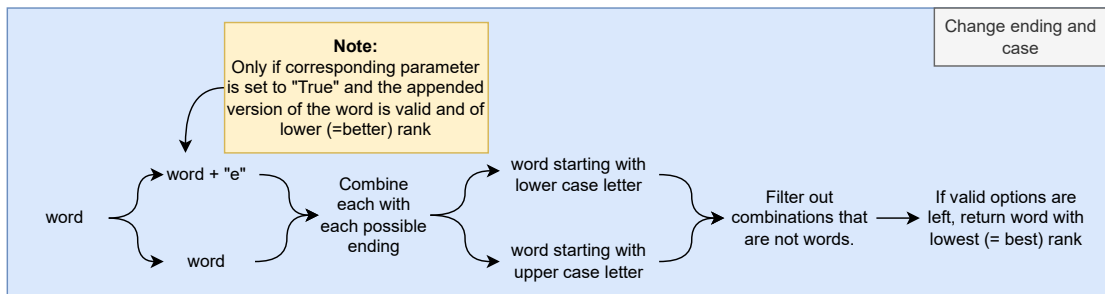


Figure 3.8.: Word Splitting: Step of changing ending and case of words

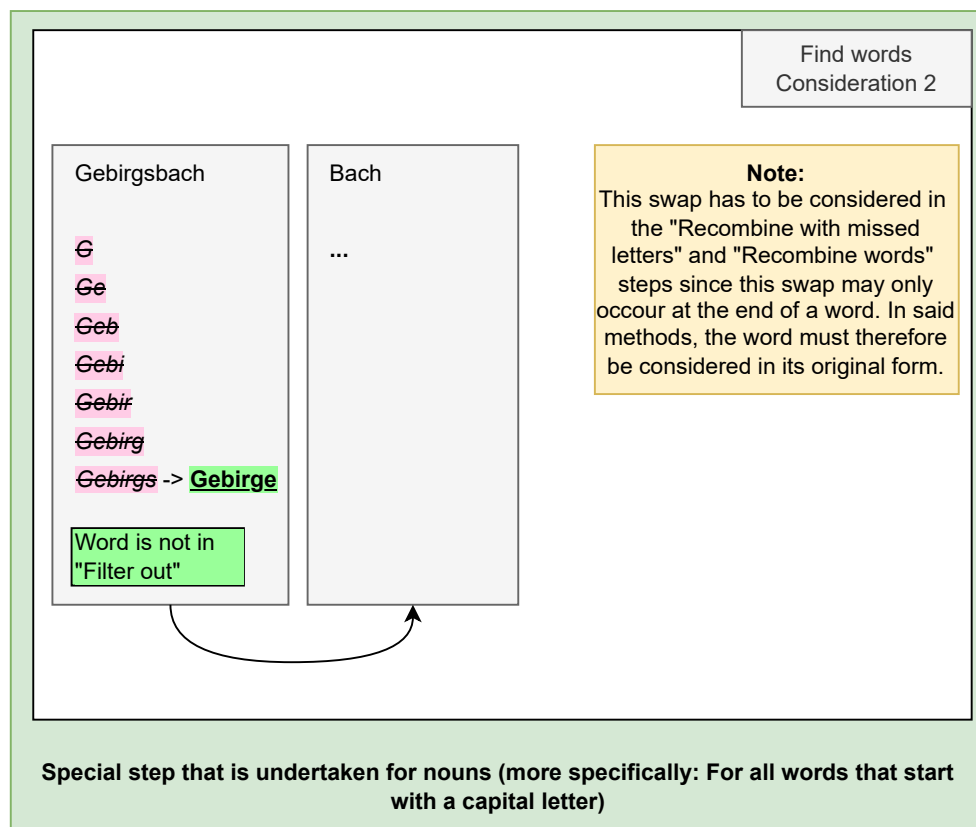


Figure 3.9.: Word Splitting: Letter substitution

3. The Data

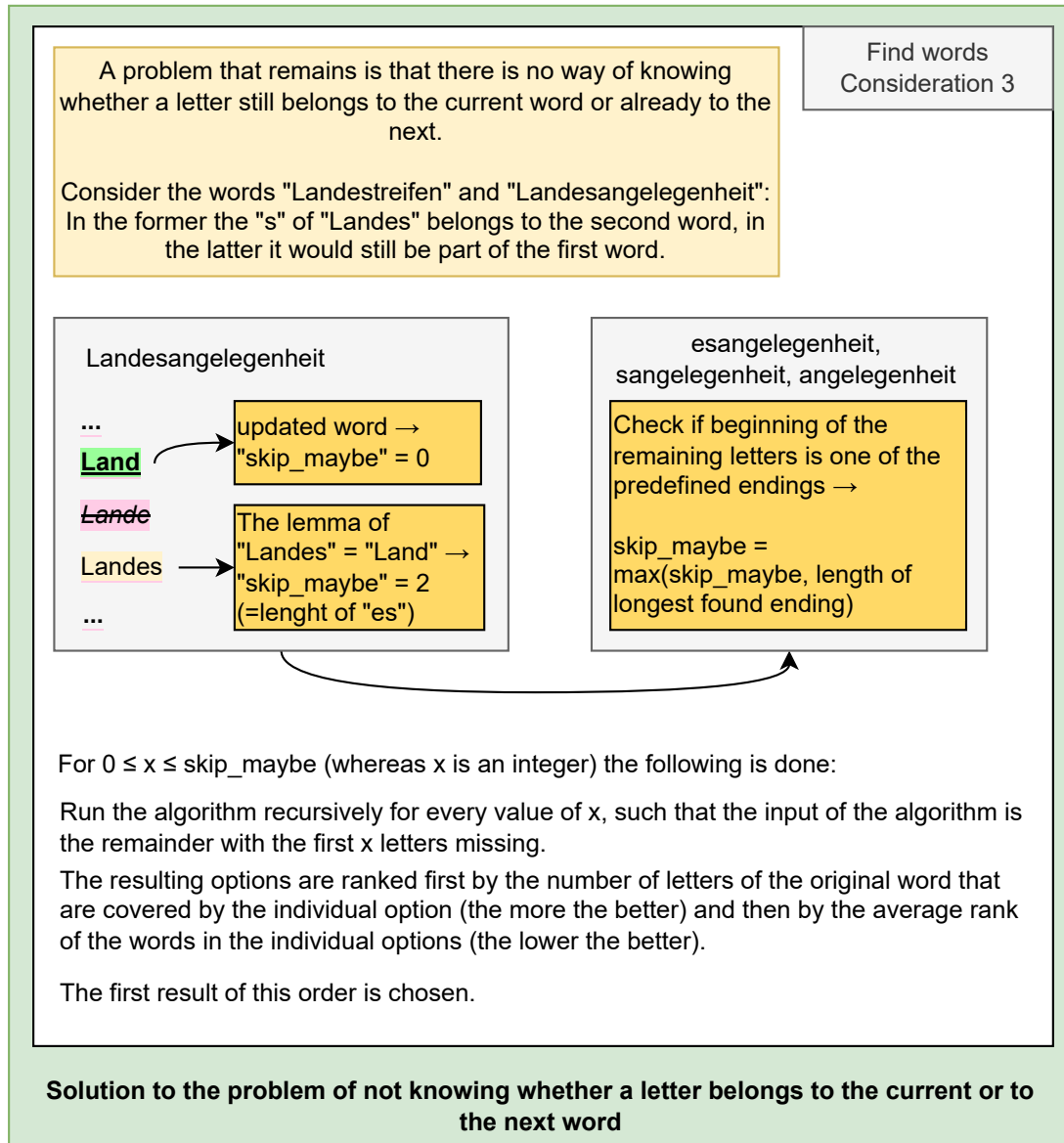


Figure 3.10.: Word Splitting: Handling the problem of not knowing whether a letter is part of the current or next word

3. The Data

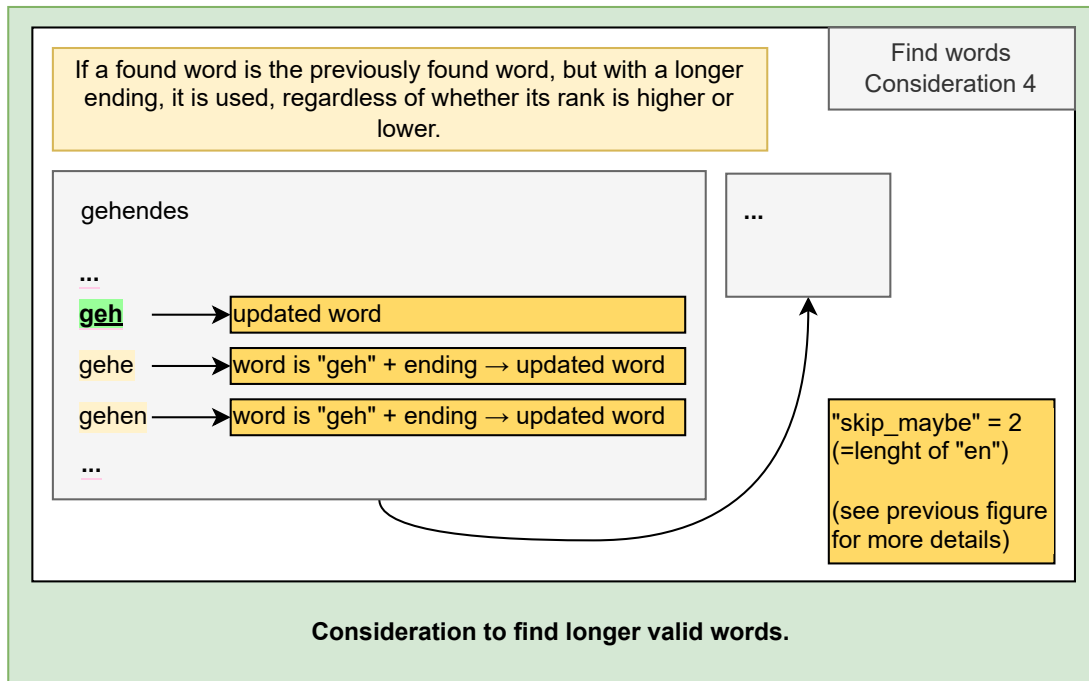


Figure 3.11.: Word Splitting: Words and Word-endings

3. The Data

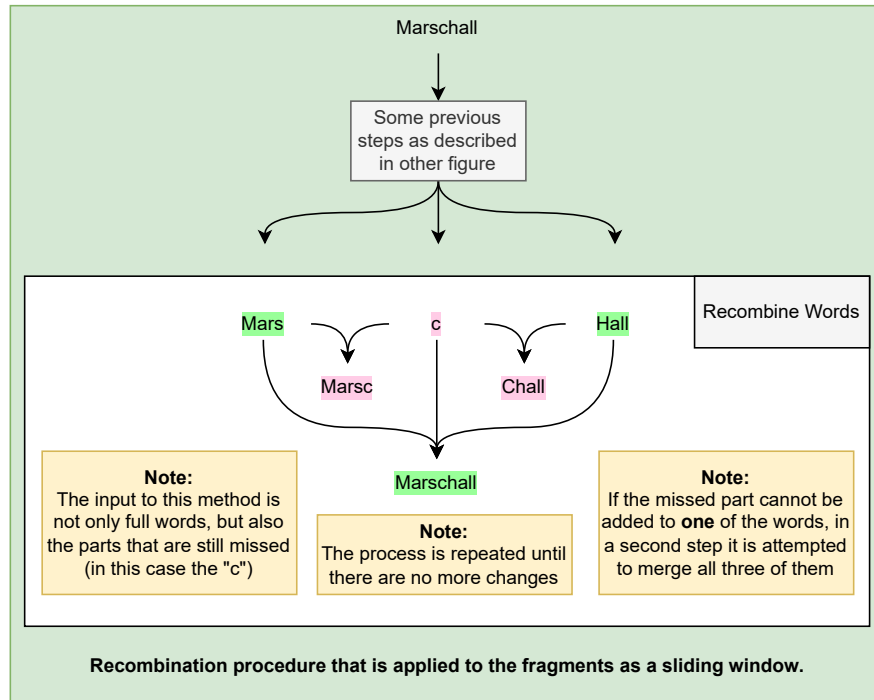


Figure 3.12.: Word Splitting: Recombine words step

3. The Data

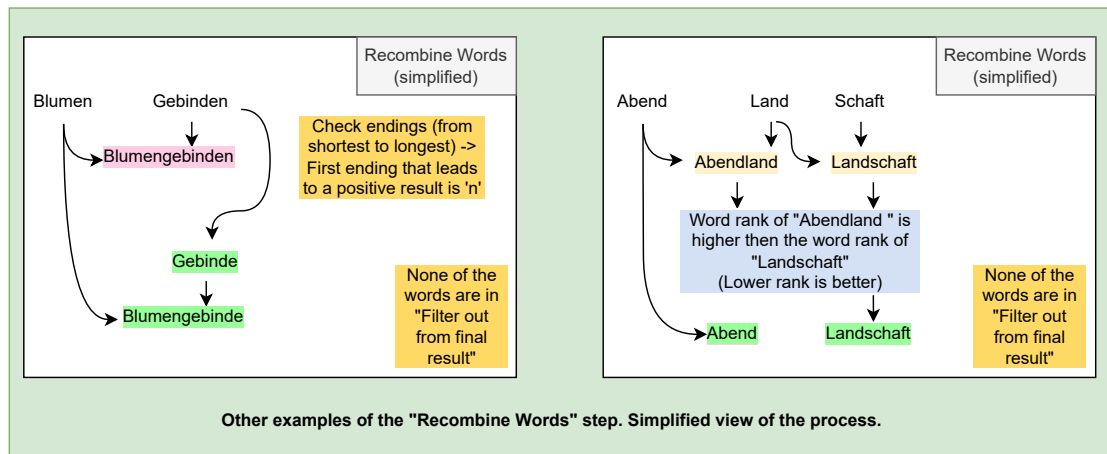


Figure 3.13.: Word Splitting: Recombine words special considerations

Finally, in table 3.3 some selected results of the word splitting algorithm are shown.

Input	Output
altmeisterlichen	Altmeister
Landschaftsstudien	Landschaft, Studie
Abendlandschaft	Abend, Landschaft
Wildschweinstillleben	Wildschwein, Stillleben
Sterbesakramenten	sterben, Sakramenten
Salbstein	Salbe, Stein
Gußhaus	Guß, Haus
Kleidermacherstochter	Kleid, Macher, Tochter
Hochschulautonomie	Hochschule, Autonomie
Gebirgskamm	Gebirge, Kamm
halbflüssiger	halbe, flüssig
Männerkopfes	Mann, Kopf
Dreifigurig	Figur
Keyserin	–Empty Result–

Table 3.3.: Examples of results of the word splitting algorithm

3. The Data

3.3.1. Statistics and Selection

At this point almost 4000 unique tags were extracted from the titles and almost 3400 unique tags were extracted from expert annotations. Considering some overlap, this totals to about 7350 unique tags. From the descriptions almost 10000 unique tags were extracted, but in this section it will be seen why these will not be considered after all.

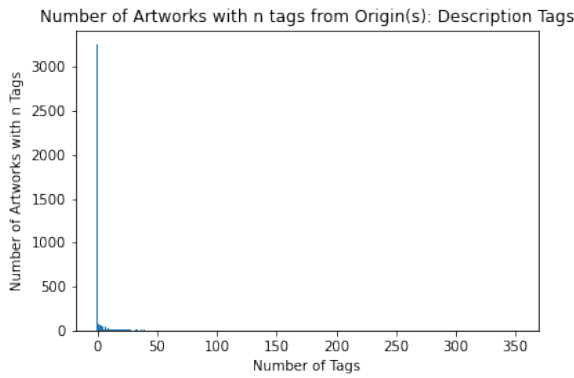


Figure 3.14.: Number of artworks with specific number of tags, considering tags extracted from Descriptions

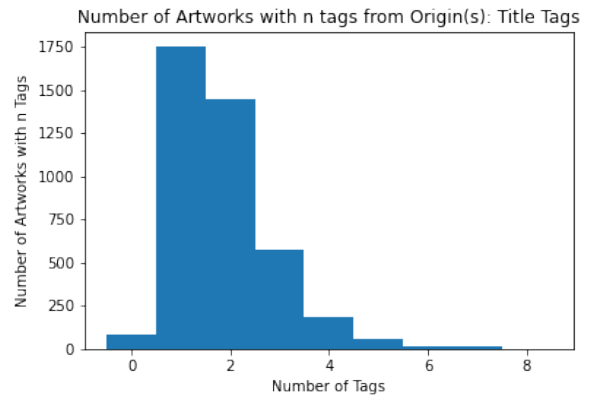


Figure 3.15.: Number of artworks with specific number of tags, considering tags extracted from Titles

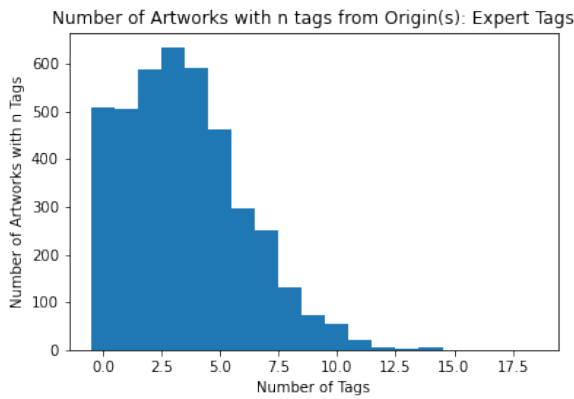


Figure 3.16.: Number of artworks with specific number of tags, considering tags extracted from Expert Annotations

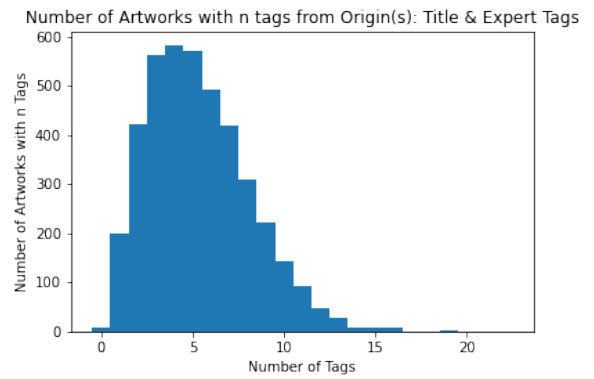


Figure 3.17.: Number of artworks with specific number of tags, considering tags extracted from Titles and Expert Annotations

3. The Data

Figure 3.14 shows a very skewed distribution: Most artworks do not have any tags generated from descriptions, but, looking at the x-axis labels, some have a lot of them. Figure 3.15 shows us that almost all artworks are associated with at least one tag extracted from its title. Figure 3.16 shows that a fair number of tags can be extracted from parsing the expert tags. In 3.17 it can be seen that considering tags from the two sources titles and expert tags combined gives a normally distributed amount of tags.

Figures showing combinations of tags extracted from descriptions together with tags of other origins were omitted since these distributions are very skewed as well.

While the program created for this thesis also supports the handling of description tags, at some point a decision had to be made about whether to use them or not.

The tags extracted from descriptions have several downsides:

- A very skewed distribution over the artworks
- A generally lower quality of the tags - In comparison to long descriptions, titles for example usually catch the “essence” of an artwork in very few words.
- Combinatoric problems suffer from a high time complexity. Hence, increasing the number of unique tags by such a large amount would lead to a very significant increase in processing time necessary to compute the caches.

Therefore, in regards to textual sources the program can now utilize tags from the title, the expert tags or the combination of both. Tags denoting objects found in the images can be added to any of these as well.

The most frequent values of these origins can be seen in figures 3.18 and 3.19.

3. The Data

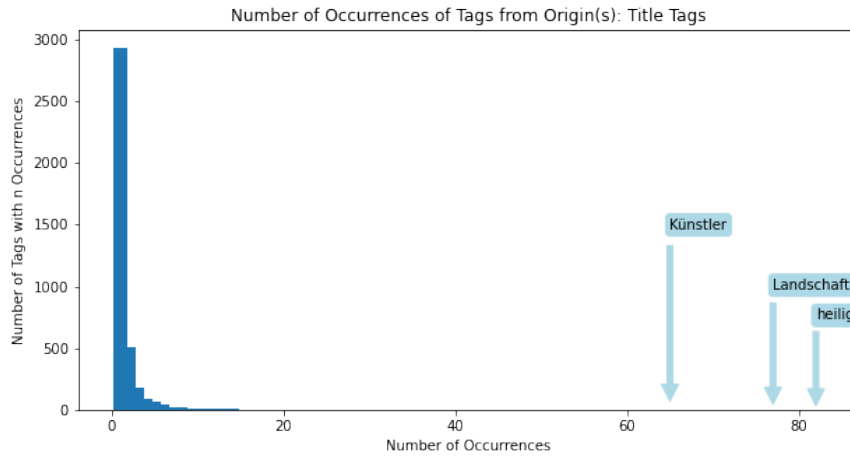


Figure 3.18.: Number of times a specific tag extracted from image titles occurs

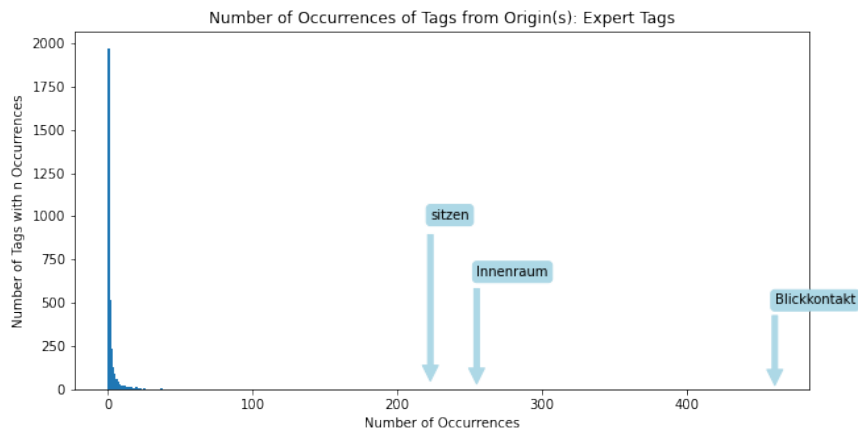


Figure 3.19.: Number of times a specific tag extracted from image expert tags occurs

3.4. Countering Bias towards longer Tags

Consider the titles of the artworks¹⁶ “Bild einer Landschaft” (=“picture of a Landscape”) and “Bild einer Landschaft im Herbst” (=“picture of a landscape in autumn”). The first title could result in the tag “Bild” and the tag “Landschaft”, whereas the second title might result in the tags “Bild”, “Landschaft” and “Herbst”.

¹⁶Invented examples

3. The Data

This means that there would exist a bias towards longer tags (or more specifically towards tags that contain more noun chunks) in later comparisons. The reason for this bias is that each resulting tag of the image in question is used for pairwise comparisons and in the end all these pairwise comparisons are averaged as seen in section 3.4. As a consequence, a source¹⁷ that leads to more extracted tags than another will weigh heavier in the final sum. To counter this, a weight is assigned to each label. Specifically the weight is defined as: $w_t = \frac{1}{n_t}$ where w_t is the weight for a specific tag and n_t is the number of tags that were extracted from the source. The concrete process in which weights are considered to counter bias is shown after giving an example of the resulting weights. It is important to mention that the same tag can have different weights assigned depending on the artwork it belongs to and the source it was extracted from (see example below).

In later stages of this thesis objects recognized on the artwork will be added as tags. For these objects the weight is assigned as $\frac{1}{|\text{objects found in artwork}|}$.

As an example, here the results of processing the textual information for the artwork called “Allegorie der Physik” made by Hanns Gasser is presented.

Title:	Result in the format (tag, source, weight):
<ul style="list-style-type: none">• Allegorie der Physik	<ul style="list-style-type: none">• (Allegorie, Title, 0.5)
Expert Tags:	<ul style="list-style-type: none">• (Physik, Title, 0.5)
<ul style="list-style-type: none">• Physik	<ul style="list-style-type: none">• (Physik, Expert Tag “Physik”, 1)• (Entwurf, Description, 0.25)
Description:	<ul style="list-style-type: none">• (Statue, Description, 0.25)
<ul style="list-style-type: none">• Entwurf für eine Statue des Arsenal in Wien	<ul style="list-style-type: none">• (Arsenal, Description, 0.25)• (Wien, Description, 0.25)

As already mentioned above, a bias exists towards sources that contain more noun chunks than others. The intuitive way of applying weights to similarities by multiplication would be problematic, as then there would be no difference between a tag that has a low general

¹⁷This means e.g. the title of an artwork or a **single** expert annotation

3. The Data

similarity and a tag that has a high general similarity, but appears only with low weights associated to it. Instead the procedure that is used in this thesis is the following:

1. Collect all tags of all sources that are set to active (e.g. from titles and expert annotations) as well as the weights associated with these tags. Before the tags of an artwork are added, they are filtered according to the method described in section 4.1.3.
2. Shuffle the collected list. Further on this list will be referred to as “all tags list”
3. Go through the “all tags list” - if a random value (between 0 and 1) is smaller than the weight of a tag (which is also between 0 and 1), discard the tag, otherwise add the tag to the “sample list”
4. Stop if the list is looked through or a maximum number of tags is reached, which can be defined in the configuration file. Note that it is possible that the same tag shows up in the “sample list” multiple times if multiple artworks have been labeled with it. This is an important detail, since the “sample list” should be a representative sample of the dataset.
5. For every unique tag t in the “all tags list” define the average similarity as the average cosine similarity between the vector representations of t and s for all s in the “sample list”.

The weights are important in this process. In section ?? it was shown how some longer sources can result in multiple tags. Without considering weights, the “sample list” would be biased towards these longer sources.

An alternative to this algorithm would be using the “TF-IDF” method, but it would only consider exact matches. While this would be fast, it would come with a considerable drawback: If in our dataset the word “boat” appears 500 times, but the word “ship” only once, then the latter would be considered a very specific term. This characteristic would not be ideal and because of the caching, the speed of this operation is not a very strict requirement.

3.5. Box Annotation Dataset

In order to train a network on the image data, first an additional dataset had to be created, which contains 250 of the artworks, each annotated with bounding boxes. Since the process of drawing bounding boxes is very time consuming, only a very small portion ($\approx 6\%$) of the artworks was labeled. In a second step the classes of the bounding boxes are aggregated in order to decrease the amount of unique classes and to increase the count of entities per class. For example “Esel” (=“donkey”) and “Pferd”(=“horse”) were bundled into “Esel oder Pferd” (=“donkey or horse”). For further processing only those classes are kept which appear in at least 50 different artworks.

This leaves the following classes, the number is the amount of samples in which the class appears at least once.

- | | |
|---------------|----------------|
| • Kopf: 165 | • Baum: 64 |
| • Person: 145 | • Gewässer: 52 |
| • Hand: 103 | • Gebirge: 51 |
| • Wolken: 92 | • Gebäude: 51 |

Unfortunately a few of these classes are some of the less interesting ones and will likely not add as much semantic information as some of the other, more specific classes would have had. For example the tags “Turban” (=“turban”) or “Ruine” (=“ruin”) would have also appeared in the box annotated dataset. The solution to this would of course be annotating a larger proportion of the dataset. Eventually even the rarer classes would have been assigned often enough to reach this required threshold. However, this remains work for the future.

Some classes were difficult to annotate, for example in the case of trees that are part of a forest it often was not clear where the individual trees begin and end.

3.6. Tree and Person Annotation Dataset

In order to preserve the box-annotated samples for training and validation, two additional datasets, each with 1000 artworks, are created. In these test sets the contained artworks are labeled with boolean values, indicating whether they contain a tree or person respectively. This labeling process is very fast as the additional bounding boxes do not have to be annotated. It is worth noting that in the annotation process artworks were skipped where it was unclear whether something is to be understood as tree/person or not. In creating the person-annotation dataset another difficulty was determining whether something is to be seen as a person or just a head. If only a head was visible the artworks were skipped and therefore are neither positive nor negative samples. If a fair bit of the upper body was visible as well, it was labeled as containing a person.

4. Implementation

In this chapter concrete details to the implementation of the algorithms as well as to the software architecture and its capabilities are discussed.

4.1. Similarity Calculations and Caching

A central question of this thesis is how to determine similarity based on tags. Therefore, in this section this process and how to handle its time complexity is described in more detail.

4.1.1. General Process of calculating Similarities

Figure 4.1 describes this process with two example artworks, whereas the first one has two text tags associated to it and the second one only one. In order to being able to calculate with tags, it is necessary to convert them to vectors, the processes for which are described in sections 2.1 and 2.3. Then the Cosine Similarity is calculated for each pair of tags for which one tag belongs to the first artwork and the other tag to the second artwork.

4.1.2. Normalization of Similarities

Averaging these values would already lead to an acceptable result and therefore some of the steps shown in “Step 2” of figure 4.1 are not strictly necessary. It should be considered however that some matches are more interesting than other. In the dataset used in this thesis many artworks include very general terms in the resulting tags like “Landschaft” (= “landscape”) for example. For this reason all similarities are corrected by their expected

4. Implementation

similarities based on how well the tags match all other tags on average as seen in “Step 2” of figure 4.1 and further described in section 3.4. The effect that this should achieve is that the concepts that appear less frequent are preferred in the similarity calculation, since these are more interesting. However, since “interestingness” is subjective, it is not possible to quantify the gains of this approach.

4. Implementation

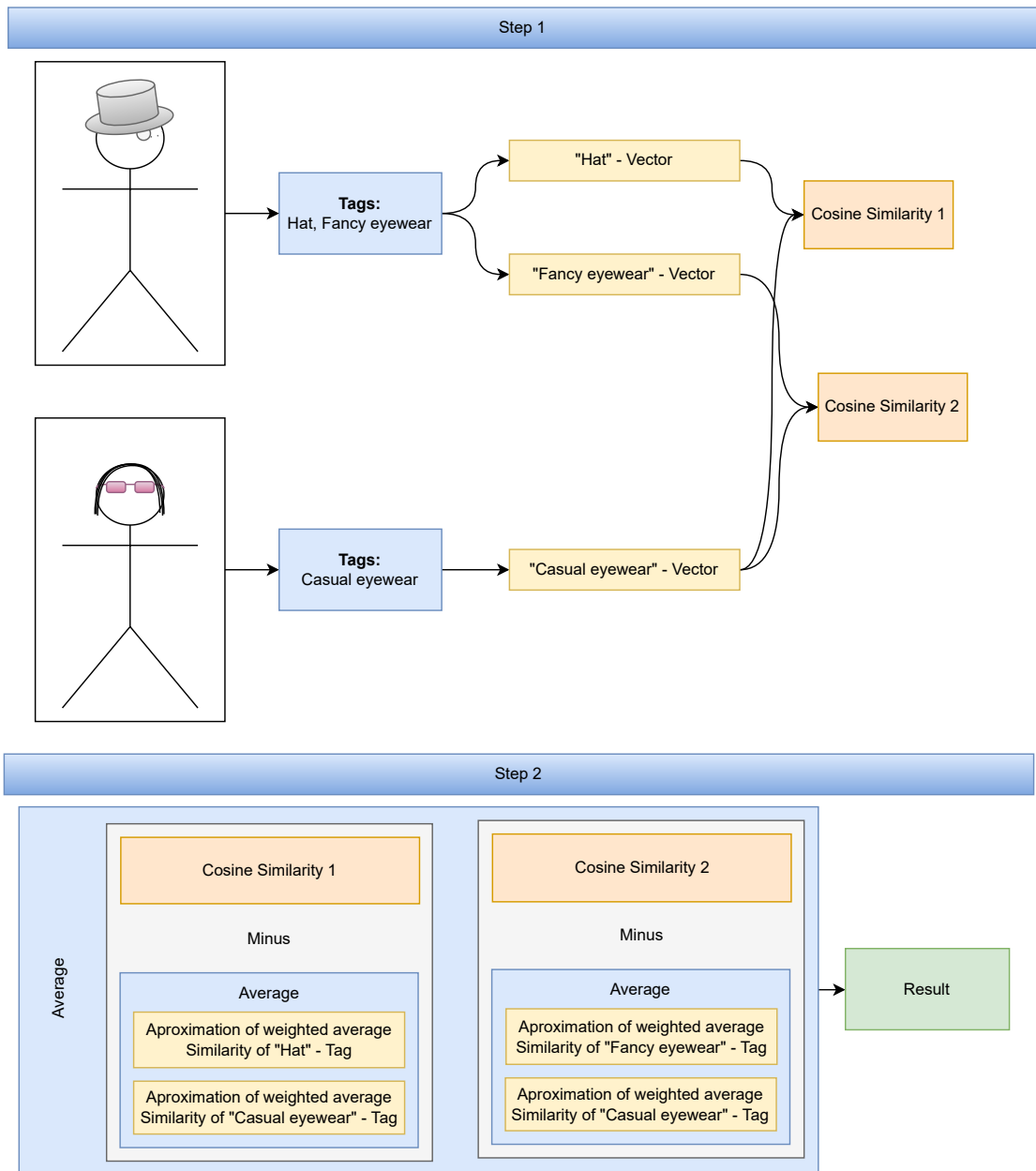


Figure 4.1.: Calculation of Image Similarity

4. Implementation

4.1.3. Framework

All similarity related functionality is bundled into a customizable framework, which is called “AutoSimilarityCache”. It requires only little configuration to handle all kinds of media and metrics (see left side of figure 4.2). The methods which are meant to be adapted are bundled into a single file in which it has to be defined how tags can be fetched from the database, how tags¹⁸ of each source should be compared to each other and what combinations of different sources are valid. The framework computes the caches and then offers methods (see right side of figure 4.2) to e.g. instantly retrieve the similarity between artworks as defined in this thesis.

Lastly, sources can be assigned to a group, having the effect that similarity comparisons are made only within-group. The similarity score is then calculated using a weighted score between the groups, whereas the weights can be applied instantly¹⁹. This feature was meant to enable the framework to also handle e.g. style information, for which it would not make sense to compare it pairwise to every semantic tag. As the analysis of image styles remains a work for the future, this feature was not used.

¹⁸which can be text or any other media

¹⁹meaning that no recalculation of caches etc. is necessary

4. Implementation

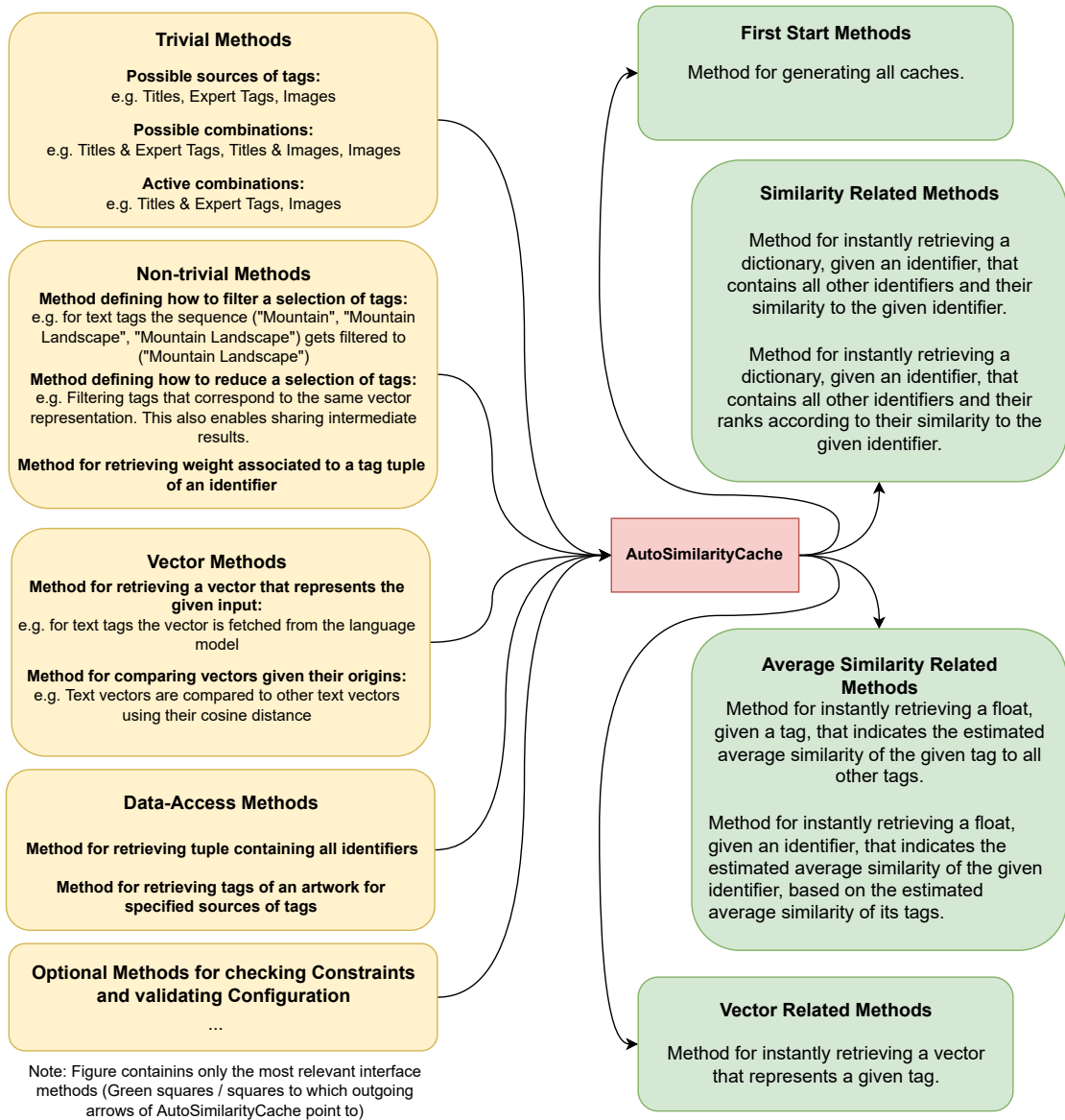


Figure 4.2.: AutoSimilarityCache - Required Methods and Interface Methods

4. Implementation

Filtering of Text Tags

In figure 4.2 in “Non-trivial Methods” one entry of the list is “Method defining how to filter a selection of tags”. For all origins that are associated with text tags, this is done in the following way:

- Identical tags are filtered out (The tag with the greater weight is kept)
- Tags that are fully contained in other tags (on a per word basis) are filtered out
- For example: Of the three tags²⁰ “Filtering is great”, “Filtering is”, “Filter” only “Filtering is” would be filtered out.

Reduction of Text Tags

In figure 4.2 in “Non-trivial Methods” one entry of the list is “Method defining how to reduce a selection of tags”. This method makes sure that tags that would result in the same vector are filtered out, so that every tag corresponds to a unique vector. The method also returns a dictionary that contains all the tags that were filtered out as keys and the tag that was not filtered out, which corresponds to the same vector as value. With this measure efficiency is further improved by enabling the sharing of intermediate results across artworks.

Increasing Efficiency

Since there are thousands of artworks, calculating the similarities between each pair of them is very time consuming. The time complexity of such an operation is high since the tags of every artwork have to be compared to the tags of every other artwork. Therefore, caching is absolutely essential in this case. The average similarities of all tags (see section 3.4) are needed for every one of these calculations and thus they have to be cached as well. The vector representations of tags should only be created once for efficiency reasons and consequently this too gets cached. Using the speed-up that the storage of intermediate results provide the main cache is generated, which contains the similarity of every artwork to every other artwork.

²⁰Made up examples

4. Implementation

This concludes the most important caches. Figure 4.3 further shows the “FilterCache” which will be described in section 4.2. What can also be seen in figure 4.3 is that some of the caches are split up according to the sources of tags that they cover. This allows not only simple and fast experimentation in the later stages of the project, but also make it possible to reuse calculations, as will be described later in this section.

4. Implementation

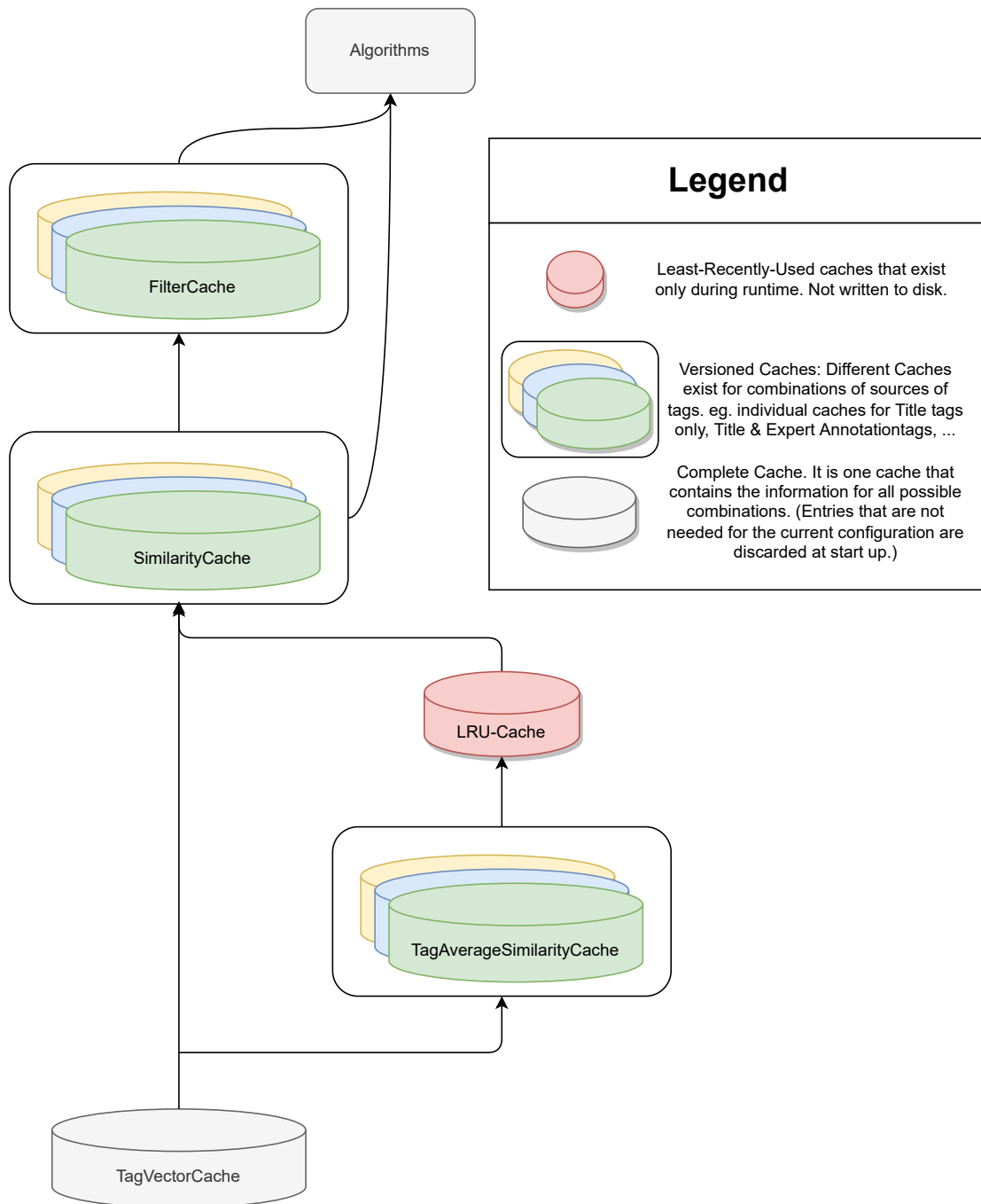


Figure 4.3.: AutoSimilarityCache - Caching Scheme

4. Implementation

When generating the “SimilarityCaches”, which store how similar all other artworks are to each specific artworks, two measures are taken to improve efficiency:

- Using symmetries
 - If item a has a similarity of e.g. 0.45 to b , then b also has the same similarity to a
 - Using this symmetry property during data generation means that only half of the entries need to be calculated
- Bottom up data generation
 - **Combinations of two sources of tags**

If tag pairs are compared considering two sources A and B then the following cases can appear:

- * The tags are of the same source, meaning both are of source A or both are of source B
- * One tag is of source A , while the other is of source B

Therefore, if tag pair similarities are first generated only considering source A and then considering only source B , then the generation of tag pairs that considers both sources can be optimized such that in the case of pairs where both tags are of the same source, the value can be looked up instead of it having to be calculated. Looking at it from the other direction this means that when calculating the cache for a combination of two sources, the caches that consider the individual sources can be generated for free.

- **Combinations of more than two sources**

Since only pairs of tags are compared, there can only ever be 2 sources involved in each comparison. Therefore, if each combination of two sources is calculated beforehand then not a single calculation has to be done. Essentially, this means that all combinations of more than two sources are generated at very low cost, given that the possible combination of source pairs that are contained are already generated.

4. Implementation

4.2. Filter Cache

In order to filter the results from the other caches in a very fast manner, the “FilterCache” was created, which precalculates certain metrics for every artwork in the collection.

The “FilterCache” currently features filtering results based on the ...

- ... (minimum / maximum) (number of tags / sum of weights of tags) associated with (the origins that are currently considered / specific origins)
 - For text tags, the sum of weights equals to the number of **sources**²¹ that tags were extracted from, making this a more reliable measure than just counting tags
- ... presence / absence of a specific tag
- ... similarity an artwork has to a given input. (Input phrase and min./max. required/allowed similarity to the phrase can be given as input in pairs).

4.3. Path-Generation Algorithm

A first application using the framework presented in the earlier sections of this chapter is creating a semantic path between two images, an example of which can be seen later in e.g. figure 5.1. The process is explained further in the rest of this section.

4.3.1. Prior work on Path-Generation

The Path-generation process of this thesis is inspired by the process used in the papers [12, 5, 14]. However, the usage of caches significantly reduces the computation time and therefore made it possible to implement some improvements to the algorithm.

²¹This means e.g. the title of an artwork or **single** expert annotations

4. Implementation

Before discussing the algorithm used in this thesis, the approach of the aforementioned papers is described. There are small differences in the algorithm used in [12, 5] and [14]²². The following description follows the one given in [5]:

1. Cosine similarities to the start and end image are computed for all artworks
2. For further processing the union of a given number of the most similar artworks to the start image and a given number of the most similar artworks to the end image are kept
3. For each of these images the ratio of the cosine similarities to the start and end images are computed
4. For every position on the path that is to be generated the ratio is computed that would be ideal
5. For each of these positions the artwork with the closest matching similarity ratio is chosen

A problem arises for the image that is right at the middle of the path [5, 12, 14]: The ideal ratio for the middle image is 1. This ratio will be the result not only when two images are an *equally good* match to the start and end image, but also when they are an *equally bad* match. For this reason the second step of the above described process is discarding most of the images and only keeping those that are similar to both the start and the end image. In [14] 95% of the images were discarded in that step.

4.3.2. The Path-Generation Algorithm for this Thesis

Instead of using as a measure the absolute distance between the actual and ideal similarity ratio as was done in [12, 5, 14] and described in section 4.3.1, in this thesis the best match is found by applying the following formula:

- $e :=$ An entity (an artwork)

²²The algorithm is generally the same, but in [12, 5] similarity is used as a metric, whereas in [14] divergence is used for the same.

4. Implementation

- s_x^y := The similarity of artwork x to artwork y as defined in section 4.1, whereas $s_x^y = s_y^x$ and $s_x^y \in \mathbb{R}, \in [-1$ (=low similarity), 1 (=high similarity)]
- I := Input (set containing the start and end artwork)
- A := A set of artworks $\notin I$
- W := A dictionary associating every position on the path with a weight towards the start artwork and a weight towards the end artwork.

$$\text{metric}(I, a, W) := \sum_i^I \frac{(1 - s_a^i) * W_i}{\sum_b^A (1 - s_b^i)} \quad (4.1)$$

$$\text{Best_match}(I, A, W) = e : \text{metric}(I, e, W) = \min(\text{metric}(I, a, W) \forall a \in A \setminus I) \quad (4.2)$$

In equation 4.1 s are indicators for similarities in the range of $[-1$ (=low similarity), 1 (=high similarity)]. Using $(1 - s_n)$ brings the similarities into a positive range of $[0$ (=high similarity), 2 (=low similarity)]. Note that the meaning of low and high results is now flipped, which means that the lowest result is the best one in the end. Using the weighted sum instead of the ratio solves the problem that was described in section 4.3.1 where a ratio could be met both by equally good matches and equally bad matches. Note that $\sum_b^A (1 - s_b^i)$ is the sum of all similarities to i (which is either the start- or the end artwork) and acts as a normalizing term, which puts the term $(1 - s_a^i) * W_i$ into context: A high result of s_a^i is less impressive if $\sum_b^A (1 - s_b^i)$ is high.

The Python algorithm for calculating a pair²³ of weights for every position between the start and end is the following, whereas “intermediate_steps” is the number of steps between start and end image:

```
for i in range(i, intermediate_steps + 1):
    weights.append(intermediate_steps - i + 1, i)
```

Figure 4.4.: Python algorithm for calculation of shifting weight pairs

²³The pair contains one weight towards the start artwork and one weight towards the end artwork

4. Implementation

Note that “range(1, intermediate_steps+1)” is equivalent to the mathematical expression $[1, \text{intermediate_steps}]$.

Setting the variable “intermediate_steps” to e.g. 5 would yield the following result:

- (5, 1)
- (4, 2)
- (3, 3)
- (2, 4)
- (1, 5)

The weight therefore shifts from the start to the end.

Resolving Ties

A last detail of the algorithm is that ties for a position are considered:

1. The scores for each position on the path are calculated such that for each position an ordered list is returned which contains $(a, \text{metric}(a, C, W)) \forall a \in A$.
2. If the same artwork is found at the first index of the result lists of more than one position on the path, it is discarded in all lists except for the one where its score is the highest
3. Repeat step 2 until every first index is unique accross all result lists.

This procedure ensures that the score of the path is maximized. Especially when the dataset is small it happens quite frequently that an artwork scores the highest for multiple positions on the path. Resolving the ties is therefore essential.

4.3.3. Alternative Path-Generation Algorithm

For this thesis a second algorithm for path-generation was tried as shown in the (greatly simplified) pseudo code in figure 4.5.

If the number of intermediate steps is even, then there are two middle spots on the path. How far they are relatively apart depends on the length of said path. The start- and end weights of the middle spots for a path with 4 intermediate steps would be (3, 2) for the left middle spot and (2, 3) for the right middle spot. For a path with 8 intermediate steps the weights would be (5, 4) and (4, 5). If these weights were to be normalized such that one of

4. Implementation

the pair must be 1, then the other weight would be “slightly greater” than one. In the case of a path with 4 intermediate steps this would result in the weight pairs (1.5, 1) and (1, 1.5). For 8 intermediate steps the results would be (1.25, 1) and (1, 1.25). This explains the variable called “slightly_greater_weight” in the pseudo code.

The method called “resolve_ties” in the pseudo code sees to it that the chosen artworks for the two spots are not the same. If there is a tie, the spot where the artwork fits better “gets to keep it” while the other spot is filled by the next best ranking candidate for it.

The method called “update_result_at_correct_position(s)” in the pseudo code simply writes the artwork(s)²⁴ that was / were chosen for the middle spot(s) to the final result. It also assures that chosen artworks will not be considered anymore for recursive calls of the algorithm.

After artworks were chosen, the algorithm is called recursively once for the left side of the remaining path to find the left middle artwork between the just selected artwork and the artwork that this call of the algorithm was given as start artwork. It is called another time in similar fashion for the (right) middle artwork and the artwork that the call of the algorithm was given as end artwork. Since globally resolving ties would be extremely difficult in this algorithm, the order in which these calls are made is randomized to counteract that one side is always evaluated first, as this might lead to better results on that side. This is especially true if not enough good matches exist to fill all the spots, which means that the same artwork would be the best choice for multiple positions.

If the number of intermediate steps is uneven then the algorithm is essentially the same, with the difference that there exists only one middle artwork which is found with the pair of weights (1, 1).

The idea behind this algorithm was to allow the artworks that are found to have an influence on the remaining positions by introducing its own tags into the path-generation. E.g. Assume that in the first step the middle between two artworks, the first of which labeled with the tag “House” and the second of which labeled with the tag “Garden”, is found to be an artwork with the tag “Apple tree”. Then, in the recursive calls of the algorithm, the two middles between “House” and “Apple tree” as well as “Apple tree” and “Garden” would be looked for.

²⁴If the number of intermediate steps is even and two middle spots exists, then there are two results to be written, otherwise there is just one.

4. Implementation

The results were mostly acceptable, but sometimes the algorithm deteriorated far from the start- and/or end-image. The semantic path was also less apparent than in the algorithm described in section 4.3.2. Lastly, the ease with which ties can be globally resolved using the algorithm described in section 4.3.2 proved to be very valuable. Globally resolving ties of recursive calls that are dependent on each others results would require significantly more work. For all the reasons stated in this section the here described algorithm is not used for this thesis.

```
def path_between(start, end, steps):
    if steps%2 == 0:
        middle_left = find_artwork_with_highest_average_similarity_to(start,
                                                                       end, slightly_greater_weight, 1)
        middle_right = find_artwork_with_highest_average_similarity_to(start,
                                                                       end, 1, slightly_greater_weight)
        resolve_ties(middle_left, middle_right)
        update_result_at_correct_position(...)
        left_first = random_zero_or_one == 0
        if left_first:
            path_between(start, middle_left, steps/2)
            path_between(middle_right, end, steps/2)
        else:
            path_between(middle_right, end, steps/2)
            path_between(start, middle_left, steps/2)
    else:
        middle = find_artwork_with_highest_average_similarity_to(start, end, 1,
                                                                1)
        update_result_at_correct_position(...)
        path_between(start, middle, floor(steps/2))
        path_between(middle, end, floor(steps/2))
```

Figure 4.5.: Pseudo Algorithm of alternative path generation algorithm

4.4. Algorithm for Filling empty Spots

Another tool that the curator can use is the “Fill Spots” algorithm. Start- and endpoints are chosen as well as optionally any artwork between them. Some spots will be left empty

4. Implementation

and for those the algorithm provides suggestions. These resulting possibilities are shown in figures 6.7, 6.8, 6.9, 6.10. The algorithm works as follows:

1. The input to this method is a list of artworks and some empty spots. A search space can also be defined. If left undefined, it defaults to all artworks in the collection except the ones contained in the input. For each empty spot a list of weights in accordance to the distances between that empty spot and the known artworks is computed. These are normalized by the length of the input and will consequently always be in the range $]0, 1[$ for a valid input. For the input $[A, B, X, C, D, E]$ where letter A to E represent artworks and X an empty spot, the resulting weights for the spot with the X would be $[\frac{1}{6}, \frac{2}{6}, -, \frac{3}{6}, \frac{2}{6}, \frac{1}{6}]$.
2. In order to strengthen the influence of close artworks over ones that are further away, these weights are then squared. In order to assure that each side weights equally heavy, the sum of the now squared weights preceding or following the empty spots are used to normalize the entries once again: Every weight left of the empty spot gets divided by the sum weights preceding X and every weight right of the empty spot gets divided by the sum of weights following X . As a result, all weights to each side sum up to one and therefore the weighting is balanced. The weights of the previously shown example would therefore be transformed into $[\frac{1}{5}, \frac{4}{5}, -, \frac{9}{14}, \frac{4}{14}, \frac{1}{14}]$.
3. Using these weights, for each artwork in the search space the weighted similarities to all artworks on the path are summed up²⁵
4. For each spot that was left empty in the input a list is returned that contains all artworks in the search space ordered by the afore mentioned sum. As can be seen in figure 6.10 the curator can then scroll through these suggestions to find the ideal artwork.

4.5. Algorithms for the Extraction of Objects from Artworks

Since it is not feasible to extract tags from descriptions of artworks as discussed in section 3.3.1 and tags were already extracted from the titles of artworks and the expert annotations,

²⁵More specifically not the similarities themselves are summed up, but $(1 - \text{similarity})$ is summed up. The reason for this is the same as was described in equation 4.1. In similar fashion also to equation 4.1 a normalization term is applied to the elements before they are added to the sum.

4. Implementation

the only option that is now left is using the images themselves. For this reason a network for object detection is fine-tuned to the art domain. It would also be feasible to extract style information from images, but this remains work for the future.

A particular challenge here is that although pretrained models for object detection and classification exist, they have been pretrained on photographs only. Therefore the task is to fine-tune these models using only a very limited number of annotated samples.

4.5.1. Enhancing Images

In a previous work [15] it was experimentally determined that the best performance in a CNN could be reached by first applying a wavelet denoiser, then sharpen the image and finally apply an ideal contrast. This procedure was also used in this thesis.

As an example the artwork “Alm mit Rindern” by Joseph Heicke is shown in original form (figure 4.6) as well as in the enhanced version of it (figure 4.7).



Image from Belvedere, Vienna, Austria (CC BY-SA 4.0
<https://creativecommons.org/licenses/by-sa/4.0/>)
Artwork seen is “Alm mit Rindern” by Joseph Heicke.



Image from Belvedere, Vienna, Austria (CC BY-SA 4.0
<https://creativecommons.org/licenses/by-sa/4.0/>)
Artwork seen is “Alm mit Rindern” by Joseph Heicke. The image was changed by algorithms.

Figure 4.6.: Artwork “Alm mit Rindern” by Joseph Heicke in its original form.

Figure 4.7.: Artwork “Alm mit Rindern” by Joseph Heicke in a processed form.

4. Implementation

4.5.2. Modelclass, Characteristics and Training

In [16] the problem of object recognition is described as a combination of category recognition and detection. The FasterRCNN model is doing both of these tasks at once and thus is the ideal choice for this thesis. Under ideal circumstances, the PyTorch model “fasterrcnn_resnet50_fpn”²⁶ would have been used, but because of limited resources the much lighter “fasterrcnn_mobilenet_v3_large_fpn”²⁷ was chosen instead.

As suggested by the “TORCHVISION OBJECT DETECTION FINETUNING TUTORIAL”²⁸ the reference implementations were used in training and validation. These contain methods for e.g. evaluation, loss calculation etc. and can be downloaded from a Github repository²⁹. Some of these implementations were modified to fit the task at hand.

For training the Adam optimizer was chosen since it is not only a common choice, but also because it reduces the amount of hyperparameters. Another benefit is that a learning rate optimizer is not needed and thus the choice of such is not required either, further reducing the number of hyperparameters.

While the backbone of the model is pretrained on ImageNet³⁰, the model itself is pretrained on “Coco train2017”³¹. Since the classes that have to be predicted are not in this dataset, the box predictor, which is the final layer that predicts the bounding boxes and classes that are contained within them was then swapped out with a new one. This is necessary since the classes of the dataset of this thesis are not the same ones that the box predictor of the pretrained model can predict.

For the training of a model with a specific configuration, the model is trained using all of the training samples for the number of epochs that was determined to be best for this configuration in the hyperparameter search.

In order to determine whether the datasets that will be used for the final testing (trees and persons) follow a similar class distribution as the box annotated dataset, in table 4.1

²⁶https://pytorch.org/vision/stable/generated/torchvision.models.detection.fasterrcnn_resnet50_fpn.html

²⁷http://pytorch.org/vision/main/generated/torchvision.models.detection.fasterrcnn_mobilenet_v3_large_fpn.html

²⁸https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html

²⁹<https://github.com/pytorch/vision/tree/main/references/detection>

³⁰<https://www.image-net.org/>

³¹Although not an exact name match, the dataset is presumably the one from the website <https://cocodataset.org/#download>

4. Implementation

a closer look is taken at the distributions of the classes in question for the different sets. Note that the numbers specified for training and validation splits are the average of the respective datasets of a multi-label stratified 3-fold crossvalidation. Consequently it could be expected that the values in the columns “Box-annotated dataset”, “Training split” and “Validation split” are identical for each row. The class distribution in regards to trees deviates only slightly (1.65 percent points) between the box annotated dataset and the test dataset, while in regards to persons this difference is moderate (14.58 percent points). If the differences were significant, it would be more difficult to evaluate the results as it would then always have to be considered that the model was trained on a different distribution than it was tested on. In order to verify that the folds used for training and verification are indeed correctly stratified, the class distributions of the individual folds were verified as well in table 4.2.

Situation	Box-annotated dataset	Training split	Validation split	Test dataset
Trees	27.83%	27.83%	27.83%	26.18%
Persons	63.04%	63.04%	63.04%	77.62%

Table 4.1.: Percentages of positive samples in datasets and splits for the classes “Baum” (=tree) and “Person” (=person)

Situation	Training 1	Training 2	Training 3	Test 1	Test 2	Test 3
Trees	28.04%	28.04%	27.39%	27.39%	27.39%	28.04%
Persons	62.06%	63.26%	63.26%	63.26%	62.06%	62.06%

Table 4.2.: Percentages of positive samples in the folds of the multilabel stratified 3 fold cross validation used for training and validation for the classes “Baum” (=tree) and “Person” (=person)

4.5.3. Data Preparation and Augmentations

In a hyperparameter search, the ideal values for the learning rate, the weight decay, as well as the number of trainable layers were determined. Furthermore, different augmentation techniques were evaluated. The ideal batch size could not be determined, since during training the 2 gigabytes of VRAM would only allow a batch size of 1.

4. Implementation

Before the images are passed as input they are resized to a square of 1200 times 1200 pixels using the “LANCZOS” filter implemented in the Python library “PIL”³² and are transformed according to the description in section 4.5.1.

In the next step, many different augmentations were implemented and tested:

- Horizontal and vertical flips with probability p_h and p_v
 - Note: Milani et al. [11] used a horizontal flip with $p=0.5$ as “on-the-fly” augmentation
- Converting the image to grayscale with probability p_g
 - This is thought to be slightly beneficial since there are at least some pencil drawings in the dataset
- Applying brightness, saturation, contrast and hue jitters with ranges b, s, c, h
 - Caneiro et al. [8] describe that in art “visual classes are not consistent in their texture and color patterns”. However, this consistency is implicitly assumed by using a model which is pretrained on ImageNet, which contains photographs. Therefore augmentations with different jitters in the coloring of the image seem highly promising.

The ideal hyperparameters were found using “Optuna”³³, which is a Python framework. This is superior to a common grid search, since the estimator used in this particular set-up (which is a so called “Tree-structured Parzen Estimator”) tries out parameters not purely at random, but considering the results of previous trials. Thus a lot less time is wasted on evaluating configurations that are very unpromising in the first place. In addition to that the framework offers various ways of “pruning”, which means terminating a trial early if it becomes apparent that its result will not be good. With all these tools it then becomes feasible to evaluate all augmentations in combination, which would have been almost impossible with a simple grid search.

A problem that can occur when using this many augmentations is that the learner can be overwhelmed right from the start. A possible solution to this would either be using the larger model (e.g. the one described in 4.5.2), which would presumably be able to

³²<https://github.com/python-pillow/Pillow/>

³³<https://optuna.org/>

4. Implementation

learn more difficult concepts or to use a “Curriculum Learning” [17] strategy. In this case the latter could mean to gradually increase the strength of the augmentations during the learning process. This would have the network learn more general rules over time. For this thesis the values chosen as the hyper parameters for augmentations start with 10% of their set value at the first epoch and gradually increase to full strength until epoch 10. Of course there could be an additional hyperparameter introduced to also control the progression of the curriculum, potentially even at a per-hyperparameter basis, but these experiments remain to be future work.

For evaluating the measures the F1-score³⁴ was used. It has to be noted here, that the validation set is relatively small and thus the variance of this measure is a bit higher than would be ideal. This is solved by using a multi-label stratified 3-fold crossvalidation. The result is in line with the previously made assumptions as jitters for brightness and hue are by far the most important augmentations.

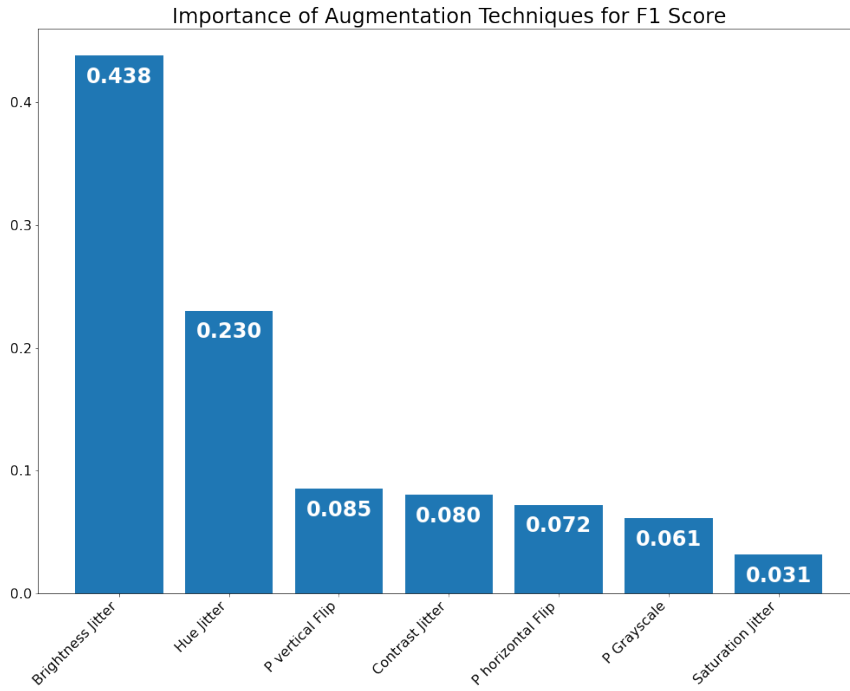


Figure 4.8.: Importance of Augmentations

³⁴Wikipedia [18] defines the F1-Score as “harmonic mean between precision and recall” [18] and states the following formula: $2 * \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$

4. Implementation

Figure 4.8 shows the results of the “fANOVA importance evaluator”³⁵ of the “Optuna” framework and shows how important each augmentation is for reaching a good F1 score.

4.5.4. Making predictions

In the following the parameter called “Non-maximum suppression” (nms) will be relevant. Its effect is displayed in figures 4.9 and 4.10. Essentially, this parameter sets a threshold on $\frac{\text{Intersection}}{\text{Overlap}}$ (=IoU)³⁶ of different bounding boxes to remove those, that are close to each other. This is important since the network produces a large amount of predictions.

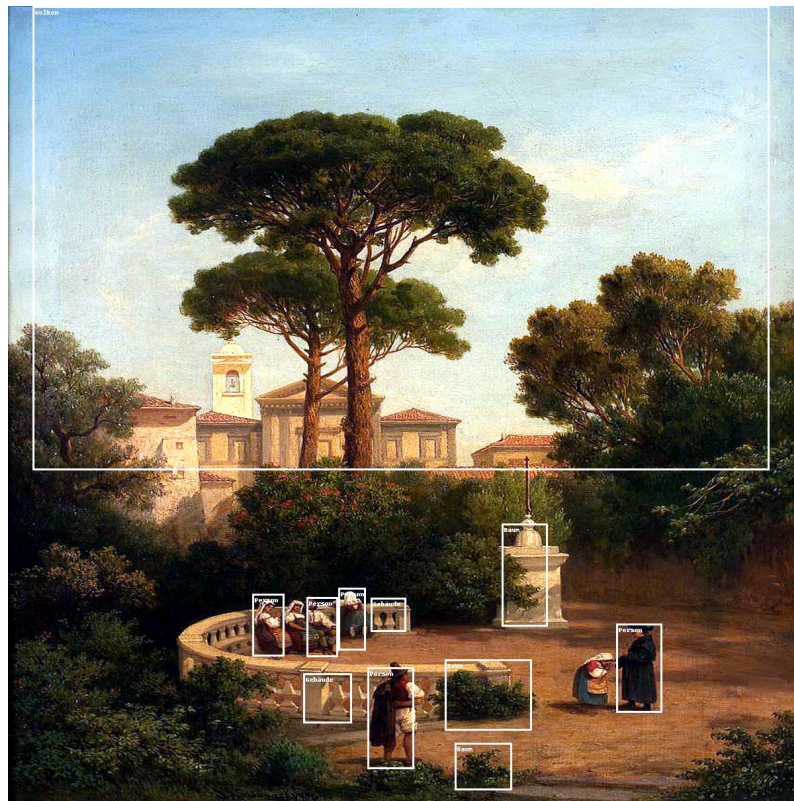


Image from Belvedere, Vienna, Austria (CC BY-SA 4.0
<https://creativecommons.org/licenses/by-sa/4.0/>)
Artwork seen is “Kloster Camalduli” by Jan Novopacky. The artwork was overlayed with bounding boxes and labels.

Figure 4.9.: Bounding boxes filtered with nms=0

³⁵using 10000 trees with max depth = 256

³⁶measures the amount of overlap between bounding boxes

4. Implementation

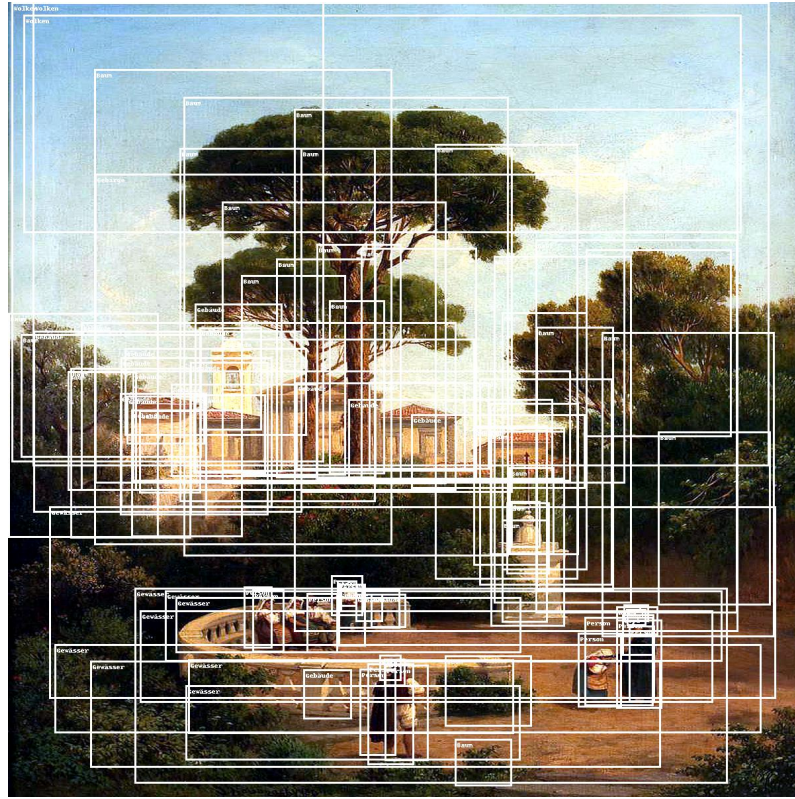


Image from Belvedere, Vienna, Austria (CC BY-SA 4.0
<https://creativecommons.org/licenses/by-sa/4.0/>)
Artwork seen is "Kloster Camalduli" by Jan Novopacky. The artwork was overlaid with bounding boxes and labels.

Figure 4.10.: Bounding boxes filtered with nms=0.5

The final network is used to make predictions about objects contained in the images. The predictions can then be filtered with a non-maximum suppression (nms) as well as a threshold on the score of the prediction³⁷. The final model was used to find the ideal parameters for these two values on the tree test dataset as well as on the person test dataset. While this might be seen as a break of the train/test-split, the aim of this analysis was to find out what results can be expected in the best case. Even though it was clear that the ideal value for each class to predict would differ, the extent of this gap is surprising. While for predicting trees the ideal nms is 0.05 and the best score threshold is 0.35, for predicting persons the ideal nms is 0.35 and the ideal score threshold was 0.45.

³⁷The network returns each bounding box with a prediction of the contained class and a score

4. Implementation

Interestingly it can be observed in table 4.3 that the model which has a box predictor³⁸ that was trained on the whole coco dataset has an ideal nms of 1, meaning that the model performs best without any filtering whatsoever. This suggests that a larger amount of data would be highly beneficial to the performance of the model.

Situation	Ideal Non-maximum suppression	Ideal Score Threshold
Predicting trees (ideal model)	0.1	0.2
Predicting persons (ideal model)	0.5	0.25
Predicting persons (coco model)	1	0

Table 4.3.: Ideal values for non maximum suppression and score thresholds in object predictions

For generating the object labels for the artworks, going forward a nms of 0.1 is used and only the predictions with a score of at least 0.25 were treated as positive. These parameters were chosen based on experiments with the tree test dataset, for which these parameters are almost ideal (see table 4.3). Since it is uncertain whether these parameters are ideal for untested classes as well, the score threshold was increased by 25 % in order to decrease the likelihood of false positives. Although these values are quite different from what was obtained from experiments with the persons test dataset, it has to be kept in mind that the pretrained model was trained to predict the class “person” and consequently it can be expected that the ideal thresholds for this class are quite different.

In figures 4.18 and 4.19 the confusion matrices produced with these ideal values for the final model are shown. In order to further motivate the choice of the final model over the one trained without augmentations, a McNemars test was conducted using the test datasets for trees and persons. While the confusion matrices and F1 scores hint that the augmented model is better in the prediction of both only the difference in the performance of predicting persons is significant (p-value = 0.01) according to the McNemars test. The difference of the performance on the tree test dataset is not (p-value = 0.75). Since the improvement is quite certain for some classes and still assumed for others, the augmented model is the final choice.

Since the class “Person” is also present in the pretrained box predictor another question is of particular interest: How well would a model with a box predictor pretrained only on

³⁸That box predictor needed to be replaced with a predictor that allows predicting the other classes of our dataset as well

4. Implementation

photographs recognize persons in the art dataset? This is shown in the confusion matrix in figure 4.11. For this experiment the ideal values for nms and score threshold were determined for the model that was not fine-tuned. The model trained on photographs (see figure 4.11) performed really well (F1 score of 0.908) and is not that much worse than the model that was fine-tuned to the art domain (F1 score of 0.910 - see figure 4.17). That is not to say that the fine-tuning did not have any significant effect, as other classes profited significantly (see figure 4.20). It has to be considered however that generally persons are in the foreground of paintings and are therefore drawn with a lot of details. Things such as trees are often in the background of paintings and are frequently only hinted at e.g. only with a smear of paint. In the case of the tree class it can therefore be expected that the difference between the performance of the models would be a lot greater.

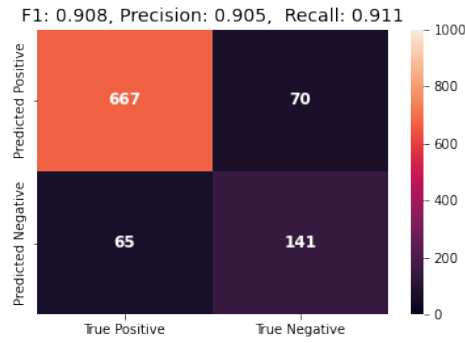


Figure 4.11.: Unchanged Coco Model Confusion Matrix for predictions of persons with ideal parameters

In figures 4.12, 4.13, 4.14, 4.15 the models with and without augmentations are compared using the tree test dataset as well as the person test dataset. For these figures note that the F1 score, precision and recall refer to the respective test dataset.

Note that the numbers in the confusion matrices do not sum up to all 1000 entries in the dataset. This is because all artworks that are also in the box-annotated dataset were excluded as this would have resulted in an overlap between training and test samples otherwise.

In order to also show the best possible results in figures 4.16, 4.17, 4.18, 4.19 the same confusion matrices are shown with the ideal values as defined by table 4.3 set for predicting each class.

4. Implementation

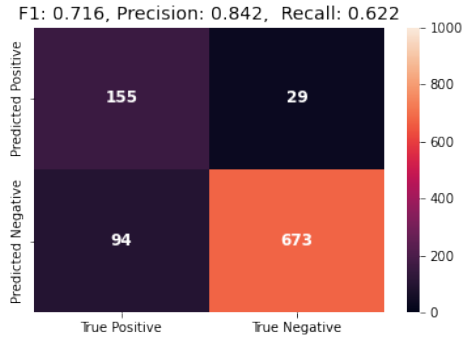


Figure 4.12.: Confusion Matrix: Tree Test Dataset, No Augmentations

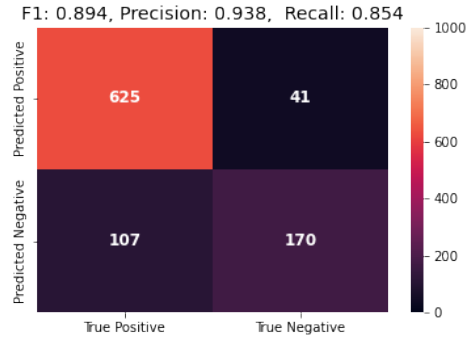


Figure 4.13.: Confusion Matrix: Person Test Dataset, No Augmentations

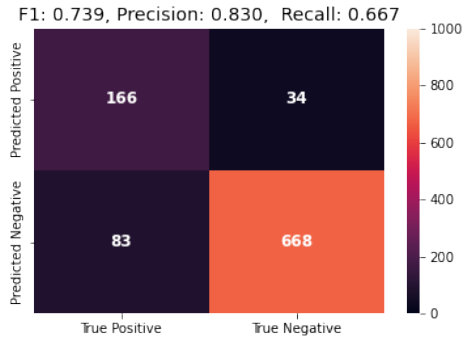


Figure 4.14.: Confusion Matrix: Tree Test Dataset, with Augmentations

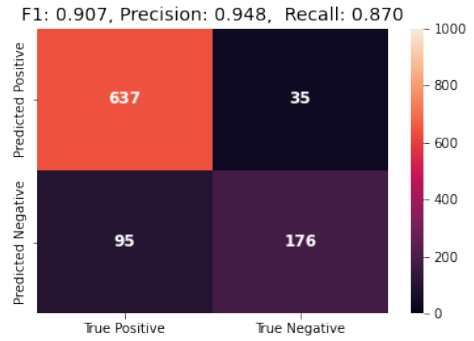


Figure 4.15.: Confusion Matrix: Person Test Dataset, With Augmentations

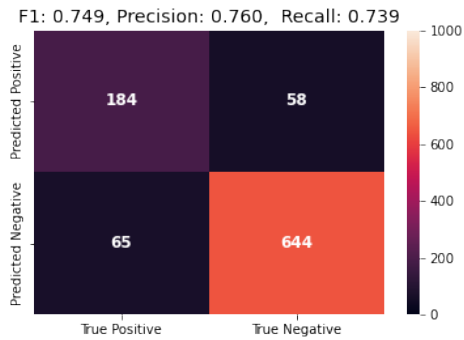


Figure 4.16.: Confusion Matrix: Tree Test Dataset, No Augmentations; Ideal threshold values

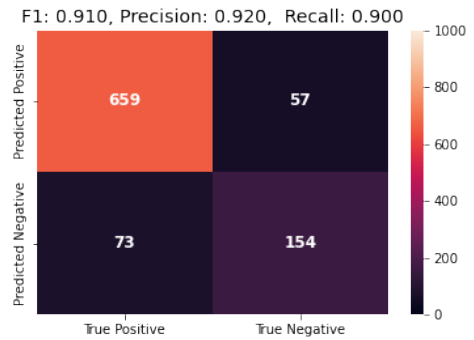


Figure 4.17.: Confusion Matrix: Person Test Dataset, No Augmentations; Ideal threshold values

4. Implementation

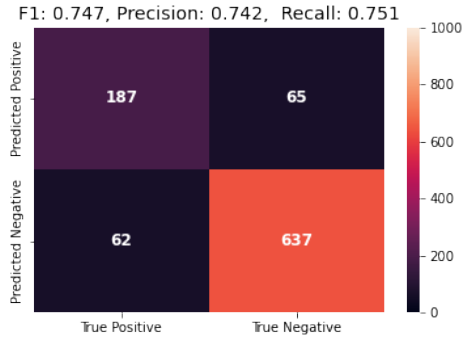


Figure 4.18.: Confusion Matrix: Tree Test Dataset, with Augmentations; Ideal threshold values

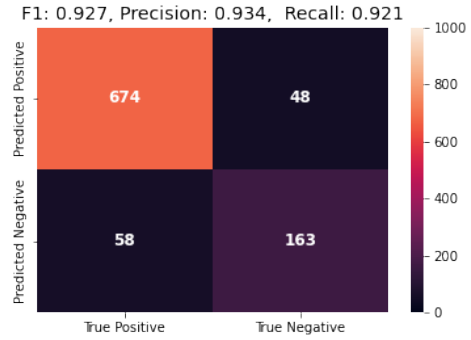


Figure 4.19.: Confusion Matrix: Person Test Dataset, With Augmentations; Ideal threshold values

4.6. Evaluation of Object Detection

In order to rate the potential of object detection for enhancing the already existing iconclass annotation, a closer look was taken at the overlap between the ground truth represented by the tree annotation dataset and the iconclass annotations. For the latter all iconclass tags of the dataset that are related to trees are taken into account. Namely those are “Wald”, “Baumgruppen”, “Laubwald”, “Bäume”, “Lichtung im Wald”, “Waldrand”, “Waldweg”. In order to further examine to what extent this potential was used by our model, the final predictions are compared to the ground truth annotations as well. Table 4.4 shows how well the ground truth from the tree dataset containing an indicator for the presence (or absence) of trees in artworks could be predicted.

The results, which are shown in table 4.4 are a good indicator that the proposed procedure has worked. Compared to the F1 Score of evaluating the Iconclass labels (0.32), the F1 score of finding trees with the object detection is significantly better (0.74 or 0.75 in the case of the fully optimized parameters). This means that it is a promising prospect to extend the object detection in the future work with classes that are not as well represented in the iconclass labels as trees are.

In table 4.4 the following predictors are used:

- Iconclass labels

4. Implementation

- Predict “True” if “Wald”, “Baumgruppen”, “Laubwald”, “Bäume”, “Lichtung im Wald”, “Waldrand” or “Waldweg” is in the iconclass labels of an artwork. Predict “False” otherwise
- Final object labels
 - Predict “True” if the object recognition assigned the tag “Baum” to the artwork. Predict “False” otherwise.
- Objects, Ideal parameters
 - Predict “True” if the results of the object recognition filtered with parameters ideal for the class “Baum” assigned the tag “Baum” to the artwork. Predict “False” otherwise.
 - This “Objects, Ideal p.” row is mainly relevant for section 4.5.3.

Predictor	True positive	False Positive	True negative	False negative	F1
Iconclass labels	48	0	702	201	0.32
Final object labels	174	50	652	75	0.74
Objects, Ideal p.	187	65	637	62	0.75

Table 4.4.: “Confusion Matrices” of iconclass annotations and final object labels

Lastly, a closer look was taken at the extent to which the amount of data influences the quality of predictions. For this purpose the F1 scores of models trained on 0%, 10%, 20%, ... 100% of the available data were recorded. The training was done using the non-maximum suppression and score threshold values that were found to be ideal for the individual test sets in section 4.5.4. In order to decrease the variance in the result 5 models were trained for each step, each with another one of 5 differently shuffled datasets. Their results were then averaged. The procedure was done for the tree-annotation test dataset as well as for the person-annotation test dataset. The results are shown in figures 4.20 and 4.21. Interestingly it can be seen that the performance on the person dataset gets saturated quite early. A possible explanation for this is that in the Coco dataset “Person” is one of

4. Implementation

the classes, but the perhaps closest match to a tree is the class “potted plant”. It would perhaps also explain, why the scores related to the person detection go much higher. This does promise that more data could improve the results for classes that are not contained in the Coco dataset.

In the case of the tree dataset the performance also stagnates towards the end. A possible explanation could be the quality of the box annotations, as this class was difficult to annotate (see section 3.5).

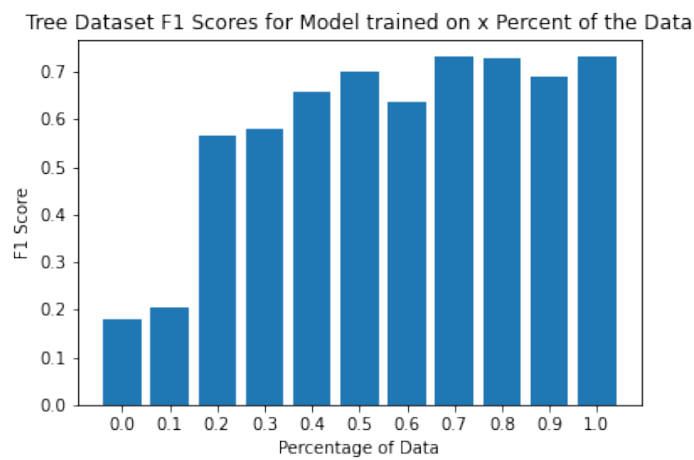


Figure 4.20.: F1 scores of model trained with x percent of dataset - Tree Test Dataset

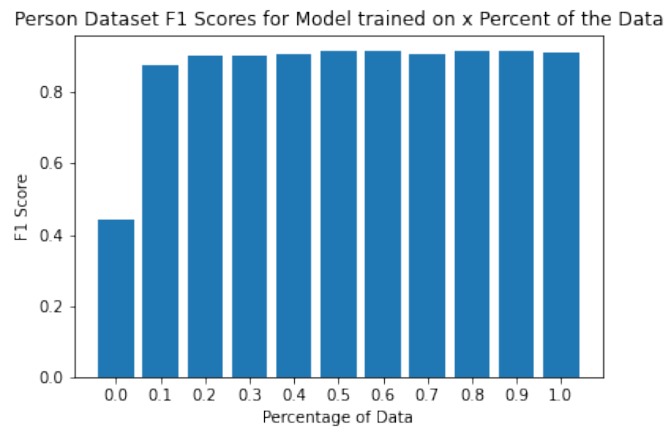


Figure 4.21.: F1 scores of model trained with x percent of dataset - Person Test Dataset

5. Results

5.1. Path-Generation

In this section the results of the process introduced in section 4.3 are presented. In section 5.1.1 semantic tags will be used for the matching. This means that only those sources of tags are considered that focus on semantic context. In particular this means that only tags from textual sources (titles and expert annotations) are considered. In section 5.1.2 it will be briefly explained why matching based on visual tags alone does not yield good results. “Visual tags” in this context means tags extracted from the images themselves by using the object detection described in section 4.5. Finally, in section 5.1.3 some results of generating paths using both types of tags combined will be shown.

5.1.1. Semantic Matching

In order for an artwork to be considered in this path-generation process it had to have at least 3 sources of tags. This means either having a title from which tags were extracted as well as at least 2 expert annotations or alternatively having at least 3 expert annotations. This leaves 2772 out of 4135 artworks to consider for the path-generation process.

Note that in the following examples individual tags are separated by commas and therefore e.g. “Fuhrwerk Fracht Wagen Fracht Karren” is one single tag. The reason that this tag reads like a list of related words rather than a precise label is the data cleaning process. In this particular case the source of this tag was the **single** expert annotation “Fuhrwerk, Frachtwagen, Frachtkarren”.



Individual images seen on this path are all from Belvedere, Vienna, Austria (CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0/>)
 Artworks from left to right: "Der Karrengaul" by Emanuel Franz Hegenbarth, "Pflügender Bauer" by Otto von Thoren, "Fünf Ackerpferde" by Johann Baptist Michael Dallinger von Dalling, "Bergige Landschaft mit Kühen" by Franz Xaver von Pausinger, "Kühe und Schafe" by Unknown Artist, "Frühlingsweide" by Giovanni Segantini, "Alm mit Rindern" by Joseph Heicke

Figure 5.1.: Example 1 of a Path resulting from semantic Matching only



Individual images seen on this path are all from Belvedere, Vienna, Austria (CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0/>)
 Artworks from left to right: "Zwei Zugpferde im Stall" by Johann Baptist Michael Dallinger von Dalling, "Kuhstall" by Michael Neder, "Der Karrengaul" Emanuel Franz Hegenbarth, "Reitergefecht" by Francesco Casanova, "Pont de Moutiers in Thiers" by Rudolf Ribarz, "Der Schriftsteller Jean Baron de Bourgoing" by Edmund von Hellmer, "Minister Vinzenz Graf Baillet de Latour" by Alois Johann Josef Delug

Figure 5.2.: Example 2 of a Path resulting from semantic Matching only

5. Results

Semantic tags of path shown in figure 5.1:

- S Karren Gaul, Fuhrwerk Fracht Wagen Fracht Karren, Pferd
- 1 pflügen, Pflug Bauer, Pferd
- 2 tief Horizont, Wolken, Herde Schafherde Rinderherde, Fünf Acker Pferd
- 3 Schaf, Ziege, Pferd, Kuh, bergige Landschaft, Rindvieh, Herde Schafherde Rinderherde
- 4 Schafe, Kuh, Vieh tränken füttern, Bäuerin, Kühe, Herde Schafherde Rinderherde
- 5 Berg Gebirge, Frühling Weide, Kuh, Wiese Weideland
- E Bergwiese Alm, Rindvieh, Wolken

The transition here is very smooth and in the middle artwork horses as well as cattle are to be found. It is also interesting that the settings of the artworks move from the plains more and more into the mountains.

Semantic tags of path shown in figure 5.2:

- S Innenraum, Zwei Zugpferde, Pferd, Stall Schuppen Hütte Zwinger, gedeckt Sattel oder Sitz auf der Rücken ein Reittier
- 1 Innenraum, Kuhstall, Stall Pferch
- 2 Karren Gaul, Fuhrwerk Fracht Wagen Fracht Karren, Pferd
- 3 auf einer Pferd Esel oder Maultier reiten Reiter Reiterin, Schlacht, Reiter Gefecht
- 4 Pont de, Auvergne Region, Thiers, Frankreich
- 5 Jean de Baron, Schriftsteller Jean Baron de, Schriftsteller Porträt
- E lesen, Minister Vinzenz Graf de Latour, de Latour Graf, Minister als Mitglied einer Regierung

Here the language model seemed to have picked up on the french name in the end artwork. The middle artwork shows a person riding a horse, which brings the start and end artwork perfectly together.

5. Results

5.1.2. Visual Matching

The starting point in figure 5.1 is a good example for the difficulties in purely visual matching. In the predictions shown in figure 5.3 it can be seen that the floor is recognized as water as the region looks like it could be waves. Most of the trees were found, even though they are only hinted at in the background. The wheel are misclassified as a person. Naturally, higher amounts of noise degrade the performance of the algorithm. Although paths can be created using *only* visual information, the results are not very meaningful.

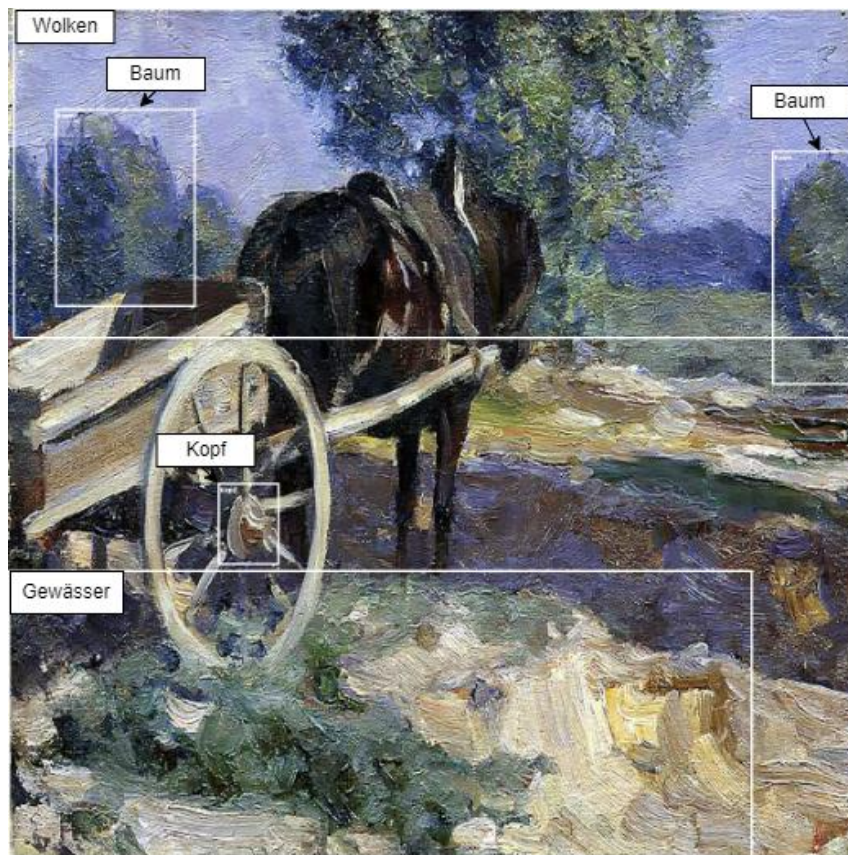


Image from Belvedere, Vienna, Austria (CC BY-SA 4.0

<https://creativecommons.org/licenses/by-sa/4.0/>)

Artwork shown is "Der Karrengaul" by Emanuel Franz Hegenbarth. The artwork was overlaid with bounding boxes and label as well as boxes and arrows describing the classes of the bounding boxes.

Figure 5.3.: Detected Objects for the artwork "Der Karrengaul" by Emanuel Franz Hegenbarth; Predictions made with Augmented model, nms=0.1, score threshold=0.25

5. Results

5.1.3. Semantic and Visual Matching combined

As already mentioned in section 5.1.2 the examples at hand are not perfect. However, this is a good reason to showcase them. For better comparability the matching is done using the same 2772 artworks that were considered section 5.1.1. In contrast to section 5.1.2 in this section the tags generated by the object recognition are considered in this section as well.



Individual images seen on this path are all from Belvedere, Vienna, Austria (CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0/>)
 Artworks from left to right: "Der Karrengaul" by Emanuel Franz Hegenbarth, "Hundekopf" by Walther Gamerith, "Waldtümpel" by Adolf Gustav Ditscheiner, "Hellgrüne Wiese gegen dunkle Walddlière" by Otto Friedrich, "Weide" by Ludwig Sigmundt, "Frühlingsweide" by Giovanni Segantini, "Alm mit Rindern" by Joseph Heicke

Figure 5.4.: Example 1 of a Path resulting from semantic and visual Matching



Individual images seen on this path are all from Belvedere, Vienna, Austria (CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0/>)
 Artworks from left to right: "Zwei Zugpferde im Stall" by Johann Baptist Michael Dallinger von Dalling, "Stier im Stall" by Edmund Mahlknecht, "Zwei Köpfe" by Rudolf Wacker, "Totes Schneehuhn" by Johann Matthias Ranftl, "Victor-Henri Marquis de Rochefort-Luçay" by Auguste Rodin "Der Schriftsteller Jean Baron de Bourgoing" by Edmund von Hellmer, "Minister Vinzenz Graf Baillet de Latour" by Artist:Alois Johann Josef Delug

Figure 5.5.: Example 2 of a Path resulting from semantic and visual Matching

5. Results

Tags of path shown in figure 5.4(Tags that were generated by the object detection are written in bold):

S Karren Gaul, Fuhrwerk Fracht Wagen Fracht Karren, Pferd, **Wolken, Baum, Kopf, Gewässer**

1 schlafende Tiere, Kopf ein Tier, Hundekopf, **Wolken**

2 Teich Tümpel, Baum, Wald Tümpel, **Gewässer**

3 Dunkelheit, Wiese Weideland, Waldrand, **Wolken**

4 Hügellandschaft, Weide, Wiese Weideland, **Wolken**

5 Berg Gebirge, Frühling Weide, Kuh, Wiese Weideland, **Wolken**

E Bergwiese Alm, Rindvieh, **Wolken, Gewässer, Gebäude**

- Note: Since this artwork is one of the 250 contained in the box-annotated dataset the object tags of this particular identifier were not generated by the model, but instead taken from the ground truth.

Tags of path shown in figure 5.5(Tags that were generated by the object detection are written in bold):

S Innenraum, Zwei Zugpferde, Pferd, Stall Schuppen Hütte Zwinger, gedeckt Sattel oder Sitz auf der Rücken ein Reittier, **Person, Kopf, Baum**

1 Stall Pferch, Stall Schuppen Hütte Zwinger, Stier, Blickkontakt, Innenraum, **Person**

2 Zwei Kopf, Kopf männliche Figur, Kopf weibliche Figur

- Note: The tag “Kopf” that would have been also generated by the object recognition is filtered out because “Kopf” is already fully contained in other tags (see section 4.1.3).

3 Totes Schnee Huhn, Schnee Hühner, ein sterbend Tier Tod ein Tier ein tot Tier, **Kopf**

4 nach unten schauen, Rochefort Henri, Victor Henri Marquis de, Schriftsteller Porträt, **Kopf**

5 Jean de Baron, Schriftsteller Jean Baron de, Schriftsteller Porträt, **Kopf**

5. Results

E lesen, Minister Vinzenz Graf de Latour, de Latour Graf, Minister als Mitglied einer Regierung, **Kopf, Hand, Wolken**

It is apparent that the paths lack the certain smoothness that could be achieved with using only semantic tags. In these particular cases the fault lies in the wrongful detection of persons and bodies of water in the input images. In order for a path to be of high quality, false positives should be avoided in the object detection. Future work might put a greater emphasis on this aspect.

A further drawback of the current implementation is that the objects that are detected are not very interesting: Trees, building, clouds, etc. are detected simply because they appear often in the box-annotated dataset and were therefore not discarded as described in section 3.5. The results will consequently be more nuanced when the number of detected classes can be increased.

5.2. Using the Iconclass System for creating a Measure of Success

In figure 3.1 an example of the iconclass hierarchy was shown, In this example we walked down the main category “Society, Civilization, Culture” all the way to the category “housing”. When in the following the phrase “aggregating iconclass tags to level x” is used, it means that this way is walked into the opposite direction. E.g. If an artwork is assigned the tag “housing” then aggregating it to level one would result in “Society, Civilization, Culture”.

Finding the ground truth to how similar artworks truly are is an impossible task, given that this concept is highly subjective. Therefore, a measure was created for which it is reasonable to assume that the quality indicated by this measure is positively correlated with the quality of the semantic path (see section 5.1.1) that it analyses.

The basic idea behind the measure is the following:

- Generate path (see section 5.1) using tags of some (combination of) origin(s)
- Aggregate all iconclass tags associated to the individual artworks on the path to a given level of the iconclass hierarchy

5. Results

- Order the categories of the iconclass system according to the following:
 - $C :=$ The set containing all iconclass categories in the form of their vector representation (see chapter 2.1) aggregated to a specified level which were assigned to at least one artwork of the path
 - $C_s :=$ The set containing all iconclass categories in the form of their vector representation aggregated to a specified level with which the start artwork is labeled. $C_s \subset C$
 - $C_e :=$ The set containing all iconclass categories in the form of their vector representation aggregated to a specified level with which the end artwork is labeled. $C_e \subset C$
 - $|C_s| :=$ The number of categories with which the start artwork is labeled
 - $|C_e| :=$ The number of categories with which the end artwork is labeled
 - $s_{c_x}^{c_y} :=$ The cosine similarity of c_x to c_y , whereas $s_{c_x}^{c_y} = s_{c_y}^{c_x}$ and $s_{c_x}^{c_y} \in \mathbb{R}, \in [-1 (= \text{low similarity}), 1 (= \text{high similarity})]$

Then the categories are ordered according to the metric: $\sum_a^{C_s} (\frac{s_a^x}{|C_s|}) - \sum_b^{C_e} (\frac{s_b^x}{|C_e|})$ where x is a specific category of C . Essentially this sorts the categories such that those are on top that express high similarity to the start artworks categories and low similarity to the end artworks categories. The same is also true the other way around, therefore the transition of the categories is smooth between the start and the end.

- Plot the result as a heatmap.

This visualization technique was vital in analyzing the continuity of individual paths during development. While humans can evaluate paths even without such a measure, one tends to see connections between images even when there are none, or at least none that were known to the algorithms. Furthermore, it becomes increasingly difficult for a human to verify the results of the algorithms, the more tags are associated with the artworks on the path.

5. Results

An example can be seen in figure 5.6³⁹ where the path shown in figure 5.1 is shown through the described measure.

³⁹Note that in the title the figure says “[...] level 5 (or higher)”, this is because the algorithm defaults to the finest available level of the iconclass tree that is available for a tag if the specified level on the tree is not available for an iconclass tag. The labels on the x-axis are the identifiers of the artworks in the dataset.

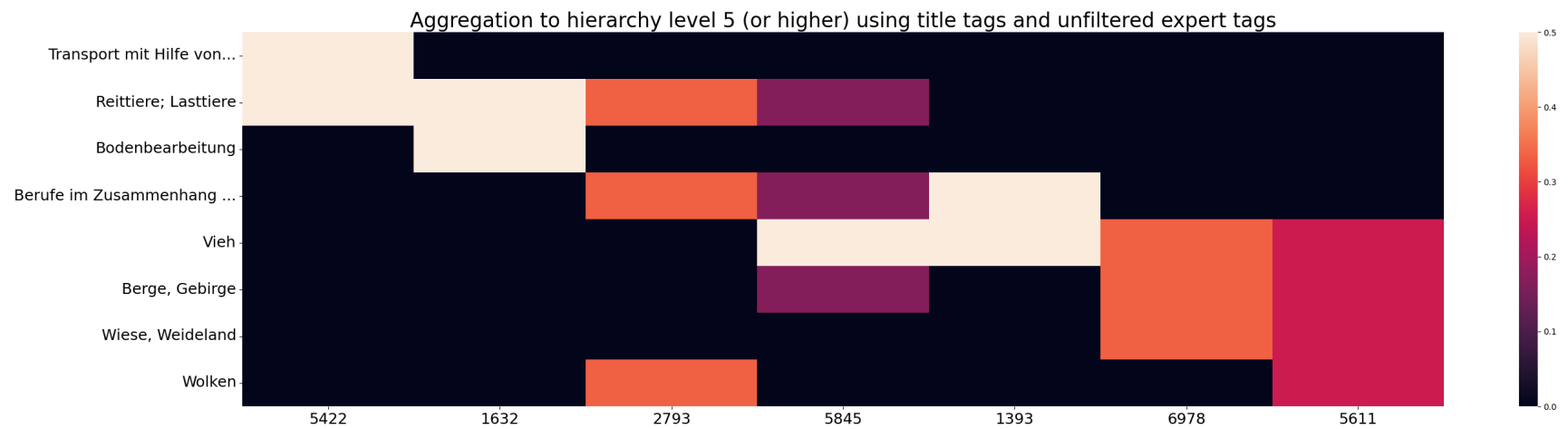


Figure 5.6.: Iconclass comparison using text labels extracted from expert tags and title tags

5. Results

Note that for this analysis it can be meaningful to exclude iconclass tags from the matching process, as otherwise this measure might be biased: If iconclass tags had a high influence on the creation of the semantic path, then this measure will show a good result. This procedure is then useful for determining the quality of the extracted tags: If the tags of some source would lead the algorithm astray, then no rough diagonal between the bright spots of the start- and end artwork would be visible in figure 5.6.

The analysis can further be done only using iconclass tags for the matching process. The resulting graphic would then show the quality of the semantic path algorithm, as the tags for creating the path are exactly the same as shown in the measure.

Unfortunately, there is no practical way to use this procedure in order to generate a quantitative analysis, which is why a different algorithm for that purpose is described in section 5.3.

5.3. Quantitative Analysis

In order to quantify the success of the thesis, a quantitative analysis has to be done. Since the goal is to evaluate the similarity calculations, this very same similarity cannot be part of the metric. Therefore the metric is designed as follows:

1. Filter out all identifiers which are associated with less than 2 iconclass tags. This leaves 1618 artworks.
2. Randomly pick 500 unique pairs from these identifiers, whereas each identifier must be unique within a pair. These 500 pairs together contain 735 unique identifiers.
3. Exclude iconclass tags from having any influence in the matching processes. Only those Expert tags are still being used that are from different labeling systems than Iconclass. Thus, a significant number of tags are dropped. This will leave the filtered artworks with a total of 6928 tags of the original 11422⁴⁰.
4. For each pair generate a path (see section 4.3.2) consisting of the first identifier of the pair, a specific number of artworks in between and the second identifier of the pair.

⁴⁰15745 of 20239 if object tags are considered as well

5. Results

5. For each identifier on a specific path calculate the average word vector of its iconclass tags, aggregated to the top most iconclass level(s)
6. Using these vectors, calculate the cosine distance of each⁴¹ artwork on a path to the start and end image.
7. Using these distances, calculate ratios $\frac{\text{Distance to start}}{\text{Distance to end}}$.
8. Save the indexes of the artworks on a path sorted by these ratios. A perfect path therefore would result in e.g. 01234.
9. Using the number of necessary swaps of adjacent elements as a metric calculate a score for these 250 paths by comparing the result of the above calculations to the ideal result. Figure 5.7 shows how an actual result might look compared to the ideal result and figure 5.8 illustrates the calculation of the metric for that example step by step.

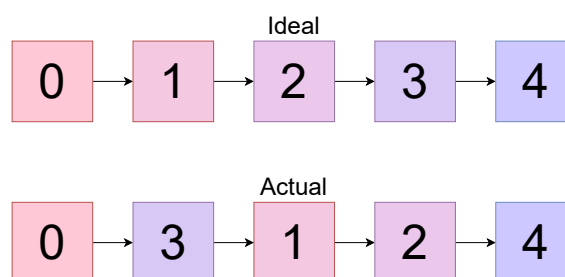


Figure 5.7.: Illustrative example of the metric: Ideally the order would be strictly ascending, the actual result however can be different.

⁴¹That includes the start and end artworks

5. Results

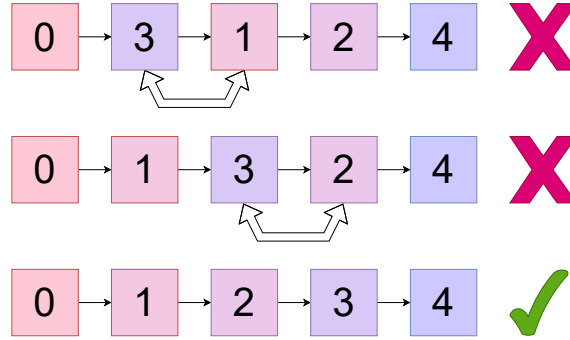


Figure 5.8.: An algorithm calculates how many times neighboring elements have to be swapped in order to transform the actual result into the ideal one.

For each pair of start and end points the paths having 3 to 23 artworks in between the start and end point were evaluated⁴² with the above algorithm. In figure 5.10 the results can be seen. Note that for the random baseline 1000 “paths” were generated by randomly picking artworks for each position. It can be seen that the object tags lead to slightly worse results altogether. On the one hand this was to be expected since the predictions are noisy and the labels that are generated by the object recognition are very general and therefore not rich in semantic information⁴³. On the other hand it should also be taken into consideration that the result might only be worse because the object recognition covers concepts not considered by the Iconclass ground truth.

The average result (across all path lengths) of using object tags (30.04) is approximately 11 percent worse than the result without the usage of such (27.06). It has to be considered here that tags generated from object recognition make up 8817 of the 15745 tags that were considered. Since they are therefore by far the most influential source for the matching processes, these 11 % are still within reason. It is interesting that the object tags degrade the performance even though the measured F1 scores are fairly high considering the limited resources that were available for training the object recognition. However, it can be seen in figure 5.9 that the more difficult the task becomes with increasing path length, the smaller the relative difference in errors becomes⁴⁴. Hence, for more difficult tasks the object tags might be beneficial after all. At this point it is again important to mention that the aim of

⁴²= paths of length 5 to 25

⁴³As already described in section 3.5 the more interesting classes would require a larger dataset in order for these classes to be found in a sufficient number of artworks.

⁴⁴This is despite the absolute distance increasing with path length, as can be seen in figure 5.10

5. Results

generating tags from visual sources is not to increase the performance of this algorithm, but to have more tags available to form connections between artworks without having to hand annotate every artwork.

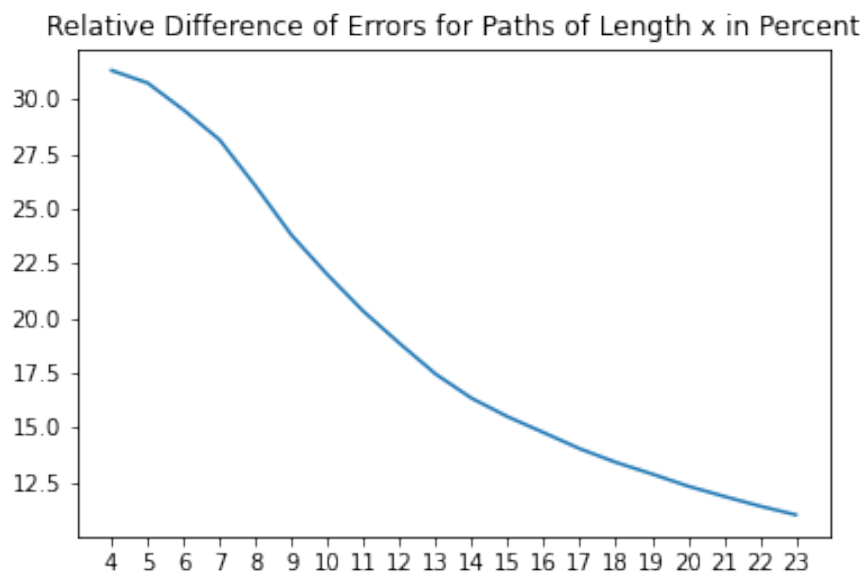


Figure 5.9.: Relative Difference of Errors for Paths of Length x in Percent

5. Results

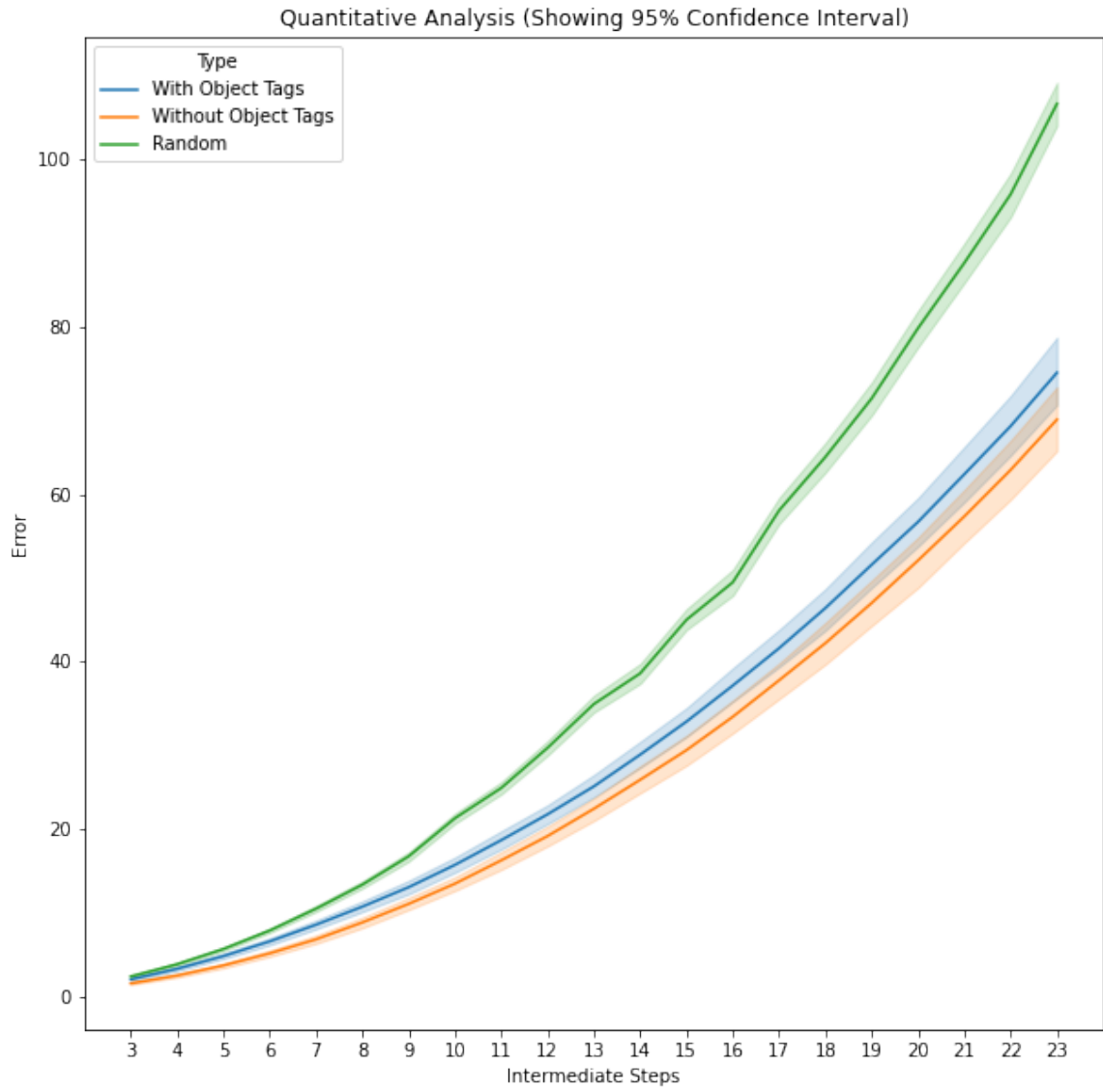


Figure 5.10.: Quantitative Analysis of Generating Paths of different lengths

6. User Interface

It is important to actually see the artworks when putting the software to work. For this reason a graphical user interface was created. The GUI library used for this thesis is called “tkinter”⁴⁵. There would be more modern alternatives, especially when also taking web interfaces into account, but good optics are not a requirement at this point and staying within the Python programming language avoids a lot of software development work. This choice will also make the program easier to update in the future. The current implementation of the GUI is really not a work of art itself, as it is only intended as a proof of concept.

- In figure 6.1 the different options are shown to the curator.
- In figure 6.2 a screen is shown that provides information about the artwork. It can be reached from the menu by clicking “Artwork Info” or in many other parts of the program by just clicking on an artwork.
- In figure 6.3 the curator can input specific requirements for any artworks that should be considered for any algorithms. Artworks can be filtered according to the following criteria:
 - Artwork must have minimum / maximum similarity to (a) search term(s)
 - Artwork must (not) be created by artist(s)
 - Artwork must have title
 - Artwork must be created before / after year
 - Artwork must have at least a certain number (of sources) of tags
 - Each source of tags can be enabled or disabled

⁴⁵<https://docs.python.org/3/library/tkinter.html>

6. User Interface

- In figure 6.4 the artwork “Abend am Mittenwaldsee” by Mathilde Sitta-Allé (which can also be seen in figure 6.2) was selected and the artworks seen in the figure are recommended as being similar.
- In figure 6.5 the entry mask for creating a semantic path is shown. If a question mark is clicked, a view similar to what can be seen in figure 6.4 is opened from which an artwork can be selected.
- In figure 6.6 the resulting semantic path can be seen. The result shows nuances between the given start and end points. By changing the amount of artworks in between the granularity of these nuances can be controlled.
- In figure 6.7 the first step of the “Fill Spots” functionality is shown, in which the number of artworks must be entered.
- In figure 6.8 the entry mask for the “Fill Spots” functionality is shown. Here any of the questionmarks can be clicked and artworks can be chosen for these specific spots. (In any case the start and end artwork must of course be given.) For all spots that are then left, the algorithm suggests artworks.
- In figure 6.9 some of the final input can be seen. (There is an additional artwork given as input at the rightmost spot, that is not displayed in the figure, as this would require scrolling to the right.)
- In figure 6.10 the result of the “Fill Spots” algorithm can be seen. The curator can scroll through suggestions which are sorted by their rank. In contrast to the semantic paths algorithm described above, here the same artwork can appear at multiple spots. That the ties must then be resolved manually by the curator⁴⁶ is a natural and intended side-effect of presenting said curator with all available options for each spot.

⁴⁶Instead of being resolved automatically as it is done in the semantic path algorithm

6. User Interface

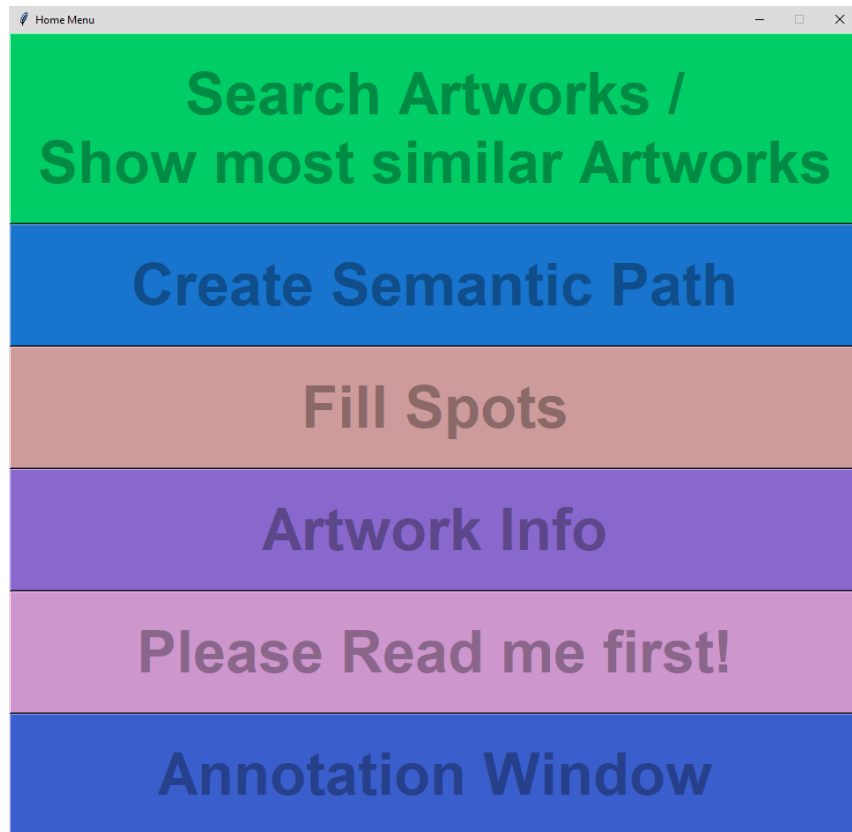


Figure 6.1.: The start menu of the User Interface

6. User Interface

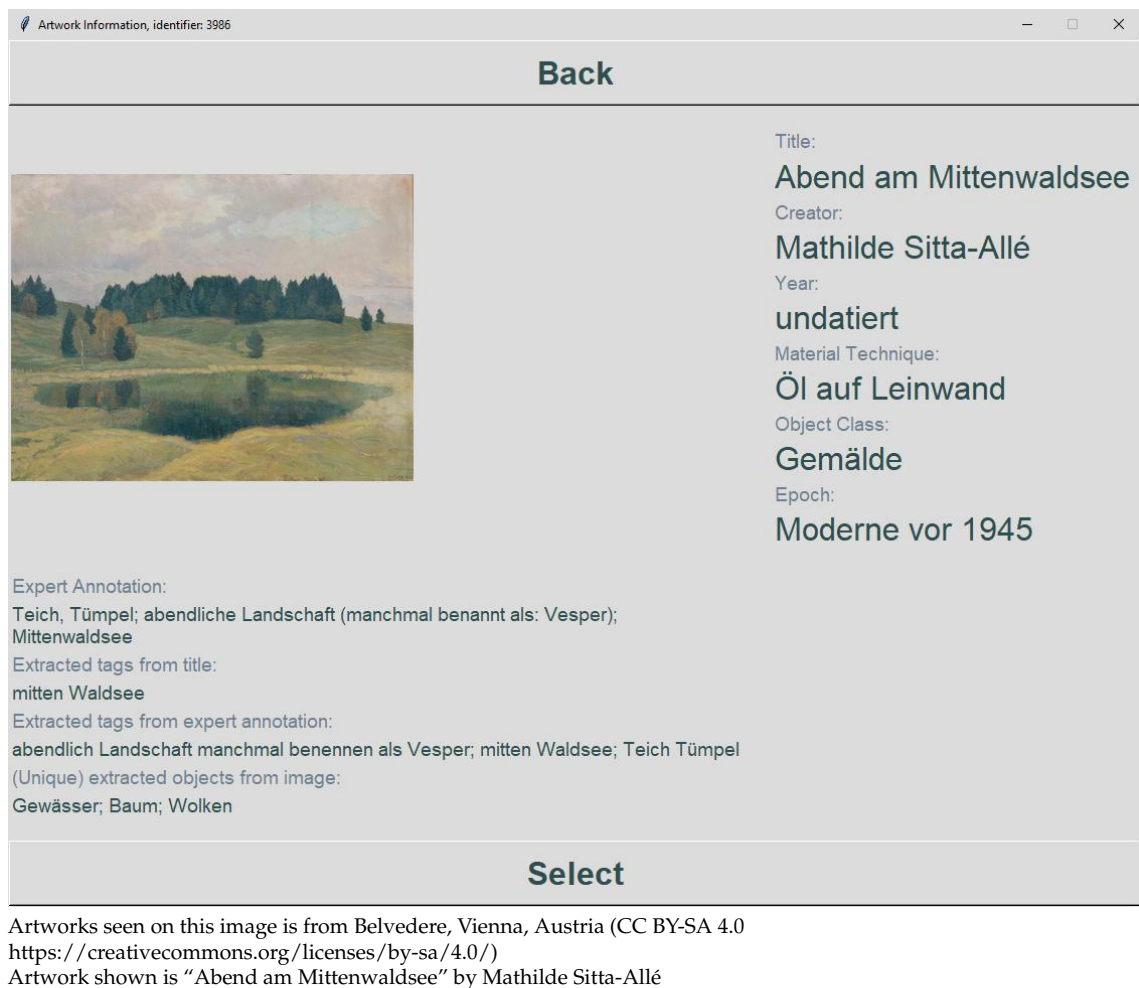


Figure 6.2.: The info menu

Set Filter Menu

Set Filter Menu

Search term:

Artist:

Title:

Start year:

Enter text

Enter the name of the artist

Enter a specific title

Enter the start year

Minimum required Similarity
☒

From artist
☒

End year:

Minimum number of tags
☒ Tags/Sources
☒ Min/Max
3 active filters:

☒ Consider title tags
☒ Consider expert tags
☒ Consider image tags

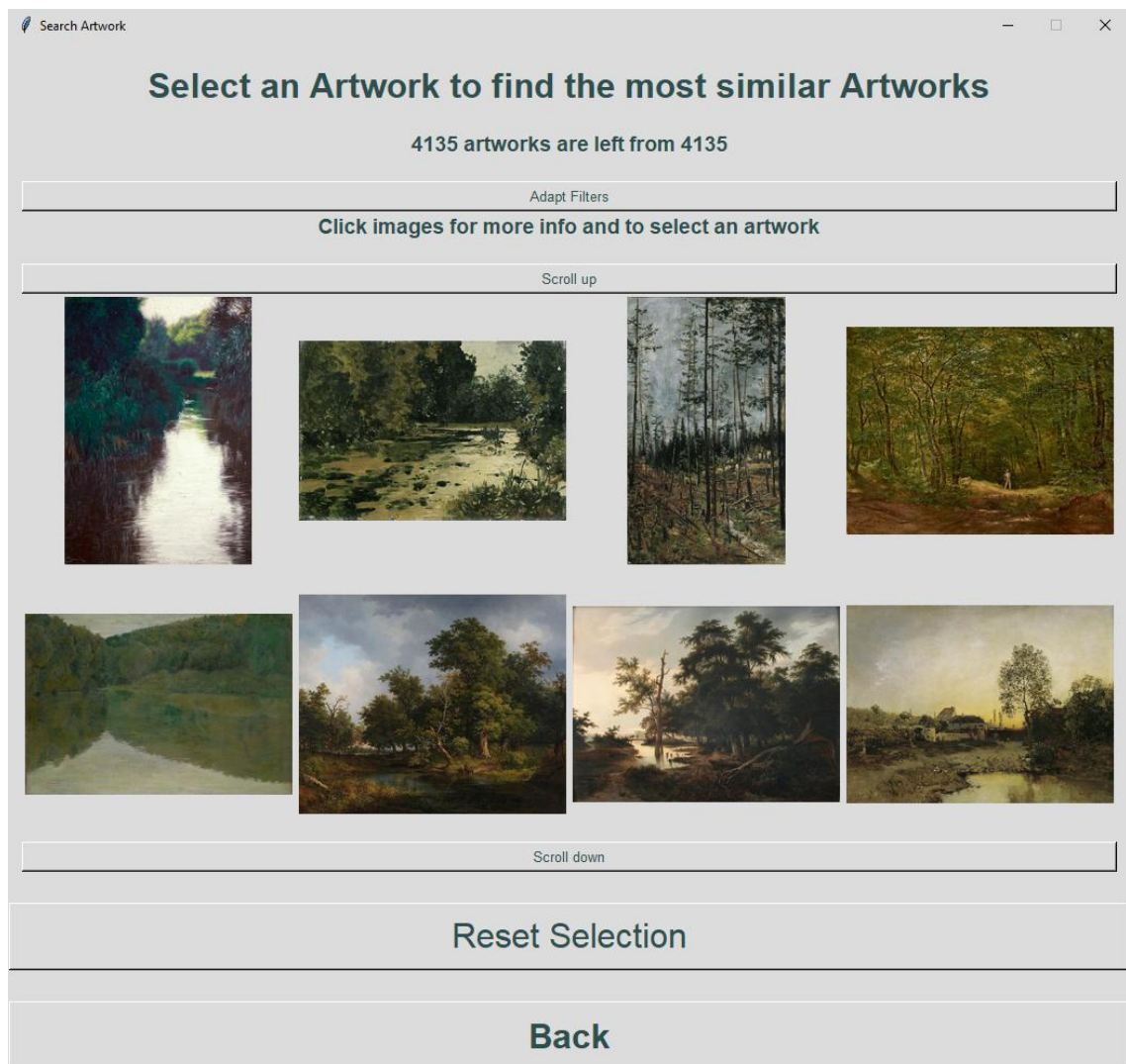
Apply Filter

2 artworks are left from 4135

Apply and return

Figure 6.3.: The Filter Menu

6. User Interface



Artworks seen on this image are all from Belvedere, Vienna, Austria (CC BY-SA 4.0

<https://creativecommons.org/licenses/by-sa/4.0/>)

Artworks from left to right and from top to bottom: "Weiher" by Wilhelm Bernatzik, "Waldtümpel" by Adolf Gustav Ditscheiner, "Waldschlag bei Gödöllö" by Theodor von Hörmann, "Spaziergänger in einer Waldlandschaft" by Friedrich Loos, "Der stille Teich" by Friedrich König, "Waldlandschaft" by Josef Feid, "Waldlandschaft" by Anton Altmann d.J., "Landschaft mit Weiher und Gehöften" by Robert Russ

Figure 6.4.: View after selecting the artwork "Abend am Mittenwaldsee" by Mathilde Sitta-Allé in the "Show most similar artworks" window

6. User Interface

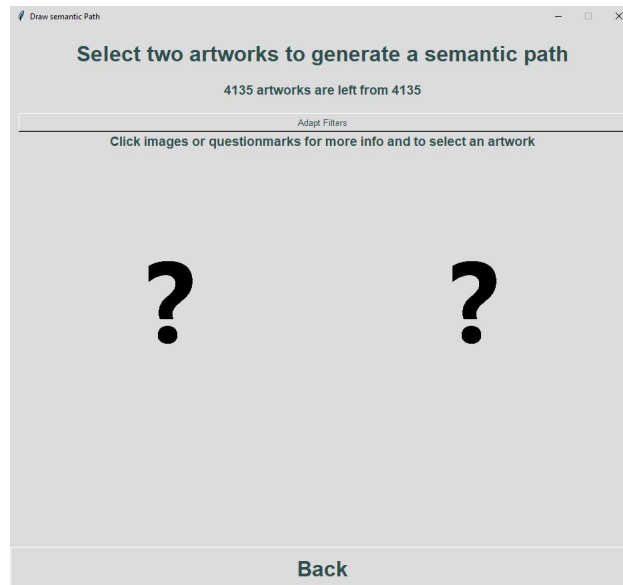
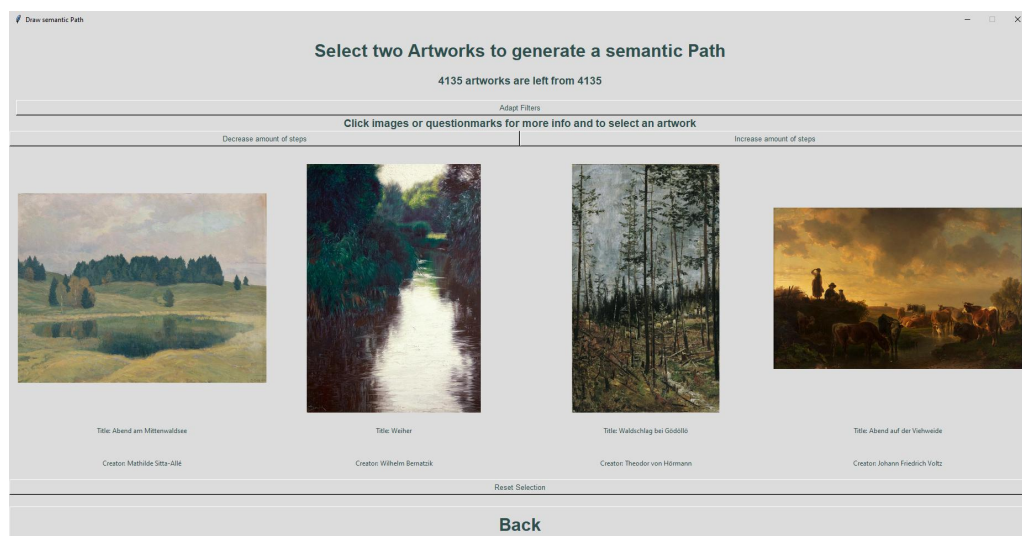


Figure 6.5.: Entry mask for “Create Semantic Path” window

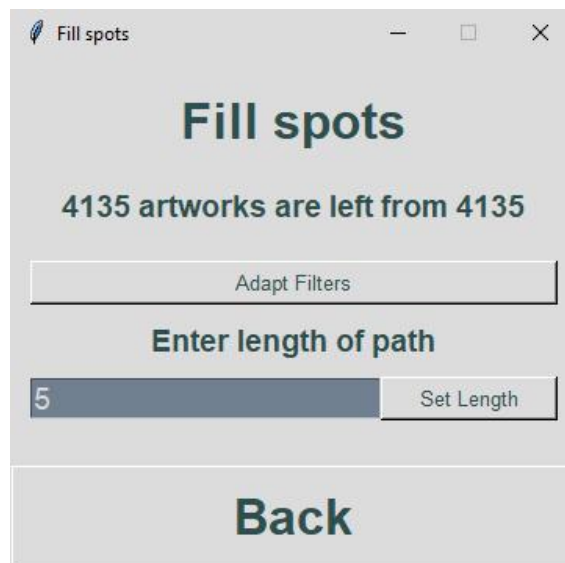


Artworks seen on this image are all from Belvedere, Vienna, Austria (CC BY-SA 4.0
<https://creativecommons.org/licenses/by-sa/4.0/>)

Artworks from left to right: “Abend am Mittenwaldsee” by Mathilde Sitta-Allé, “Weiher” by Wilhelm Bernatzik, “Waldschlag bei Gödöllö” by Theodor von Hörmann, “Abend auf der Viehweide” by Johann Friedrich Voltz

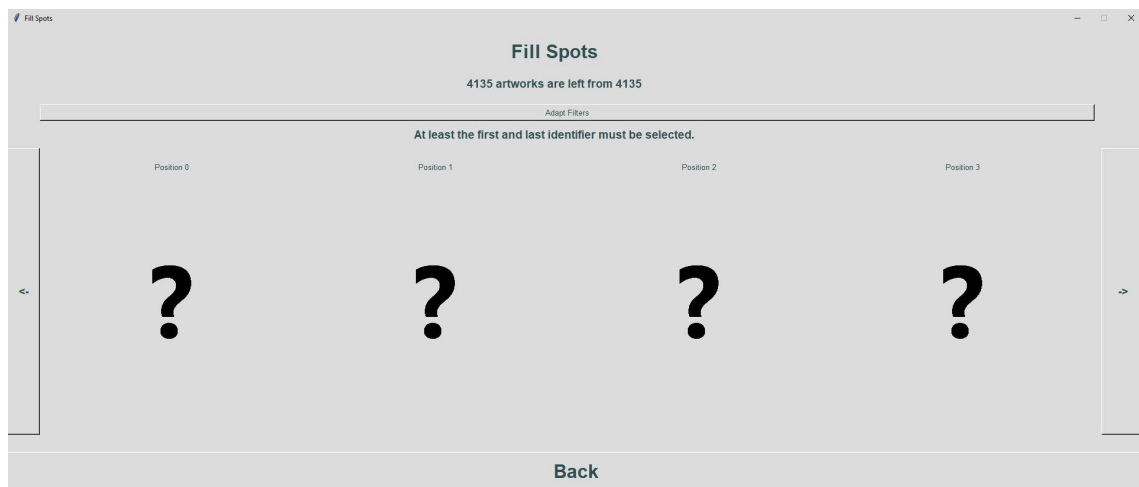
Figure 6.6.: Results of a semantic path in the GUI

6. User Interface



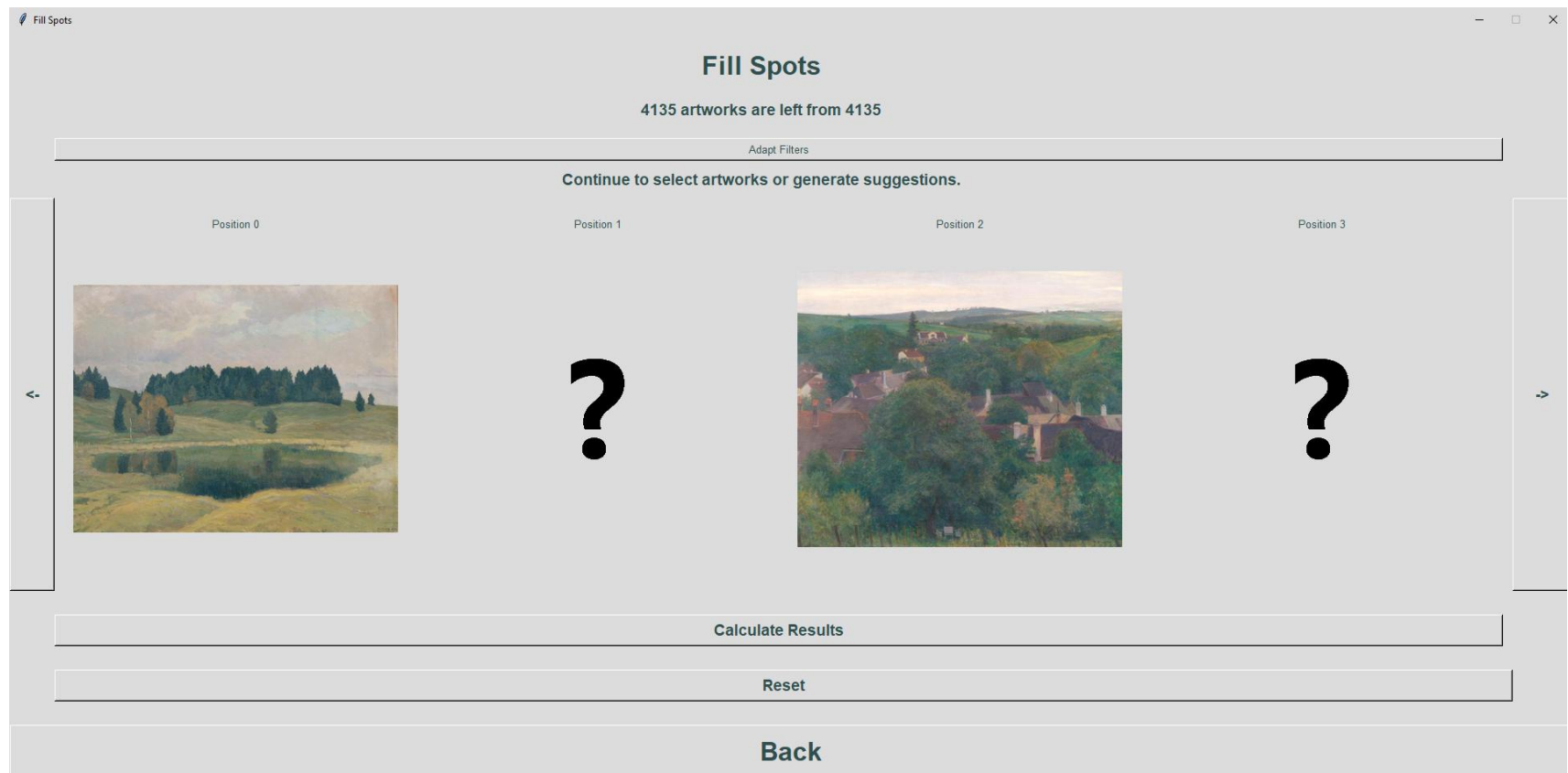
The screenshot shows a window titled "Fill spots" with a standard macOS-style title bar (red, yellow, and green buttons). The main content area has a light gray background. At the top, the text "4135 artworks are left from 4135" is displayed. Below this is a button labeled "Adapt Filters". The next section is titled "Enter length of path" and contains a text input field with the number "5" and a button labeled "Set Length". At the bottom of the window is a large button labeled "Back".

Figure 6.7.: Entry mask 1 of “Fill Spots” window



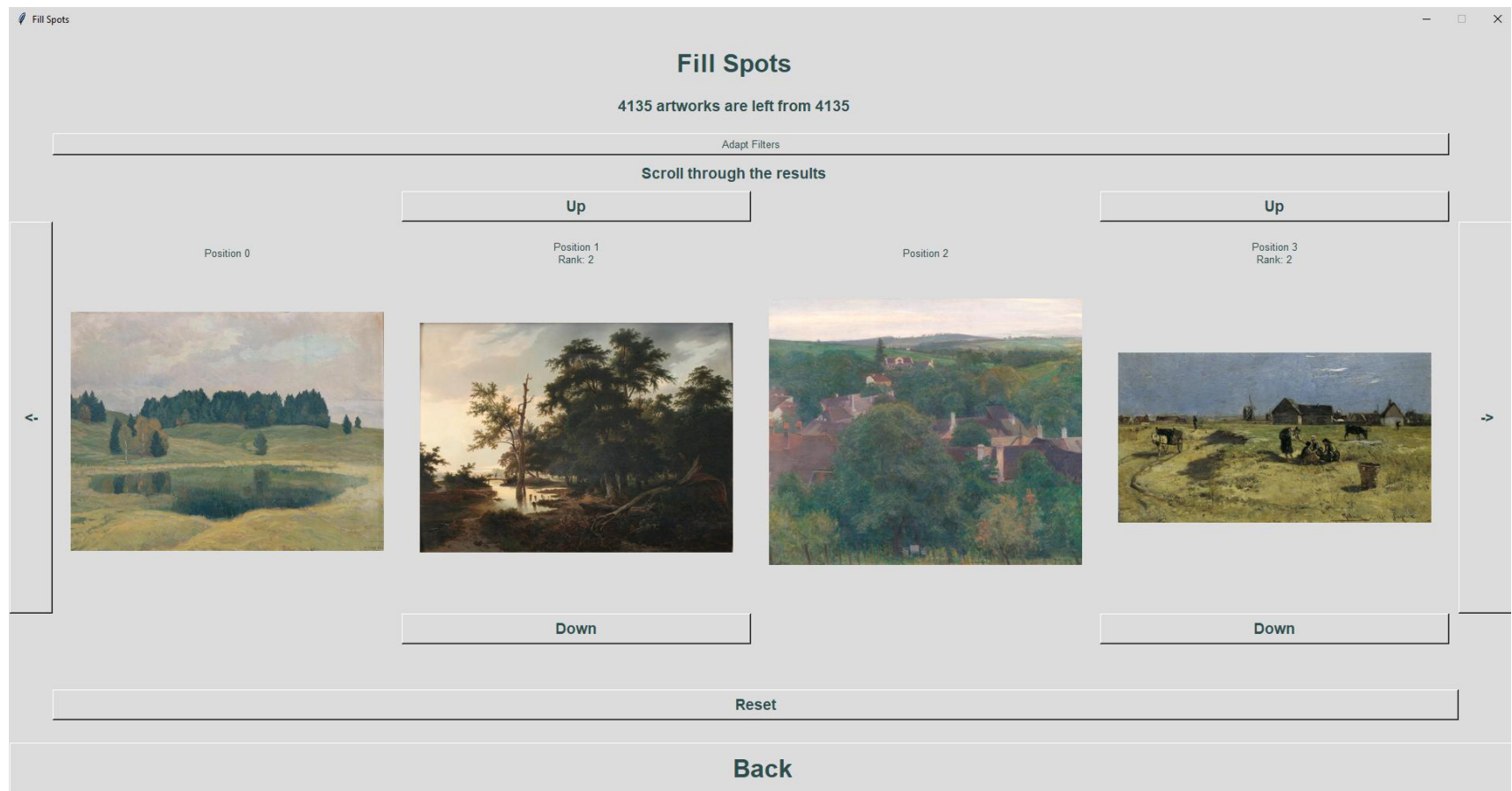
The screenshot shows a window titled "Fill Spots" with a standard macOS-style title bar. The main content area has a light gray background. At the top, the text "4135 artworks are left from 4135" is displayed. Below this is a button labeled "Adapt Filters". The next section is titled "At least the first and last identifier must be selected." and contains four positions labeled "Position 0", "Position 1", "Position 2", and "Position 3". Each position has a large black question mark below it. On the left and right sides of the positions are vertical scroll bars. At the bottom of the window is a button labeled "Back".

Figure 6.8.: Entry mask 2 of “Fill Spots” window



Artworks seen on this image are all from Belvedere, Vienna, Austria (CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0/>)
 Artworks from left to right: "Abend am Mittenwaldsee" by Mathilde Sitta-Allé, "Abenddämmerung in Ober-Sievering" by Rudolf Bacher

Figure 6.9.: Example input of "Fill Spots" in the GUI (only a part of the input is visible)



Artworks seen on this image are all from Belvedere, Vienna, Austria (CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0/>)

Artworks from left to right: "Abend am Mittenwaldsee" by Mathilde Sitta-Allé, "Waldlandschaft" by Anton Altmann d. J., "Abenddämmerung in Ober-Sievering" by Rudolf Bacher, "Landschaft aus Cayeux" by Rudolf Ribarz

Figure 6.10.: Example results of "Fill Spots" in the GUI

7. Outlook

In this thesis semantic as well as visual information was considered for similarity related tasks. The results were quite satisfactory, but work is still to be done:

- In regards to the object recognition...
 - Since false positives possibly have a larger impact than false negatives, experiments can be made with an increased focus on precision.
 - Adding hyper parameters to control the step sizes in curriculum learning (potentially even on a per-hyperparameter basis) could further increase the result.
 - A larger box-annotated dataset would likely boost the performance of the object recognition significantly.
 - The threshold values for filtering bounding boxes could be evaluated for every class on the box-annotated dataset in order to maximize performance.
- In regards to new sources of tags...
 - Image style information could be extracted using CNNs and added to the framework, which already supports such an addition.
- In regards to the GUI...
 - A more modern graphical user interface would increase the usability of the software.
- In regards to scaling the application to larger datasets...

7. Outlook

- The processes regarding similarities could be optimized such that not every artwork is compared to every other artwork. Instead each artwork would be assigned to one or multiple groups. In a first step the algorithm would then calculate in which groups matches are likely and will then only compare to artworks belonging to these groups. Of course this process could be taken further still by bundeling the groups themselves and so on.
- In regards to increasing the capabilities...
 - Data could be also extracted from e.g. Wikipedia to add art historical context to the images. In a similar fashion to what was shown in this thesis, artworks could then be filtered / matched / etc. by their historical background.

8. Conclusion

In this thesis it was shown how information can be extracted from text and image data and how this information can then be used to form connections between artworks. These were then used to perform various tasks such as smoothly transitioning between artworks. The data used in this thesis mostly takes into account what is visually apparent in the artworks. This has the welcome side effect of making the results of the algorithms shown in this thesis somewhat visually verifiable. However, curators might find the processes in this thesis to be of more value when applied to a dataset containing information about e.g. art history.

The matching processes used to create e.g. the semantic paths worked quite well and it is apparent from looking at the results or the implemented measures that the algorithms work as desired. The caching of intermediate results and the efficiency measures turned out to be absolutely vital in order to being able to compute results in reasonable time. The recognition of objects in artworks worked reasonably well with a surprisingly small amount of training data. However, false positives turned out to be problematic. Data augmentations during training of the object recognition model helped to further improve the results. Introducing jitters in brightness and hue worked especially well. It has been observed that the amount of training data for fine-tuning improves the results, however the increase in F1 score stagnates pretty soon. A reason for this could be the quality of the annotations, as the class for which this test was made was particularly difficult to annotate.

The results of this thesis are satisfying and as discussed in chapter 7 there are still many options left that can be taken to further improve the results.

A. Appendix

A.1. Acknowledgements

Special thanks goes to Dr. Arthur Flexer for the continuous and excellent supervision of this thesis and his valuable input, as well as to Univ.-Prof. Dr. Widmer for scientifically supervising this thesis.

The template for the document of this thesis was provided by the JKU on their website. Small changes were made when required.

The author of this work was not involved in creating the dataset that is the very basis of this project and without which this project would not have been possible. Therefore, thanks goes also to the “Dust and Data”-Project for enabling this thesis.

A.2. Disclaimer

The author of this thesis and software is in no case liable for any claim, damage or liability that occurs in connection with dealings of the same. Typos, printing-, software- and typesetting errors in this work reserved. While an effort was made documenting features, some may remain undocumented.

A.3. License of Data

The artworks in the provided dataset were all under the license CC BY-SA 4.0 (<https://creativecommons.org/licenses/by-sa/4.0/>)

A.4. Acknowledgements

The software “draw.io Diagrams” was used in order to generate graphics for this thesis. Therefore, thanks goes to the creators of this software as well.

Bibliography

- [1] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL] (cit. on pp. 3, 4, 5).
- [2] Tomas Mikolov et al. *Distributed Representations of Words and Phrases and their Compositionality*. 2013. arXiv: 1310.4546 [cs.CL] (cit. on pp. 4, 5, 6).
- [3] Piotr Bojanowski et al. “Enriching word vectors with subword information”. In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 135–146 (cit. on pp. 6, 7, 8, 23).
- [4] Edouard Grave et al. “Learning word vectors for 157 languages”. In: *arXiv preprint arXiv:1802.06893* (2018) (cit. on p. 7).
- [5] Arthur Flexer. “Discovering X Degrees of Keyword Separation in a Fine Arts Collection”. In: *Proceedings of the 37th International Conference on Machine Learning, Machine Learning for Media Discovery Workshop, PMLR*. Vol. 108. 2020 (cit. on pp. 8, 51, 52).
- [6] Spacy.io. *Embeddings, Transformers and Transfer Learning*. [Online; accessed 16-November-2021]. 2021. URL: <https://spacy.io/usage/embeddings-transformers> (cit. on p. 9).
- [7] Nikolay Banar, Walter Daelemans, and Mike Kestemont. “Multi-modal Label Retrieval for the Visual Arts: The Case of Iconclass.” In: *ICAART (1)*. 2021, pp. 622–629 (cit. on pp. 9, 12, 13).
- [8] Gustavo Carneiro et al. “Artistic Image Classification: An Analysis on the PRINTART Database”. In: *Computer Vision – ECCV 2012*. Ed. by Andrew Fitzgibbon et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 143–157. ISBN: 978-3-642-33765-9 (cit. on pp. 10, 61).

Bibliography

- [9] Andrea Frome et al. “DeViSE: A Deep Visual-Semantic Embedding Model”. In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. 2013, pp. 2121–2129 (cit. on pp. 10, 11).
- [10] Diana Kim et al. “Computational Analysis of Content in Fine Art Paintings.” In: *ICCC*. 2019, pp. 33–40 (cit. on pp. 11, 12).
- [11] Federico Milani and Piero Fraternali. “A Dataset and a Convolutional Model for Iconography Classification in Paintings”. In: *Journal on Computing and Cultural Heritage (JOCCH)* 14.4 (2021), pp. 1–18 (cit. on pp. 12, 61).
- [12] Arthur Flexer. “Computational Filling of Curatorial Gaps in a Fine Arts Exhibition”. In: *Proceedings of the Twelfth International Conference on Computational Creativity, México City, México (Virtual), September 14-18, 2021*. Association for Computational Creativity (ACC), 2021, pp. 2–5 (cit. on pp. 13, 51, 52).
- [13] Spacy.io. *Linguistic Features*. [Online; accessed 09-January-2022]. 2022. URL: <https://spacy.io/usage/linguistic-features/> (cit. on p. 21).
- [14] Arthur Flexer et al. “Playlist Generation using Start and End Songs”. In: *ISMIR 2008, 9th International Conference on Music Information Retrieval, Drexel University, Philadelphia, PA, USA, September 14-18, 2008*. 2008, pp. 173–178 (cit. on pp. 51, 52).
- [15] Thomas Lintner. “Exploration of the Effects on Data Preprocessing on Convolutional Neural Network Classification Performance”. In: (2020). Unpublished technical report (cit. on p. 58).
- [16] CM Sukanya, Roopa Gokul, and Vince Paul. “A survey on object recognition methods”. In: *International Journal of Science, Engineering and Computer Technology* 6.1 (2016), p. 48 (cit. on p. 59).
- [17] Yoshua Bengio et al. “Curriculum learning”. In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 41–48 (cit. on p. 62).
- [18] Wikipedia contributors. *F-score — Wikipedia, The Free Encyclopedia*. <https://en.wikipedia.org/w/index.php?title=F-score&oldid=1077757029>. [Online; accessed 6-May-2022]. 2022 (cit. on p. 62).