Copenhagen Business School

Department of Digitalization

Center for Business Data Analytics

Master Thesis for the Cand.merc.(it.) (Data Science)

Revolutionizing the Danish Broadcasting Industry: A Comprehensive Study of the Influence of AI on Video Metadata Management

Supervisor

Prof. José Parra-Moyano

Students

Konrad Schulte
kosc21ab@student.cbs.dk
149872

Marin Mes
mame21ak@student.cbs.dk
149899

Contract Number: 29913
Page Count: 69

May 15, 2023

Abstract

As digital content experiences extraordinary growth, multimedia organizations increasingly require effective and scalable metadata generation strategies. This research project investigates the potential of Artificial Intelligence (AI) in this field by addressing the following research question: "To what extent can AI play a role in the process of generating metadata for digital video archives in broadcasting corporations?"

A quantitative approach was employed, involving the implementation and evaluation of pre-trained AI models performing popular tasks such as multilabel image classification, image segmentation, object detection and image captioning. In total, seven individual models were tested on data provided by TelevisionCo, a Danish broadcasting company. Besides the individual models, a hard voting ensemble approach was pursued to investigate whether an ensemble of AI models can outperform individual AI models. Finally, this thesis investigated the necessity of analyzing each frame in a video to produce coherent metadata.

The results revealed that ensembles outperform individual models. The best performing ensemble yielded a similarity score of 36.85% and carried a business value of 96,439,139 DKK for TelevisionCo, outweighing their current metadata costs. Moreover, the results indicated that not all frames needed to be analysed for coherent metadata, and suggested an optimal duplicate removal threshold where $MeanSquaredError = 640$.

The findings of this project have significant implications for both multimedia corporations and the academic community. The results demonstrate the potential of AI to revolutionize metadata generation processes, enhance efficiency, and contribute to cost reduction and resource allocation. Furthermore, these findings open new avenues for academic research in the application of AI within the multimedia domain.

Keywords: automatic metadata generation, digital video archives, broadcasting corporations, artificial intelligence, ensemble learning

Contents

List of Acronyms **IV**

List of Figures **V**

List of Tables **VII**

1 Introduction **1**
1.1 Business Case . . . 1
1.2 Ensemble Learning . . . 3
1.3 Academic Contribution & Business Value . . . 3
1.4 Research Design . . . 4
1.5 Thesis Outline . . . 5

2 Literature Review **7**
2.1 The Value of Metadata . . . 7
2.2 Automatic Generation of Metadata for Digital Video Archives . . . 8
2.3 Computer Vision . . . 10
2.3.1 Multilabel Image Classification . . . 10
2.3.2 Image Segmentation . . . 11
2.3.3 Object Detection . . . 12
2.3.4 Image Captioning . . . 13
2.3.5 Optical Character Recognition . . . 14
2.4 Recent Developments . . . 15

3 Theoretical Framework **17**
3.1 Multilabel Classification . . . 17
3.2 Image Segmentation . . . 18
3.3 Object Detection . . . 19
3.4 Image Captioning . . . 20
3.5 Optical Character Recognition . . . 21
3.6 Named Entity Recognition . . . 22
3.7 Ensemble Learning . . . 22
3.8 Leveraging Pretrained Models . . . 23
3.9 Cosine Similarity . . . 24

4 Methodology **25**
4.1 Business Communication . . . 25
4.2 Project Approach . . . 26
4.3 Data Overview & Resources . . . 26
4.3.1 Videos . . . 27

4.3.2 Metadata . . . 27
4.3.3 Company-Specific Lists . . . 27
4.3.4 Computational Environment & Software Dependencies . . . 28
4.4 Algorithmic Workflow Design . . . 28
4.4.1 Preprocessing . . . 29
4.4.2 Tasks . . . 31
4.4.3 Summarization . . . 37
4.5 Ensemble Method . . . 41
4.5.1 Model Ensembles . . . 41
4.5.2 Runtime Ensembles . . . 42
4.6 Evaluation . . . 42

5 Results 44
5.1 Models and Ensembles . . . 44
5.2 Duplicate Removal . . . 48
5.3 Ensemble Size . . . 51
5.4 Examples and Scene Specific Analysis . . . 52

6 Discussion 55
6.1 Sub Questions Answering . . . 55
6.1.1 Sub Question 1 . . . 55
6.1.2 Sub Question 2 . . . 56
6.1.3 Sub Question 3 . . . 56
6.1.4 Sub Question 4 . . . 57
6.1.5 Sub Question 5 . . . 58
6.2 Research Question Answering . . . 60
6.3 Academic Implications . . . 61
6.3.1 Optimal Duplicate Removal Threshold . . . 61
6.3.2 Optimal Ensemble Size & Approach . . . 61
6.3.3 Custom Models and Task-Specific Training . . . 61
6.4 Business Implications . . . 62
6.4.1 Adoption of Artificial Intelligence . . . 62
6.4.2 Proven Potential of Open-Source Technologies . . . 62
6.4.3 Cost Savings and Resource Reallocation . . . 62
6.5 Recommendations for TelevisionCo . . . 62

7 Limitations & Further Research 64
7.1 Limitations . . . 64
7.1.1 Model and Data Limitations . . . 64
7.1.2 Evaluation Metric Limitations . . . 65
7.1.3 Ensemble Method Limitations . . . 66
7.2 Further Research . . . 66
7.2.1 Dataset and Model Improvements . . . 66
7.2.2 Ensemble Method Refinements . . . 66
7.2.3 Efficiency and Performance Enhancements . . . 67
7.2.4 Extensive Evaluation and Generalization . . . 67

8 Conclusion 68

Bibliography **70**

Appendix **77**

A Overview of Communication with TelevisionCo. **77**

B Supplementary Files from TelevisionCo. **82**
B.1 List of Abbreviations & Fixed Terms 82
B.1.1 Abbreviations 82
B.1.2 Fixed Terms 83
B.2 List of Valuable Object Types 84

C Word Count Delimitation **87**

D Correlation **91**

E Upset Plots **92**

F Code **94**
F.1 Pipeline 94
F.1.1 Requirements 94
F.1.2 Preprocessing 96
F.1.3 Methods 103
F.1.4 Summarization 116
F.2 Master File 125
F.3 Word Count Delimitation Tests 127
F.4 Ensemble Method 132
F.4.1 Predictions 132
F.4.2 Runtimes 134
F.5 Results Merge & Evaluation 135
F.5.1 Runtime Concatenation 135
F.5.2 Prediction Evaluation 136
F.5.3 Runtime & Score Merge 138
F.6 Correlation Test 138

List of Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
BERT	Bidirectional Encoder Representation from Transformers
BiT	Big Transfer
CNN	Convolutional Neural Network
HD	High Definition
IC	Image Captioning
IS	Image Segmentation
LOC	Location
MC	Multilabel Classification
MD	Metadata
MSE	Mean Squared Error
NLP	Natural Language Processing
NER	Named Entity Recognition
NR	Night Recognition
OCR	Optical Character Recognition
OD	Object Detection
ORG	Organisation
PER	Person
R-CNN	Region-based Convolutional Neural Network
SR	Scene Recognition
SSD	Single Shot Detector
TR	Text Recognition
XML	Extensible Markup Language
YOLO	You Only Look Once

List of Figures

3.1 Visualization of Different Classification Tasks. 18
3.2 Semantic Segmentation Compared to Instance Segmentation (Aiforia, 2021). 19
3.3 Example Output of an SSD Object Detection Model (TensorFlowHub, 2023). 20
3.4 Architecture of Image Captioning. 21
3.5 An Example of a Named Entity Recognition Analysis. 22
3.6 Visualization of the Ensemble Learning Approach Employed in this Thesis. 23

4.1 Project Architecture 26
4.2 Illustration of Algorithmic Workflow. 28
4.3 Occurrence of Company-Specific Abbreviations in 952 Scenes. 35
4.4 Occurrence Distribution of "Ext. opt." and "Int. opt." with Delimited Thresholds at x=0.28 and x=0.7, Respectively. 39
4.5 Occurrence Distribution of "Natopt." and without ("Day") with Delimited Threshold at x=54.15. 40

5.1 Performance and Runtime Overview for each Individual Model. 44
5.2 Scatterplots with Specific Models Highlighted in Red. 45
5.3 Average Performance per Ensemble at MSE=0 47
5.4 Average Similarity Score and Average Runtime per Frame for Top 10 Performing Ensembles. 48
5.5 Information Loss per MSE Value. 49
5.6 Relative Performance and Relative Runtime 49
5.7 Absolute Performance versus Absolute Runtime per Frame for each Individual Model. 50
5.8 Performance and Runtime by Ensemble Size. 51
5.9 Screenshots from Top 10 Performing Scenes in Dataset. 53

7.1 Screenshot of Scene where AI Ensemble Potentially Outperforms Ground Truth. 65

A.1 Overview of Brainstorm Session with TelevisionCo. 79
A.2 Evaluation of Pro's and Con's per Topic from Brainstorm Session. 79
A.3 Initial Architecture. 80
A.4 Proposed Project Plan. 81

C.1 Line Plot Highlighting Optimal Number of Words for (Individual) IC Model. 87
C.2 Line Plot Highlighting Optimal Number of Words for (Individual) IS Model. 88

C.3 Line Plot Highlighting Optimal Number of Words for (Individual) OD Model. 88
C.4 Line Plot Highlighting Optimal Number of Words for (Individual) MC Model. 89
C.5 Line Plot Highlighting Optimal Number of Words for (Individual) TR Model (Maximal 10 Words). 89
C.6 Line Plot Highlighting Optimal Number of Words for (Individual) TR Model (Maximal 20 Words). 90

D.1 Correlation Matrix for Correlation Between Scene Lengths and Performances. 91

E.1 Upset Plot Describing Performance of each Ensemble Including Image Captioning (IC), Sorted by Ensemble Size. 92
E.2 Upset Plot Describing Performance of each Ensemble Including Image Segmentation (IS), Sorted by Ensemble Size. 92
E.3 Upset Plot Describing Performance of each Ensemble Including Object Detection (OD), Sorted by Ensemble Size. 92
E.4 Upset Plot Describing Performance of each Ensemble Including Multilabel Classification (MC), Sorted by Ensemble Size. 93
E.5 Upset Plot Describing Performance of each Ensemble Including Scene Recognition (SR), Sorted by Ensemble Size. 93
E.6 Upset Plot Describing Performance of each Ensemble Including Night Recognition (NR), Sorted by Ensemble Size. 93
E.7 Upset Plot Describing Performance of each Ensemble Including Text Recognition (TR), Sorted by Ensemble Size. 93

List of Tables

2.1 Table 2.1: Overview of References and Topics. 16

4.1 Table 4.1: Data Overview. 26
4.2 Table 4.2: Technical Overview of Pipeline Functions. 30
4.3 Table 4.3: Duplicate Removal Effect on Frame Count. 32
4.4 Table 4.4: Overview of Applied Tasks & Models. 32

5.1 Table 5.1: Top 10 Scenes with Highest Similarity Scores 52

7.1 Table 7.1: Overview of Five Most Important Limitations & Further Research Implications. 67

A.1 Table A1: Overview of Meetings Between Authors and TelevisionCo. . . 78

Chapter 1

Introduction

Over the past decade, there has been a remarkable surge in the field of Artificial Intelligence (AI), resulting in numerous breakthroughs and innovative research. The release of OpenAI's ChatGPT and DALL-E applications has broadened AI's accessibility and impact from a select audience to the majority of society, and the forthcoming launch of Microsoft Copilot is anticipated to even further integrate AI into society's everyday life. These advancements, along with future developments in AI, promise to unlock a new era of possibilities and opportunities for both businesses and individuals in this rapidly evolving domain.

The popular AI applications ChatGPT and DALL-E are predominantly used by society for personal entertainment purposes and for aid with minor tasks. However, the true value of these applications, and all other AI models, resides in their potential to drive transformation within companies and businesses. Whenever companies begin to adopt AI as a crucial component of their internal or external business processes, that is when AI's true power can become evident. However, what exactly this true power is and how exactly it creates business value remains a grey area and may differ across industries. Therefore, the purpose of this thesis is to investigate the role AI can currently play in the corporate landscape and what value it can create for a business.

In seeking to enhance the understanding of AI's potential applications and advantages for businesses, this thesis will focus on the broadcasting industry. The broadcasting industry, encompassing both television and radio, plays a vital role in disseminating information and entertainment to the masses. With the shift from physical to digital archives, broadcasters have accumulated extensive digital repositories, containing video, image, and audio content that spans various genres, time periods, and formats. As the volume of digital content continues to grow considerably, these archives are expanding at an unprecedented rate. This ongoing expansion presents both challenges and opportunities, particularly in the context of AI applications. In the following section, the business case of this thesis will be introduced.

1.1 Business Case

The business case of this thesis revolves around a Danish company which is responsible for broadcasting radio and television programs in Denmark, hereinafter referred to as TelevisionCo. Established as Denmark's oldest and largest electronic media enterprise, TelevisionCo has an enduring presence in the country's media landscape.

As a producer of television programs, TelevisionCo has a vast digital video archive containing years worth of video footage such as interviews, nature recordings, street recordings, and more. The majority of this footage can be reused and recycled for upcoming news items, documentaries and talk shows. To facilitate the reuse of this content, it is essential that each recording is accompanied by descriptive metadata explaining what is happening in a video at different timestamps. This allows TelevisionCo to navigate through their archives, and is therefore of utmost importance.

The subsequent statistics about TelevisionCo seek to provide an understanding of the scope of the business problem being addressed:

- Current size of the video archive: **460,229 hours**
- Yearly influx of footage: **38,000 hours**
- Yearly amount of footage for which metadata is created: **1,718 hours**
- Yearly amount of hours spent on creating metadata: **910 hours**, equivalent to **0.6 FTE**
- Yearly costs for generating metadata in terms of salary: **360,000 DKK**

TelevisionCo's current workflow for creating descriptive metadata suffers from two critical flaws that impede the process and, consequently, hinder effective navigation through their video archives:

1. The creation of descriptive metadata is a manual process. Employees must watch all incoming footage, whether it be an interview or a two-hour video of the same street, and manually document the events occurring in each scene. This approach is time-consuming and non-scalable, which is reflected in the fact that TelevisionCo cannot keep up with the yearly influx of video footage.
2. The existing descriptive metadata lacks consistent quality across the archive. Over the years, TelevisionCo has utilized various frameworks and templates for descriptive metadata, encompassing numerical, sentence-based, and keyword-driven approaches, the one more successful than the other. This inconsistency greatly complicates the process of locating and retrieving specific video content.

Upon examining these shortcomings in greater detail, it can be stated that their common factor is "time". Firstly, TelevisionCo invests considerable time in the creation of metadata, and then secondly, TelevisionCo spends a great amount of time on searching for appropriate footage within their archives.

The first major time expenditure is felt hardly by the employees at TelevisionCo. In an initial meeting, all of the participating employees acknowledged that the manual creation of metadata is a lengthy process they wish to eliminate, as it would enable them to allocate their time to other important tasks (Appendix A). Note that this is not a new or unknown problem. For decades, research has consistently identified that the most significant opportunities in the field of descriptive metadata lie in devising a method for its automatic generation (Wactlar & Christel, 2002).

In addition to the time-consuming nature of manual metadata generation, this process is also inherently non-scalable. As previously mentioned, digital media content continues to grow considerably, and TelevisionCo's archives will undoubtedly expand significantly in the coming years. While hiring additional staff to generate metadata may initially appear viable, this approach becomes increasingly untenable as the archive's size reaches unprecedented levels.

The second substantial time expenditure stems from the inconsistent quality of the existing descriptive metadata in the archive. High-quality and consistent metadata is crucial for TelevisionCo, as it facilitates efficient navigation through their video archives and allows employees to make a well-informed decision on whether a video can be re-used for a certain purpose or not. This implies that the higher the quality of the generated metadata, the more satisfied the employees of TelevisionCo will be.

Bringing the above together, this business problem presents an excellent opportunity to investigate how AI models can enhance workflows and increase efficiency within a company such as TelevisionCo.

1.2 Ensemble Learning

The scope of this thesis is not limited to testing and evaluating the performance of individual AI models in generating metadata for TelevisionCo. While there have been notable advancements in the field of computer vision, current AI models have their limitations when it comes to their capabilities in tackling more complex problems such as generating metadata (Grover & Ermon, 2018). To address this challenge, this thesis proposes to research whether an ensemble approach, that combines the output of a multitude of AI models, can create coherent descriptive metadata and outperform individual AI models.

In Ensemble Learning, the key philosophy is that a "set of weak learners can create a single strong learner, provided that there are a sufficient number of weak learners and they are sufficiently diverse" (Géron, 2019, p. 251). This implies, that an ensemble approach may allow for synergies between different AI models, and therefore higher performance.

1.3 Academic Contribution & Business Value

The primary objective of this master thesis is to contribute to the academic field of automatic metadata generation for digital video archives. The focus will be on exploring the capabilities of AI models to generate descriptive metadata and determining whether an ensemble of AI models can outperform individual AI models. In addition to the academic contribution, this research aims to provide a practical solution for TelevisionCo. Specifically, it seeks to offer the company a more efficient process for creating descriptive metadata, as well as a consistent quality of metadata, allowing for smoother navigation through their digital video archives. The central hypothesis of this research is that the automation of the metadata generation process can lead to optimized resource utilization and an enhanced overall user experience for individuals accessing the digital archive.

In order to guide the research of this thesis, the following research question has been formulated:

To what extent can AI play a role in the process of generating metadata for digital video archives in broadcasting corporations?

To find an answer to this research question, the following five sub questions should be answered:

1. Which AI models can be used to automatically generate descriptive metadata for videos?
2. Can an ensemble of AI models produce synergies and outperform individual AI models?
3. To what extent does a trade-off exist between performance and runtime for the individual models and ensembles utilized?
4. Is it necessary to analyze every frame within a scene, or can the examination of a reduced set of extracted frames also provide sufficient information?
5. What potential business value might a pipeline of AI models, designed to create metadata for digital video archives, have for a broadcasting company?

1.4 Research Design

This section outlines the research design for finding an answer to the research questions. First of all, the AI models employed in this thesis to generate the descriptive metadata, will solely be open-source, pretrained AI models. Using pretrained models significantly reduces the computational cost and time required to obtain results. Since videos are in general computationally heavy to process, and due to the fact that the computing power for this thesis was limited, the decision was made to rely only on open-source, pretrained models.

Secondly, this thesis focuses on generating descriptive metadata solely based on the visual features of a video. This means that only AI models using techniques from the field of computer vision will be employed. Other aspects of videos such as sound and speech are disregarded as most videos did not include these features. Besides the visual aspects, this thesis will also exploit the technical metadata a camera produces when shooting a video. This technical metadata is provided to the researchers by TelevisionCo and contains all supplementary information needed to produce a comprehensive output.

Finally, during the evaluation process of the individual models and ensembles, both the performance and the computational runtime complexity will be taken into consideration. The performance will be evaluated as follows: for each video in the dataset, TelevisionCo has provided descriptive metadata that is of outstanding quality. This metadata will be referred to as the "Ground Truth" in this thesis. A comparison between the generated output and the Ground Truth will be made for each model and ensemble, and will be used as the performance indicator.

In order to derive a monetary business value from the technical results, the following formula has been constructed:

$$V_m = \min\{Q_m \times C_{TV} \times \frac{T_{TV}}{T_m}, C_{TV}\}$$

Where:

- V_m corresponds to the value of the proposed solution for TelevisionCo.
- Q_m corresponds to the quality of the proposed solution (determined by average similarity score).
- C_{TV} corresponds to the current costs for TelevisionCo to produce metadata.
- T_{TV} corresponds to the current time it takes TelevisionCo to produce metadata.
- T_m corresponds to the time it takes for the proposed solution to generate metadata.
- min{} indicates that the smaller of the two arguments should be selected.

In Chapter 6, a more extensive explanation of the formula will be provided, as well as the final outcome.

1.5 Thesis Outline

This thesis is structured as follows. Chapter 2 will present an overview of the relevant literature, analyzing the most relevant works on descriptive metadata, digital video archives, the automatic generation of metadata, and the different AI models and machine learning techniques that have been developed to process images and videos. In doing so, this chapter aims to provide a comprehensive overview of the current state of the art in the field of automated metadata generation for digital video archives.

Following this, Chapter 3 will present a theoretical framework in which the different concepts and methods introuced in the literature review will be explained. The chapter will provide a brief but concise explanation of the technical specifications of the machine learning techniques used by the employed AI models, the reasoning behind the employment of these models, and any potential weaknesses.

Chapter 4 will introduce the methodology of this thesis, and will delve into the various steps undertaken during the project. This includes, but is not limited to, the preprocessing, modelling, and postprocessing steps, as well as the challenges encountered during the process.

Then, Chapter 5 will present the results and the findings of this thesis. Among other things, the performance of the models and the ensembles with regards to both similarity score and time will be described in detail. This chapter is closely followed by Chapter 6, which will present a discussion of the findings, as well as academic implications, business implications and recommendations for TelevisionCo. This chapter will analyse and discuss the results by means of the sub-questions mentioned in Section 1.3, and thereafter will formulate an answer to the research question.

Subsequently, Chapter 7 will be dedicated to identifying the limitations of this thesis, as well as outlining areas of potential further research. Finally, this thesis concludes with Chapter 8 which will provide an extensive summary of the key findings of this thesis and their implications.

Chapter 2

Literature Review

This chapter is structured as follows. It starts by examining established papers on the value of metadata and its role in digital video archives. It then delves into past projects about the automatic generation of metadata and offers a brief overview of the most relevant use cases. Subsequently, the chapter presents a comprehensive overview of the most pertinent and established machine learning techniques that can be applied to perform the task of this thesis. Finally, this chapter highlights the most cutting-edge, recent research that emerged in the field of computer vision. The aim of this chapter is to offer a concise overview of the current literature in this field.

2.1 The Value of Metadata

The exponential growth of digital video content in the late 20th century resulted in a corresponding increase in size and complexity of digital video archives. Consequently, due to its sheer volume, managing these archives became a significant challenge (Wactlar & Christel, 2002). In 2000, Laven and Hopper suggest that due to this change, the effective management and navigation of video archives will become increasingly reliant on accurate descriptive metadata. Wactlar and Christel (2002) supported this claim by stating that the significance of metadata is evident as it enables digital archives with enormous amounts of video content to become a valuable information resource. In the absence of metadata, such archives would merely be an collection of information with no coherent meaning. Hence, they state that metadata serves as the gateway to accessing and utilizing these archives.

In their study, Wactlar and Christel (2002) explore the use for, and production of, *descriptive metadata*, a term coined by Wendler (1999) to describe metadata which provides information about the content of a resource. Similar to Wactlar et al. (1999) and Witbrock and Hauptmann (1997), they find that descriptive metadata does not have to be faultless for it to be useful. Metadata that contains wrong information can still be resourceful for information retrieval and navigation purposes in a digital archive. They conclude by stating that the biggest opportunities for descriptive metadata lie in finding automated ways to create it, since manual creation is costly and unsuitable for managing large digital video archives. They predict that the automatic analysis of video content will become an inevitable step in managing digital video archives.

2.2 Automatic Generation of Metadata for Digital Video Archives

As Wactlar and Christel (2002) predicted, automatic generation of descriptive metadata has been the focus of much research in the past two decades. Most of these studies concerned creating synthetic descriptions for instructional videos using textual and audio data. For example, Liddy et al. (2002), Dorai et al. (2006), Maratea et al. (2013) and Rafferty et al. (2015), were all successful in automatically creating descriptive metadata for instructional videos by analyzing the audio spoken by the instructor and text that can be seen on the slides in the video.

Despite most prior research on automatic metadata generation being focused on audio and text analysis, there have been some projects exploring the possibility of using computer vision as well. The first known attempt was the Informedia Project by Wactlar et al. (1999) at Carnegie Mellon University, which employed computer vision techniques, as well as speech recognition and natural language processing methods, to automatically generate descriptive metadata for digital video libraries. This research was a major breakthrough in the field since they demonstrated a way to extract previously inaccessible information from visual aspects of videos (on top of audio and speech) for describing videos (Wactlar et al., 1999).

However, since computer vision was not very developed yet in 1999, the visual processing was rather primitive and based on human perceptual color clustering. The visual process worked as follows: users of the digital archive would see an example image after which they point out which colors or parts of the picture they are interested in. Following this, the system would construct feature vectors of colors and areas of interest, and use these to search the video database using a nearest-neighbours algorithm (Wactlar et al., 1999).

In addition to this application, the authors dedicated a considerable part of the project to utilizing image processing for identifying distinct scenes within a video. While modern video cameras automatically handle this task during recording, back then, they had to utilize computer vision approaches.

Since this was the most they achieved with image processing, computer vision only played a minor role in the automatic generation of metadata, and the focus lay more on the audio and textual data. The Informedia Project concluded by encouraging future research to concentrate on developing models capable of detecting objects within images, thereby enabling content-based searches in digital archives rather than color-based searches.

Ten years later, Snoek et al. (2009) conducted the MediaMill project in which they demonstrated that it is possible to automatically label videos based on their semantic content. The MediaMill project extends the Informedia Project by detecting low-level visual features such as color, texture, and motion using machine learning techniques, and using Support Vector Machines to detect semantic concepts such as "person walking" or "car driving".

These support vector machines were trained using a large, annotated dataset, and then they were used to automatically label the videos with the concepts they contain. Then the detected video content was translated to a set of words, using a bag-of-visual words approach where the obtained low-level visual features were clustered, and the resulting clusters were used as words. These visual words were then used to generate captions for the video. For example, if the support vector

machine detected "person walking", the system would generate a caption such as "a person walking in the video".

In 2017, Fernández et al. introduced ViTS, a video tagging system for massive web multimedia collections that can generate multiple basic words to be matched to a video. ViTS generates tags based on the video itself, its web context and comments shared on social media. Additionally, ViTS maintains a knowledge base that updates without any human interference in real time (Fernández et al., 2017, pp. 337). The goal of ViTS is to create tags that accurately describe the entire video, but also to determine the relevant scenes in a video in order to create an accurate summary.

ViTS works in two blocks: the first block is the Video Action summarization algorithm that analyzes the full video using computer vision techniques. The video action summarization algorithm works as follows (Fernández et al., 2017, pp. 240): a video is split up into segments or scenes based on changes in color and motion by using optical flow. Then, objects are detected and tracked based on K-means clustering across space and time, something Snoek et al. also demonstrated in the MediaMill project (2009). Next, each object is assigned a score that takes into consideration the object's size, focus and position over each frame, as well as the frequency of occurrence over the whole video. In addition, the optical flow is further analyzed to compute an action score based on recognizing a set of predefined activities such as running or playing sports. Finally, a Video Segment Score is computed by summing the object and action scores, so that the most relevant segments of the videos are kept as video summaries.

The second block (pp. 340-341) is the Contextual Tagging Algorithm which crawls the internet to find keywords associated to the indexed videos and then maps back the found entities (Fernández et al., 2017). The contextual tagging algorithm looks at the title, description, full text, existing metadata, Twitter comments, hashtags, YouTube comments and YouTube metadata. Then a key word extractor runs over all of the scraped information and maps back key words as results (Fernández et al., 2017).

A final relevant use case in this section concerns a research paper by Shakurov et al. (2021). This paper addresses the same business problem as the one explored in this thesis, but with a focus on facial recognition. Specifically, the goal of Shakurov et al. was to create software which would reduce the workload of video archivists at the Russian TV company Volgograd-TRV by providing them with automatically generated metadata containing the names of all detected persons in a certain video.

To detect faces, the researchers used the directional gradient histogram method combined with a convolutional neural network. They designed their solution as a software, enabling the archivists to work in parallel with the program, thereby increasing work efficiency, or allowing them to configure the system for sequential task processing over extended periods, such as weekends (Shakurov et al., 2021, pp. 2).

Although the researchers do not touch upon the accuracy of the model, they acknowledge that their solution does not fully replace the work done currently by the archivists. They caution that, when evaluating the metadata provided by the software, users must take into account that the program may determine a person with insufficient accuracy. Therefore, in cases of low accuracy, it is recommended to manually check the specified time interval in the video file (Shakurov et al., 2021). The paper by Shakurov et al. (2021) concludes by stating that the main direction

for further development of their software is adding the ability to recognize objects such as monuments and famous places.

Upon examining the aforementioned use cases, it is evident that they share similarities with the subject matter of this thesis. However, it is important to note that none of them directly align with its central focus. The Informedia Project (1999) primarily relies on audio and textual data to generate metadata, which is not present in the available data for this thesis. Additionally, the Informedia Project's visual analyses were primarily aimed to determine scenes within a video, while the videos used in this thesis have readily predefined scenes. The MediaMill project (2009) comes closer to the aim of this thesis, but only generates basic descriptions and is unable to label a video with more than one description. Meanwhile, the ViTS application (2017) can successfully create multiple tags per video, but relies mainly on contextual text data scraped from the web. Moreover, like the Informedia Project, the visual analyses conducted in the ViTS are primarily focused on constructing a summary of a video by detecting the most important scenes. Lastly, the research by Shakurov et al. (2021) aims to automate the work of archivists at a Russian TV company, but only focuses on detecting faces in videos, covering only a subpart of the current project.

2.3 Computer Vision

Despite the use cases from the previous section differing in their ultimate goal, they all approached video analysis in the same way: by analyzing the different frames within the videos using computer vision. Therefore, in this section the focus will shift from videos to images.

In the field of computer vision for images, previous research has identified five common machine learning methods for analyzing and detecting objects, actions, settings, and text. In the following subsections, each of these methods will be outlined, together with an overview of the most important research achievements in each method.

2.3.1 Multilabel Image Classification

The first computer vision technique to be discussed is Multilabel Image Classification. Image classification is a widely researched computer vision approach and concerns the categorization of images into one or more categories. Specifically, *multilabel image classification* is a method that involves assigning a set of one or more categories to images (Tidake & Sane, 2018). Over the past two decades, much research has been conducted in the field of multilabel classification in computer vision and some of the most important developments will be discussed in the following paragraphs.

In the early years of multilabel classification, Tsoumakas and Katakis (2007) proposed two distinct approaches. The first approach, coined as *transformation*, involves transforming the data that requires classification into the format that fits a single-label classification task, so that existing and established algorithms can be used to solve the task. In other words, "fit the data to the algorithms" (Zhang and Zhou, 2014, p. 4). The second approach, referred to as *adaptation*, involves

modifying simple classification algorithms so that they can solve multilabel classification tasks. Put differently, "fit the algorithm to the data" (Zhang and Zhou, 2014, p. 4). A multitude of studies researched multilabel classification with the latter approach leading to the successful modification of single-label classifiers, such as Decision Trees, Support Vector Machines, Bayes Classifiers, K-Nearest Neighbours and Neural Networks (Tidake & Sane, 2018).

The first machine learning model to be successfully adapted to the multilabel image classification problem, was the support vector machine (X. Li et al., 2004; Vaidya et al., 2008). Following this, the next breakthrough in the field of classification was in 2012, when Krizhevsky et al. introduced image classification using Deep Convolutional Neural Networks. Subsequently, researchers focused on developing deep neural network methods that could address the multilabel classification problem. Wei et al. (2014), for instance, introduced a method that converted the multilabel classification task into multiple binary classification tasks and employed a deep neural network architecture to revert the outcomes back into a multilabel classification format. Similarly, J. Wang et al. (2016) proposed a combined convolutional neural network and recurrent neural network for multilabel classification.

One of the most recent breakthroughs in the AI domain was the discovery of transformer models by Vaswani et al. (2017). This new paradigm of machine learning started in the field of natural language processing, but it did not take long until the success of transformers was also passed on to computer vision (Dosovitskiy et al., 2020). Subsequently, research exploring the potential of transformer models for multilabel image classification gained momentum. Lanchantin et al. (2021) presented C-Tran, a Classification Transformer framework for multilabel image classification. They discovered that transformers excel at exploiting complex dependencies among visual features and therefore yield improved results (Lanchantin et al., 2021). In contrast, deep neural networks struggled with this task due to their intricate and time-consuming learning processes (Devkar & Shiravale, 2017).

2.3.2 Image Segmentation

The second computer vision technique to be discussed is Image Segmentation which concerns the process of dividing an image into segments. Image segmentation can be divided into two main categories: *semantic segmentation* and *instance segmentation*. Semantic segmentation is the task of classifying each pixel in an image with a semantic label, whereas instance segmentation is the task of partitioning each individual object and providing these with a semantic label (Minaee et al., 2022).

Image segmentation research has a long history, dating back to Rosenfeld and Kak's (1976) exploration of various techniques to perform image segmentation, such as thresholding, edge detection, boundary detecting, region merging, and clustering. In 1979, Otsu proposed a non-parametric and unsupervised method for automatic threshold selection in picture segmentation, in 1988, Kass et al. studied active contour detection methods, and in 1994, Najman et al. successfully applied the watershed algorithm for edge detection. In 2004, Nock and Nielsen proposed statistical region merging as a novel approach for image segmentation and in 2005, Starck et al. proposed an image segmentation technique based on textures and natural scenes. Plath et al. (2009) combined a Conditional Random Field algorithm and an image classification algorithm as a new way to tackle semantic segmentation.

However, like with multilabel classification, deep learning models ushered in a new era of image segmentation, achieving unprecedented accuracy rates (Minaee et al., 2022). In 2015, Long et al. demonstrated that Deep Neural Networks can be trained to perform semantic segmentation using Fully Convolutional Neural Networks. The authors pointed out that the dense layers at the top of a convolutional neural network could be replaced by regular convolutional layers which, in contrast to dense layers, can easily process images of any size (Long et al., 2015). After this milestone, there have been a multitude of successful papers improving the initial performance of convolutional neural networks in image segmentation, leading to the emergence of Region-based Convolutional Neural Networks (R-CNNs) such as Faster R-CNN and Mask R-CNN (Dai et al., 2016; S. Liu et al., 2018; X. Chen et al., 2019; Lee and Park, 2020).

With the introduction of transformer models by Vaswani et al. (2017), the focus of image segmentation research also transitioned from deep neural networks to transformers. In 2021, Zheng et al. successfully applied transformers to the task of semantic segmentation. The authors proposed to approach semantic segmentation as a task of predicting a sequence of outputs, i.e. a sequence-to-sequence task. By using a transformer as the encoder, instead of a convolutional network, and combining that with a simple decoder, they achieved unprecedented results.

2.3.3 Object Detection

The third computer vision method to be discussed is Object Detection, which concerns the identification and localization of objects in an image, as well as assigning these to specific categories. In essence, given an image, an object detection algorithm is capable of determining whether or not an object from a certain category is present, and if so, where it is located (L. Liu et al., 2020).

Initial research efforts in the field of object detection relied on template matching methods and simple part-based models, which primarily focused on specific objects possessing relatively rigid spatial layouts, such as faces (Fischler & Elschlager, 1973). From then until 1990, the prevalent approach was grounded in geometric representations. However, around the switch of the millenium, the focus shifted towards statistical classifiers based on image features such as neural networks (Rowley et al., 1998), support vector machines (Osuna et al., 1997), and boosting techniques (Viola and Jones, 2001; Xiao et al., 2003). This family of object detectors achieved significant success and laid the groundwork for subsequent research.

The transition from detecting objects as global representations to local representations that are invariant to changes in scale, rotation, illumination, viewpoint, and occlusion was achieved by Mikolajczyk and Schmid (2005) and is still considered a milestone. Consequently, local descriptors and fine-tuned pipelines dominated the field for years, until the introduction of Deep Convolutional Neural Networks (Krizhevsky et al., 2012). With the application of deep convolutional neural networks in the field of object detection, remarkable improvements where achieved in detecting more general object categories. In 2014, L.-C. Chen et al. successfully deployed deep neural networks for performing object detection, resulting in the region-based convolutional neural network (R-CNN). Since then, numerous improvements have been made to the region-based convolutional neural network, including Fast R-CNN (Girshick, 2015), Faster R-CNN (Ren et al., 2015), and Masked R-CNN

(Lee & Park, 2020).

Most of the research in object detection has primarily focused on the localization of highly structured objects like cars, faces, bicycles, humans and animals (L. Liu et al., 2020). Unstructured scenes, such as the sky, grass, and clouds, were often overlooked. However, in most recent years, the research community has shifted towards developing general-purpose object detection systems (L. Liu et al., 2020). Nevertheless, despite the significant progress made in this field, the design of accurate, robust, and efficient detection models that can achieve human-level performance across different categories of objects remains an unresolved challenge (L. Liu et al., 2020).

Lately, a new angle of object detection has increasingly gained attention in the research field: the detection of objects in real-time. In 2016 the first major breakthrough was made with the introduction of the You Only Look Once (YOLO) model (Redmon et al., 2016). Ever since, researchers have been building on the initial YOLO architecture to keep improving its performance. Currently, their latest model is the YoloV7 model (C.-Y. Wang et al., 2022). This breakthrough and its constant improvements have allowed for the surge of applications using real-time object detection, such as self driving cars, traffic management and sports analytics.

2.3.4 Image Captioning

The fourth computer vision technique explored in this thesis is Image Captioning, which involves both visual image analysis and the generation of a description for the given image (Hossain et al., 2019). Image captioning is a *multimodal* technique as it translates one data format (image) into another (text). Unlike the other techniques discussed earlier, where the scope of analysis is predefined and limited to a set of objects from a training set, image captioning can generate a tailor-made caption for each image it processes. To generate these customized and meaningful captions, it is crucial for an image captioning model to possess a comprehensive understanding of both syntactic and semantic aspects of language (Vinyals et al., 2017). These characteristics distinguish image captioning as a Generative AI technique, setting it apart from the other techniques employed in this thesis.

Early approaches to image captioning can be categorized into two families (He and Deng, 2017, p. 110). The first approach is based on *template matching*, which entails detecting objects, actions, scenes, and attributes in images, and then filling them into a pre-designed sentence template (p. 110). This approach has been proven successful in studies by Farhadi et al. (2010), S. Li et al. (2011), and Kulkarni et al. (2013). However, generated captions using the template matching approach lacked fluency and expressiveness, and therefore highlighted the need for more advanced techniques in image captioning.

Subsequently, the second approach to image captioning emerged, coined as *retrieval-based* approaches (He & Deng, 2017). Retrieval-based models typically select a set of visually similar images to the new image from a large database, and then transfers the captions of these similar images to fit the new image (Ordonez et al., 2011; Hodosh et al., 2013). However, the limitation of this work was that there is limited flexibility in modifying words based on the content of the new image since the captions directly rely on those of training images. It is not possible to generate novel, tailor made captions (He & Deng, 2017).

This limitation was tackled in 2012, with the introduction of Deep Convolutional Neural Networks, in line with the developments in multilabel classification, image segmentation and object detection (Krizhevsky et al., 2012). The introduction of deep convolutional neural networks paved the way for the generation of coherent and expressive captions, which can be effectively generalized beyond the scope of the training dataset (He & Deng, 2017). This approach does not require additional templates or structures, but instead it relies on high-level image features and word representations, which are derived from deep neural networks and language models, respectively (Hossain et al., 2019).

Kiros, Salakhutdinov, and Zemel (2014) were the first to propose a successful method for generating image captions using deep convolutional neural networks to extract image features. However, they also realised that deep neural language models were inefficient in handling large datasets and long-term memory. To address these issues, Kiros, Salakhutdinov, and Zemel (2014) extended their work by incorporating a Long-Short-Term Memory network for sentence encoding and a novel neural language model for caption generation. This improved approach led to significant enhancements in generating realistic image captions compared to their previous method. A downside was, however, that these networks were slow and computationally heavy.

This limitation lingered until 2017. Just like for the other computer vision techniques, the image captioning research field was impacted greatly by the introduction of the transformer architecture (Vaswani et al., 2017). The transformer technique was extended to the field of image captioning in 2019, when Yu et al. presented a multimodal transformer model with multi-view visual features. As of today, this method is still the standard on which most recent research has been based upon.

2.3.5 Optical Character Recognition

Section 2.2 has demonstrated that text plays a critical role in generating metadata for videos. This encompasses text in titles, descriptions, comments, but also the text that appears in videos. Therefore, in addition to analyzing visual features in images, this thesis must also address the detection and recognition of text that can be seen in the data, such as street signs, company names, or building names. Hence, the fifth and final image processing technique to be discussed is Optical Character Recognition, the task of detecting, locating and outputting text seen in images.

The history of optical character recognition dates back to the early 20th century when mechanical devices were invented to recognize characters (Islam et al., 2017). These early systems were primarily designed to recognize machine-printed text, relying on template matching techniques that compared an image to a library of images, and were limited in their speed and accuracy (Chaudhuri et al., 2017). With the advent of digital computers in the mid-1940s, optical character recognition systems began to take shape, spurred on by the possibility of commercial and business applications (Chaudhuri et al., 2017).

The 1990s marked a turning point in the development of optical character recognition systems, as new tools and methods emerged and were empowered by the continuously growing information technologies (Chaudhuri et al., 2017). During this era, researchers were able to efficiently combine image processing and pattern

recognition techniques with machine learning and artificial intelligence methods. This resulted in the creation of more sophisticated optical character recognition algorithms (Chaudhuri et al., 2017).

Presently, optical character recognition systems have benefited from the advancement of more powerful computers and the availability of more accurate electronic devices such as scanners, cameras, and electronic tablets. These technologies are now paired with modern methodologies such as artificial neural networks, hidden Markov models, fuzzy set reasoning, and natural language processing to further enhance optical character recognition systems (Chaudhuri et al., 2017). Despite these advancements, the ultimate goal of simulating fluent human reading for handwritten and machine-made text remains elusive (Chaudhuri et al., 2017). Most optical character recognition models still have difficulty detecting curved text, text upside down, badly lit text and text from an unknown angle. Hence, there is still much work to be done to reach this objective.

2.4 Recent Developments

In a remarkable development this April 2023, the computer vision field was introduced to InternImage, a state-of-the-art, large-scale, convolutional neural network-based foundation model (W. Wang et al., 2023). InternImage outperforms prior deep convolutional neural network models and transformer-based models in binary image classification, image segmentation, and object detection tasks.

Unlike other recent convolutional neural networks that focus on using large dense kernels, InternImage uses a type of convolution called *deformable convolution* as its core operation. This allows InternImage to have a large effective receptive field (i.e., the portion of the input image that influences the output), while also having adaptive spatial aggregation (i.e., combining information from different parts of the input image in a flexible way). As a result, InternImage has less strict assumptions about the types of patterns it can learn, and can use large-scale parameters to learn more robust patterns from massive amounts of data.

Since InternImage shows unprecedented performance in the most important computer vision tasks, InternImage seems to move the field of computer vision from transformers back to convolutional neural networks. However, since the model has only been released recently and is not yet fully accessible to the public, it remains to be investigated whether InternImage can outperform existing models in multilabel classification and image captioning tasks. Nevertheless, the prospects are promising.

Table 2.1 gives an overview of all papers reviewed in this chapter, and shows for which topics these papers are relevant. The abbreviations stand for MC: multilabel classification, IS: image segmentation, OD: object detection, IC: image captioning, OCR: optical character recognition, and MD: metadata.

Reference	**MC**	**IS**	**OD**	**IC**	**OCR**	**MD**
Chaudhuri et al. (2017)					■	
L.-C. Chen et al. (2014)	■	■	■	■		
X. Chen et al. (2019)		■				
Dai et al. (2016)	■	■	■	■		
Devkar and Shiravale (2017)	■					
Dorai et al. (2006)						■
Dosovitskiy et al. (2020)	■					
Farhadi et al. (2010)				■		
Fernández et al. (2017)						■
Fischler and Elschlager (1973)			■			
Girshick (2015)			■			
He and Deng (2017)				■		
Hodosh et al. (2013)				■		
Hossain et al. (2019)				■		
Islam et al. (2017)					■	
Kass et al. (1988)		■				
Kiros, Salakhutdinov, and Zemel (2014)				■		
Kiros, Salakhutdinov, and Zemel (2014)				■		
Krizhevsky et al. (2012)	■	■	■	■		
Kulkarni et al. (2013)				■		
Lanchantin et al. (2021)	■					
Laven and Hopper (2000)						■
Lee and Park (2020)	■	■	■	■		
S. Li et al. (2011)				■		
X. Li et al. (2004)	■					
Liddy et al. (2002)						■
L. Liu et al. (2020)			■			
S. Liu et al. (2018)	■	■	■	■		
Long et al. (2015)		■				
Maratea et al. (2013)						■
Mikolajczyk and Schmid (2005)			■			
Minaee et al. (2022)		■				
Najman et al. (1994)		■				
Nock and Nielsen (2004)		■				
Ordonez et al. (2011)				■		
Osuna et al. (1997)			■			
Otsu (1979)		■				
Plath et al. (2009)		■				
Rafferty et al. (2015)						■
Redmon et al. (2016)			■			
Ren et al. (2015)			■			
Rosenfeld and Kak (1976)		■				
Rowley et al. (1998)			■			
Shakurov et al. (2021)						■
Snoek et al. (2009)						■
Starck et al. (2005)		■				
Tidake and Sane (2018)	■					
Tsoumakas and Katakis (2007)	■					
Vaidya et al. (2008)	■					
Vaswani et al. (2017)	■	■	■	■		
Vinyals et al. (2017)				■		
Viola and Jones (2001)			■			
Wactlar and Christel (2002)						■
Wactlar et al. (1999)						■
C.-Y. Wang et al. (2022)			■			
J. Wang et al. (2016)	■					
W. Wang et al. (2023)	■	■	■	■		
Wei et al. (2014)	■					
Wendler (1999)						■
Witbrock and Hauptmann (1997)						■
Xiao et al. (2003)			■			
Yu et al. (2019)				■		
Zheng et al. (2021)		■				

Table 2.1: Overview of References and Topics.

Chapter 3

Theoretical Framework

Chapter 2 has revealed that recent advancements in image processing have predominantly utilized Deep Learning techniques such as deep neural networks and transformers. In light of this, this chapter will touch upon the workings of the state-of-the-art algorithms across the five methods outlined in the literature review. A comprehensive, yet concise, explanation of each method will be presented to provide a clear understanding of the underlying principles. Furthermore, this chapter includes a section on Named Entity Recognition as well as a section that delves into the complexities of ensemble learning. Concluding the chapter, a suggested framework will be presented for the effective utilization of pretrained open-source models and the concept behind the cosine similarity metric will be explained.

3.1 Multilabel Classification

Multilabel classification concerns the process of predicting more than one category label for a given input by outputting multiple binary tags (Géron, 2019). For instance, a multilabel classifier that has been trained to recognize three objects, namely a tree, a building, and a car, should, when presented with an image containing a tree and a car, output [1, 0, 1] (Géron, 2019). In contrast, binary and multiclass classifiers can only generate outputs such as [1, 0] or [0, 0, 1] respectively, i.e., only with one positive boolean value. This essential disparity highlights the primary difference between regular classification and multilabel classification (Géron, 2019). Figure 3.1 shows the difference between the classification tasks visually.

This fundamental distinction between multilabel classification and binary or multiclass classification originates in the output function. For regular classification problems, the default output function is the *softmax function*. Consider a basic deep convolutional neural network. The neural network first computes a score, *sk(x)*, for each class k (Géron, 2019). Then, all these scores are passed to a softmax function to estimate the probability of an image belonging to each class k. The class with the highest probability will receive a 1 as output, while assigning zeroes to all other classes (Géron, 2019).

To allow for an output vector with more than one detected class, multilabel classification uses the *sigmoid function* as output function. The sigmoid function will be applied to each output node individually, rendering each output node independent of the others, creating a multitude of binary classification problems. Hence, if there are k classes, there will be k binary classification problems. This approach enables

each output node to possess its own distribution and to be represented in the output vector (Géron, 2019). Thus, if more than one binary classification problem indicates the presence of an object, the output vector can sum up to more than 1.

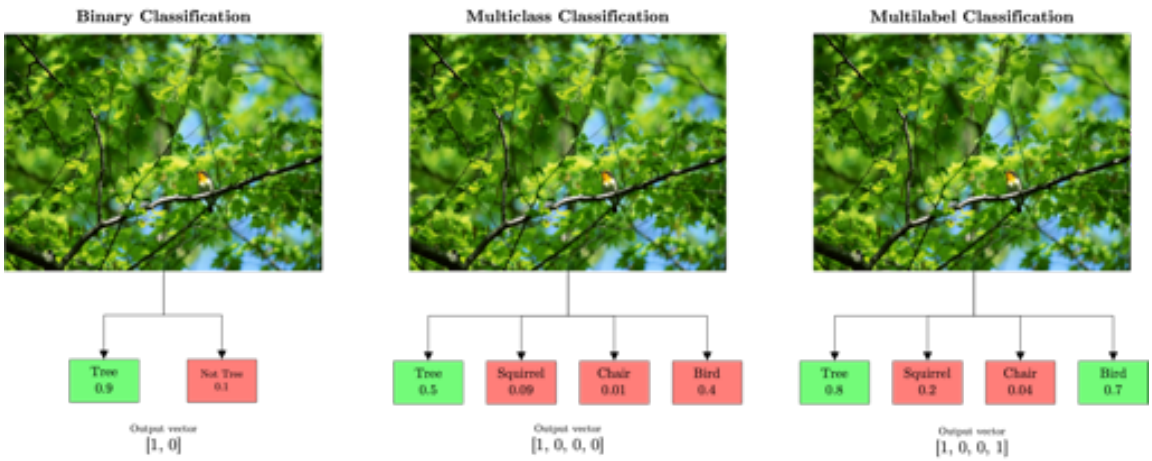


Figure 3.1: Visualization of Different Classification Tasks.

3.2 Image Segmentation

Image segmentation refers to the process of dividing an image into multiple segments (Géron, 2019). In *semantic segmentation*, all pixels belonging to the same type of object are assigned to the same segment (Géron, 2019). For instance, all individuals in an image would be allocated to the "person" category. In contrast, *instance segmentation* assigns all pixels belonging to the same individual object to the same segment (Géron, 2019). Thus, each person in the image would have a separate label or segment. Figure 3.2 offers a visual representation of the difference between the two tasks. In this thesis the focus lies on semantic segmentation, as it is desirable for the generated metadata to comprise of *giraffes* rather than *giraffe 1*, *giraffe 2*, and *giraffe 3*.

To achieve pixel classification, semantic segmentation relies on feature extraction and feature representation, in order to create a segmentation mask (Barla, 2023). The network that is used to extract the features and downsample the image is known as the *encoder*. The network that is used for enlarging at the end of the process is known as a *decoder* (Raj, 2019). For a long time after the discovery of deep convolutional neural networks, convolutional neural networks with an encoder-decoder structure were proven to be the most effective at the task of semantic segmentation.

However, with the introduction of transformer models (Vaswani et al., 2017), the focus shifted to learning how to apply the transformer methodology to computer vision. An example of how the transformer method can be applied to semantic segmentation is proposed by Zheng et al. (2021). In their approach, the input image is divided into small patches, which are fed into the transformer encoder. The transformer encoder produces a sequence of feature vectors based on these patches, which function as input to the transformer decoder to produce the final segmentation map. This method proved to be much more effective than the convolutional neural network-based encoder and decoder structure, since transformers are able to consider

the global context of the image at every step (Zheng et al., 2021).

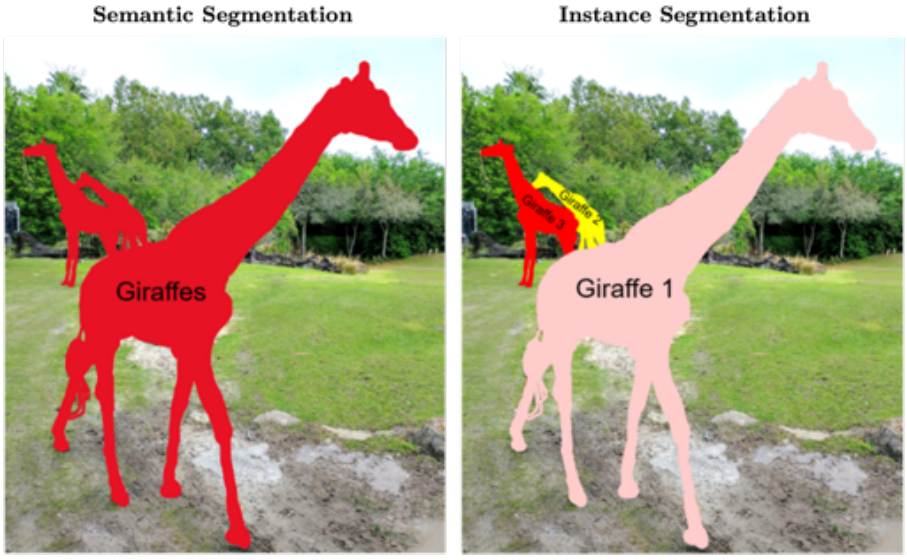


Figure 3.2: Semantic Segmentation Compared to Instance Segmentation (Aiforia, 2021).

3.3 Object Detection

In object detection, the goal is to locate and classify multiple objects within an image (Géron, 2019). Object detection models typically consist of three main components: "a backbone for extracting features from the image, a feature network for further processing these features, and a final network that uses these features to predict the class and location of the object" (Deci, 2021). The final output of an object detection model consists of a bounding box surrounding an object, a class that the object is predicted to belong to, and a score of how confident the model is that it has predicted the object correctly (Figure 3.3) (Deci, 2021).

Despite the fact that object detection models share a common architecture, there are many different approaches to it. The following paragraph will explain the Single Shot Detector (SSD) approach in particular, since the object detection model employed later on in this thesis is a single shot detector (Sandler et al., 2018).

The single shot detector is an object detection architecture that operates by detecting objects in an image through a single forward pass. A single shot detector has two components: a backbone model and a SSD head (Esri, 2023). The backbone model is (usually) a pretrained image classification network which functions as a feature extractor and produces feature maps (Esri, 2023). The SSD head comes after the backbone and consists of one or more convolutional layers and its outputs are the bounding boxes and classes of objects detected in the image (Esri, 2023).

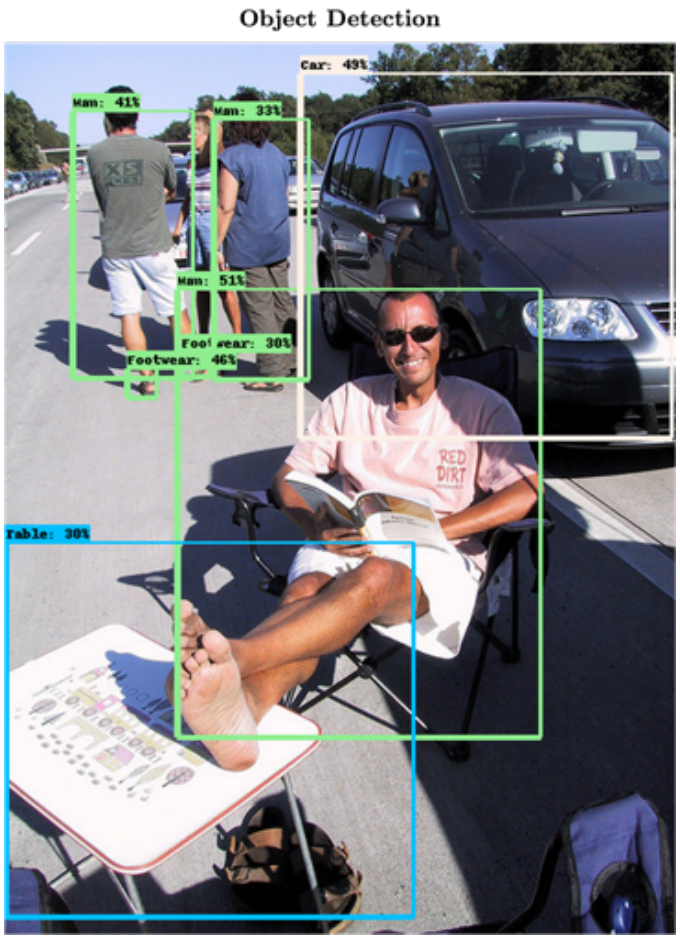


Figure 3.3: Example Output of an SSD Object Detection Model (TensorFlowHub, 2023).

3.4 Image Captioning

Image Captioning involves generating a textual description of the content of an image (Géron, 2019). The process of generating captions involves utilizing both computer vision and natural language processing techniques. Therefore, this technique is categorized as a multimodal approach. The computer vision component is responsible for extracting relevant visual features from the input image, while the natural language processing component generates a textual description based on these extracted features. Through this process, image captioning has the ability to automatically describe the content of images in a human-like manner.

The majority of image captioning models adopt an encoder/decoder framework to generate descriptions of images (Von Platen, 2020). The encoder is responsible for processing the input image and encoding it into a feature vector (Von Platen, 2020). Then the decoder is responsible for generating a descriptive text sequence based on the encoded information, taking the feature vector as input (Von Platen, 2020). In essence, the encoder processes inputs, and the decoder generates outputs.

For a long time, both the encoder and the decoder components were implemented using deep neural networks, where the encoder takes the form of a convolutional neural network, and the decoder takes the form of a recurrent neural network

(Von Platen, 2020). However, with the discovery of transformers by Vaswani et al. (2017), this changed. The default transformer language model replaced the recurrent neural network since it is able to analyze connections between pairs of input tokens, to create captions. With that it resolves the inefficiencies of recurrent neural networks to handle large data (Kiros, Salakhutdinov, and Zemel, 2014 & Siva, 2021). Additionally, transformer based vision models proved to be extremely successful as encoders in image captioning and replaced the convolutional neural networks (Von Platen, 2020; Siva, 2021).

Transformers have the ability to capture complex relationships and dependencies among image features, and have a much lower runtime complexity than neural networks (due to their self-attention feature) (Siva, 2021). Hence, transformers became the new benchmark for image captioning tasks. Therefore, the current state-of-the-art architecture for image captioning includes transformer models both as encoder and decoder, as presented in Figure 3.4.

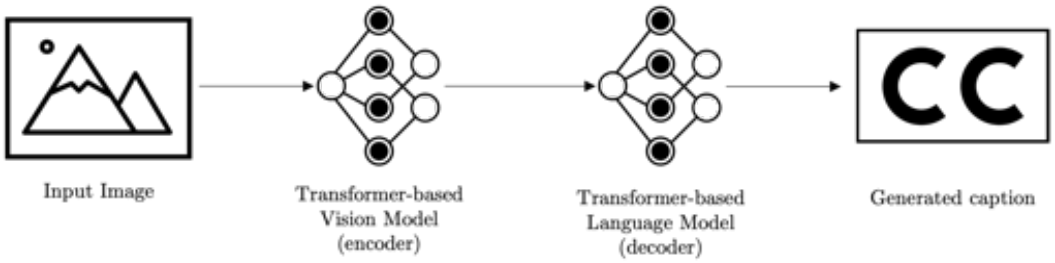


Figure 3.4: Architecture of Image Captioning.

3.5 Optical Character Recognition

Optical Character Recognition concerns "classifying optical patterns contained in a digital image corresponding to alphanumeric or other characters" (Chaudhuri et al., 2017, p. 9). Typically, optical character recognition involves two main steps (IBM, 2022). The first step involves detecting the appearance of text in an image and loading the characters, which are then rendered as a high-contrast bitmap. In the second step, the bitmap is processed by a series of algorithms to extract the characters and convert them into machine-readable text. Various algorithms can be used for this purpose, depending on the specific requirements of the optical character recognition task at hand (IBM, 2022). Like in image captioning and image segmentation, the first network concerning the detection of the text in an image is called the encoder, and the second network concerning the processing of the bitmap, is called the decoder.

Popular bitmap processing algorithms include *pattern recognition* and *feature analysis* (IBM, 2022). Pattern recognition involves training a computer with a large set of known characters to learn all possible variations of the letters. Once the computer has been trained, identifying characters becomes a matter of comparing the identified character with the set of known characters and finding the closest matching one (IBM, 2022). Feature analysis, on the other hand, relies on the specific characteristics of each individual character, such as the number of lines it has, whether it has curved lines, or if any of those lines intersect. Unlike pattern recognition, feature analysis is more rule-based and requires a deeper understanding of the

structure of the characters (IBM, 2022). Depending on the specific requirements of the optical character recognition task at hand, these processing algorithms can take many forms, from a simple perceptron to a complex deep neural network.

3.6 Named Entity Recognition

A notable drawback of optical character recognition is its lack of an internal spelling check mechanism. As a result, the system outputs any characters and numbers it recognizes, regardless of their coherence in forming words or numbers. This may result in erroneous and noisy outputs, such as partial words or single characters that do not convey the intended meaning.

To address this problem, a potential solution is to conduct a spelling check on the output of the optical character recognition model. However, this approach is limited for this thesis since there is a lack of available open source spelling check models for the Danish language. To mitigate the issue, a pretrained Named Entity Recognition Model for the Danish language was employed to filter out most of the misspelled words.

Named entities are a set of items that can be referenced by their name, including persons, locations, and organizations (Jurafsky & Martin, 2022). The most commonly used entity types are person (PER), location (LOC) and organization (ORG) (Jurafsky & Martin, 2022). In cases where an n-gram is not recognized as a particular entity type, the model will simply skip it and proceed to the next (Jurafsky & Martin, 2022). Figure 3.5 displays the results of a classical named entity recognition task on an example text.

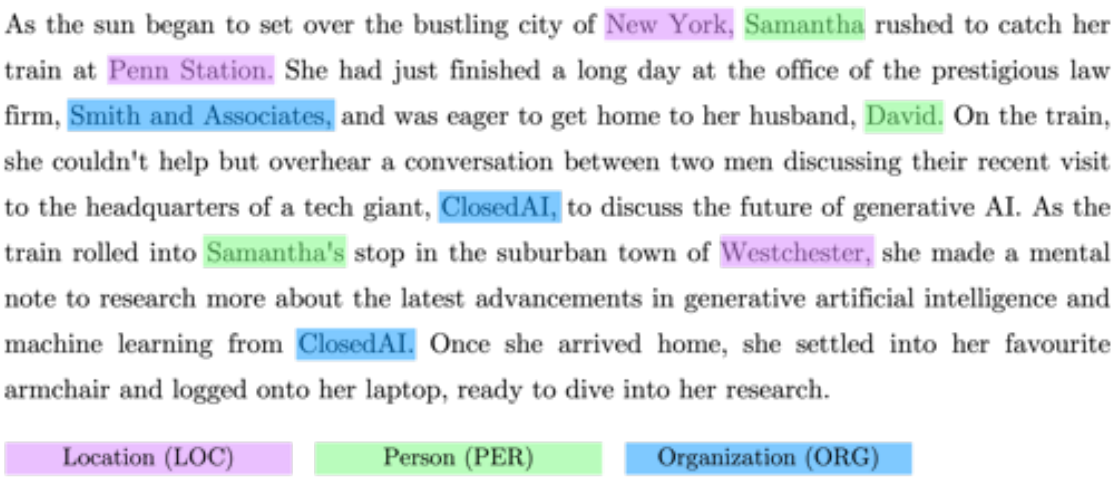


Figure 3.5: An Example of a Named Entity Recognition Analysis.

3.7 Ensemble Learning

As discussed in Section 1.2, this thesis employs ensemble learning as part of its methodology. The central idea of ensemble learning is to combine the predictions of a set of diverse models to obtain superior predictions compared to those obtained from the best individual predictors. Such a collection of predictors is referred to as an *ensemble* (Géron, 2019). Ensemble learning is frequently employed in the

later stages of a project when multiple dependable predictors have been established, aiming to combine them to create an even more robust predictor (Géron, 2019).

In ensemble learning, even if each model performs badly, an ensemble can still perform well if the weak models are sufficiently diverse (Géron, 2019). This means that the different models should have been trained with different training algorithms and on different datasets for the ensemble to be fruitful. In this thesis, each individual model has been trained on a different dataset and follows a different approach, thus allowing for potential synergies (Table 4.4).

In this thesis, the *hard voting* approach is employed. This approach "aggregates the predictions of the individual models and then predicts the output that receives the highest number of votes" (Géron, 2019, p. 251). Although there are various other ensemble techniques available such as bagging, boosting, and pasting, these methods are not feasible for this project as they require specific training techniques. Therefore, this research will exclusively focus on the *hard voting* method as it allows to combine the predictions of the applied models without further modifying their training methods. Figure 3.6 shows an example of how the hard voting approach in this thesis is applied.

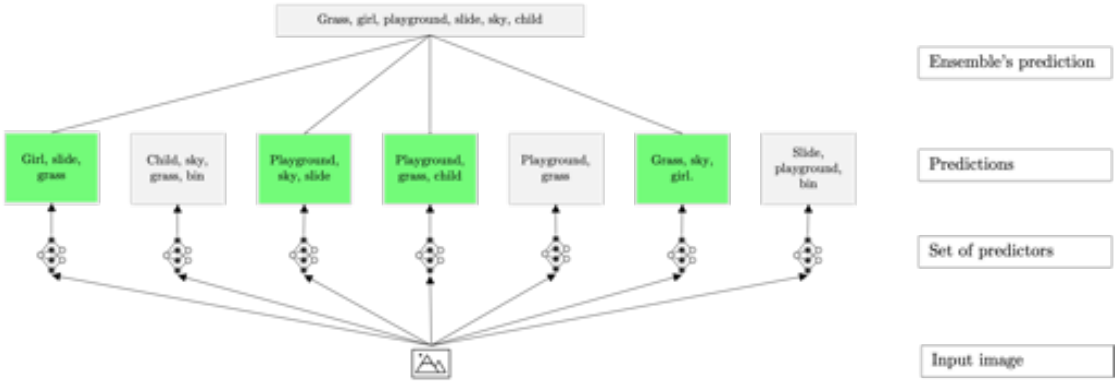


Figure 3.6: Visualization of the Ensemble Learning Approach Employed in this Thesis.

3.8 Leveraging Pretrained Models

In the field of deep learning, it is often not practical to train a large neural network from scratch due to the significant computational resources needed and the time required (Géron, 2019). To address this issue, the use of pretrained models has become increasingly prevalent. Pretrained models are readily trained models that are available for others to perform a task the model was trained to do.

In recent years, the availability of free and open source, pretrained machine learning models has considerably increased, with websites such as HuggingFace, TensorFlowHub and Keras Applications providing a wealth of resources for researchers and practitioners alike. Frequently, the adoption of a pretrained model only requires a brief installation and preprocessing procedure. This efficiency enables researchers to rapidly integrate these models into their studies, bypassing the requirement for comprehensive training.

While using pretrained models offers many advantages, it is unlikely to find a pretrained model that is tailored to the exact task one is currently undertaking. In

that case, one can leverage a technique called *Transfer Learning* by reusing part of the pretrained network and substituting its last layers with ones that suit the new task (Géron, 2019). Although transfer learning is a popular technique, it would still be time constraining to apply it in this thesis, since it would require partly changing the architecture of each model. Furthermore, the pretrained models applied in this thesis are all geared towards image analysis tasks and have been trained on diverse datasets. This provides a solid foundation for the goal at hand (generating metadata) while leaving room for possible synergies in an ensemble. Given these considerations, it was decided by the authors to utilize the pretrained models in their original form, without any major modifications.

3.9 Cosine Similarity

The final concept to be examined in this chapter is *cosine similarity*. In order to evaluate the performance of the AI models utilized in this study, their results are compared with TelevisionCo's Ground Truth.

Cosine similarity is a widely employed metric for measuring the similarity between two vectors by calculating the cosine of the angle between them. This metric is particularly useful for comparing high-dimensional vectors, such as those derived from text data, as it takes into account the orientation of the vectors and not their magnitude (Strehl et al., 2000).

Within the scope of this thesis, cosine similarity is a suitable option for evaluating the generated metadata against the Ground Truth, since it effectively captures semantic similarity between text descriptions, while reducing the effects of differences in word usage or phrasing. Metrics like Jaccard Similarity have the drawback of requiring exact word matches. For instance, these traditional methods would not acknowledge similarity between "car" and "truck", whereas semantic similarity recognizes that both are vehicles and, as such, more similar than "car" and "tree". Moreover, cosine similarity has been successfully implemented in a variety of natural language processing tasks, including document similarity, information retrieval, and text classification (Thompson et al., 2015), which further supports its application in this thesis.

Chapter 4

Methodology

The methodology of this thesis is divided into six parts. Firstly, an overview of the communication with TelevisionCo is presented. Secondly, a systematic project approach is described. Thirdly, an overview of the available data is presented and the computational environment is described. Fourthly, a detailed explanation of the algorithmic workflow design is presented. Fifthly, the ensemble approach used to combine the predictions of the employed models is described. Finally, the evaluation process of the predictions produced by the different models and ensembles is thoroughly explained.

4.1 Business Communication

To establish effective communication with TelevisionCo, a communication plan was designed based on the principal of continuous exchange. This involved regular updates in form of in-person meetings, and frequent email communication. By following this communication plan, the TelevisionCo team was able to provide valuable feedback to the authors of this thesis and contribute to the project's success. An in-depth overview of the content of the meetings between TelevisionCo and the authors of this thesis can be found in Appendix A. A brief summary is presented below.

After an initial online meeting, the first in-person session with TelevisionCo's team started with a concise introduction to machine learning and artificial intelligence, presented by the researchers. This was followed by a collaborative brainstorming session with the entire team from TelevisionCo about potential use cases of artificial intelligence within TelevisionCo (Appendix A, Figure A.1). The ideas generated were subsequently evaluated for feasibility and academic relevance by the researchers. After the thesis topic was chosen, a next meeting was held to examine sample raw data from TelevisionCo and discuss various practicalities. Following this, a project timeline was established during a kick-off meeting, serving as a transparent means of tracking progress, and outlining the support required from TelevisionCo's team, as well as the anticipated milestones (see also Appendix A, Figure A.3). Further meetings were conducted to discuss preliminary results, focus areas, and any upcoming issues and questions.

4.2 Project Approach

In order to carry out the project of automatically generating metadata for videos efficiently, a project architecture was outlined during the initial stages of the project (Figure 4.1). The proposed architecture involved (1) acquiring data, including videos and their corresponding metadata files containing uniform video descriptions. Subsequently, (2a) all relevant video frames and metadata were extracted while (2b) incorporating an option to remove potential frame duplicates. Then, (3) pretrained machine learning models were utilized to perform image processing, (4) followed by a summarization of the descriptions per scene, and the storage of results. Step 2 to 4 were performed in a Python-based pipeline, which will be elaborated upon in Chapter 4.4. (5) After processing all videos through the pipeline, all predictions were appended into one file. (6) The hard voting ensemble approach was employed to generate all possible combinations of the individual models, and finally, (7) the predictions were evaluated against the Ground Truth. The hard-voting ensemble approach and the evaluation methods are explained in detail in Chapters 4.5 and 4.6, respectively.

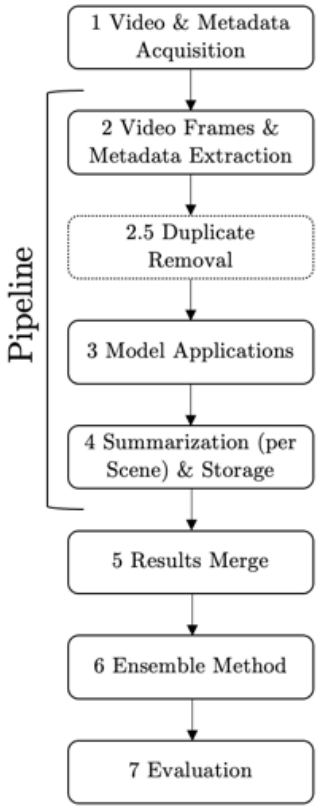


Figure 4.1: Project Architecture

4.3 Data Overview & Resources

Before examining the details of the algorithmic workflow design, this section introduces the data used in this thesis. Table 4.1 presents a brief overview of the described data.

Videos	**Metadata**	**List of Abbreviations & Fixed Terms**	**List of Valuable Object Types**
• 52 videos (63 mp4 files) • HD quality (1280x720 pixels, 25 fps) • 19h, 19.6 GB • 00:01:43h to 01:16:58h	• 52 xml files • technical information • scene descriptions (ground truth) • scene lengths: 1 second to 10 minutes	• 1 list • 34 abbreviations • 17 fixed terms	• 1 list • 10 topics

Table 4.1: Data Overview.

4.3.1 Videos

The cornerstone of this study were the videos themselves, supplied by TelevisionCo, totaling 52. In order to address legal concerns, such as privacy rights, the TelevisionCo team occasionally had to manually remove certain portions of the raw videos, which resulted in some videos being divided into multiple files. This led to a final count of 63 mp4 files, equivalent to 19 hours of footage or 19.6 GB. These videos, extracted from the TelevisionCo archive, were provided in High-Definition (HD) quality with a resolution of 1280x720 pixels at 25 fps. The videos displayed a wide range of durations (from 1 minute 43 seconds to 1 hour 16 minutes 58 seconds), content (covering topics such as interviews, urban pedestrian zones, and winter landscapes), as well as variations in the number and length of scenes (numbers ranging from 5 to 106 and lengths from 1 second to 10 minutes). This heterogeneity enhances the reliability of the evaluation results.

4.3.2 Metadata

In addition to the videos, TelevisionCo supplied the researchers with 52 corresponding metadata files from the archive, formatted in Extensible Markup Language (XML). These files contained various technical details about the videos and their storage within the archive. Additionally, they included valuable information for this project, such as location, date-time (when the video was captured), and a headline. However, during a meeting, TelevisionCo informed the researchers that these details were inconsistently maintained and, as such, could not be integrated into this project.

The metadata files also contained timestamps that marked the beginning of a scene in a video. TelevisionCo clarified that these timestamps were generated automatically when the videographer initiated a new recording, or manually when the videographer pressed a button to indicate the start of a new scene. Each scene was assigned a unique scene ID and description to serve as Ground Truth in the evaluation. To ensure consistency and comparability of metadata descriptions, TelevisionCo assigned three employees to describe all scenes from the 52 videos with their Ground Truth. This was later on used to evaluate the quality of the generated predictions.

4.3.3 Company-Specific Lists

While presenting and discussing initial results with TelevisionCo, the researchers recognized two things: first, TelevisionCo expressed a preference for more detailed descriptions in certain areas and less detail in others. For example, in some cases, TelevisionCo desired high-level descriptions for their scenes, such as "A snow-covered street with people walking through it". In contrast, they sometimes required more detailed descriptions, like "A woman in a red dress approaching a house with a green mailbox" (A comprehensive record of communication instances between the authors and TelevisionCo can be found in Appendix A). In order to determine when a more extensive analysis was needed, the researchers were given a list of valuable object types (Appendix B.2). This list offered information on the 10 most crucial fields to TelevisionCo and the exact facets of interest concerning these fields. Example topics are People, Vehicles, Infrastructure, and Nature.

Second, the company adopted its own abbreviations and fixed terms to efficiently describe general video concepts (e.g., "Næropt." for close-up recordings or "Droneopt." for drone recordings). Therefore, the researchers were also presented with a list of abbreviations & fixed terms, containing 34 unique abbreviations and 17 fixed terms (Appendix B.2).

4.3.4 Computational Environment & Software Dependencies

The execution of all codes and programs associated with this thesis was carried out on a desktop computer, equipped with an Intel Core i7-8700 processor running at 3.20GHz, 16GB of RAM, and a 64-bit operating system, running Windows 11 Home (Version 22H2, Build 22621.1413). Additionally, a NVIDIA GeForce GTX 1080 graphics processing unit (GPU) was used for some of the models.

Python 3.10.9 was used as the programming language, and the following Python libraries were utilized: deep-translator 1.10.1, NLTK 3.7, NumPy 1.23.5, OpenCV-python 4.7.0.72, PaddlePaddle 2.4.2, Pandas 1.5.3, Pillow 9.4.0, requests 2.28.1, Scikit-learn 0.24.1, sentence-transformers 2.2.2, sewar 0.4.5, SpaCy 3.5.1, TensorFlow 2.12.0, TensorFlow-Hub 0.13.0, torch 2.0.0, torchvision 0.15.0, and transformers 4.27.4. The complete project code can be found in Appendix F.

4.4 Algorithmic Workflow Design

In order to effectively apply a range of techniques to each video and secure future reusability for TelevisionCo, a dedicated pipeline was established in Python. This pipeline contains three primary segments: preprocessing, tasks, and summarization, all of which are composed of numerous tailored functions. The pipeline is configured to take one video, together with its associated metadata as input, and then generate a Pandas DataFrame featuring the relevant scenes of the video, the scene IDs, start and end times, the Ground Truth, and the predictions of all implemented models and ensembles. A visual representation of the pipeline is provided in Figure 4.2 and Table 4.2, while the code itself can be found in Appendix F.1 and a more thorough description is provided in the following sections.

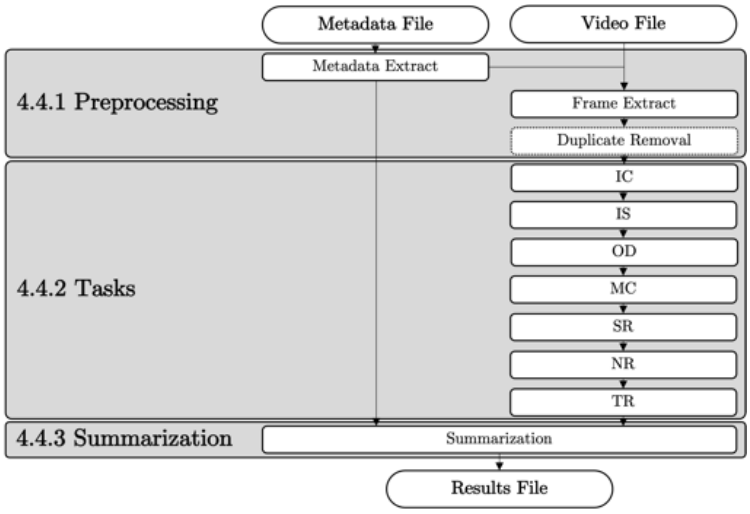


Figure 4.2: Illustration of Algorithmic Workflow.

4.4.1 Preprocessing

Metadata Extraction

The pipeline starts with the *extract_ metadata* function, which accepts a folder path and a video ID as input parameters (see Appendix F.1.2). The folder path designates the storage location of the metadata file. The function processes the specific XML file and compiles all scenes, their associated IDs, start times, and descriptions (Ground Truth) into a Pandas DataFrame.

TelevisionCo's Ground Truth is written in Danish, but the evaluation model (see Chapter 4.6) works better with English text. Therefore, the descriptions must undergo translation to English. The *extract_ metadata* function integrates description translation by utilizing the GoogleTranslator application programming interface (API). To prevent unintended and potentially misleading translations of organization-specific abbreviations, the 15 most prevalent abbreviations (found in Appendix B.1) are excluded from the translation procedure.

The provided XML files include the start times of scenes, but not their end times. An assumption was made that each scene concludes immediately before the start of the subsequent scene, so the end times are deduced accordingly. The end time of the final scene in a video corresponds to the video's total duration.

Lastly, the metadata DataFrame is filtered to retain only relevant scenes. A scene is deemed relevant if it features a scene-specific Ground Truth description. Although all scenes would be pertinent for analysis, predictions cannot be evaluated without a Ground Truth description of TelevisionCo. Additionally, scene unspecific descriptions, such as background information or those encompassing the entire video, are also excluded from analysis as they refer to more than a single scene. Furthermore, scenes with unreasonable duration due to errors or inconsistencies in the metadata files are removed by the function. These filters ultimately reduced the total number of scenes from 1,845 to 952.

The *extract_ metadata* function ultimately returns a Pandas DataFrame (called metadata) containing all relevant scenes of a video, along with their IDs, start times, derived end times, and translated Ground Truth descriptions.

Frame Extraction

The subsequent preprocessing stage involves the extraction of video frames, which is accomplished using the *extract_frames* function (see Appendix F.1.2). Analogous to the metadata extraction, this function requires a folder path (specifying the storage location of the video), a video ID, and the previously established metadata DataFrame as inputs.

When initiated, the function examines the provided folder path, searching for single or multiple mp4 files containing the given video ID. As mentioned in Chapter 4.3.1, TelevisionCo sometimes delivered videos in multiple files due to the removal of legally sensitive scenes. Consequently, this video ID-based lookup within the function enables the collective extraction of frames from a single video ID.

While a considerable part of existing literature performing video analyses concerns key frame extraction (Asha Paul et al., 2018), the researchers in this project opted for a per-second frame extraction. Considering that scene duration can vary significantly (up to 10 minutes, as shown in Table 4.1) and TelevisionCo occasionally

Segment	Function	Input	Output
4.4.1 Preprocessing	extract_metadata	• folder path • video ID • evaluation	• metadata DataFrame • columns: Scene ID, Start Time, End Time, Ground Truth Description
	extract_frames	• folder path • video ID • metadata DataFrame	• frames DataFrame • columns: Timestamp, Frame • raw frames count
	drop_duplicates	• frames DataFrame • MSE threshold	• reduced frames DataFrame • new column: Counter • unique frames count
4.4.2 Tasks	IC	• frames DataFrame	• new column: IC • IC model runtime
	IS	• frames DataFrame	• new column: IS • IS model runtime
	OD	• frames DataFrame	• new column: OD • OD model runtime
	MC	• frames DataFrame	• new column: MC • MC model runtime
	SR	• frames DataFrame	• new columns: SR: Scene Categories, SR: Scores • SR model runtime
	NR	• frames DataFrame	• new column: NR • NR model runtime
	TR	• frames DataFrame	• new columns: TR, NER • TR model runtime
4.4.3 Summarization	summarize_predictions	• frames DataFrame • metadata DataFrame • Video ID	• {Video_ID} _Summarization.csv

Table 4.2: Technical Overview of Pipeline Functions.

required very specific details (which may be visible in only a few frames), this per-second extraction method is better suited for the project.

The function subsequently extracts all frames of a video as NumPy arrays, along with their corresponding timestamps, and stores this information in a Pandas DataFrame (called *frames*).

In the final step, and similar to the metadata extraction process, the frames DataFrame is filtered to retain only frames associated with relevant scenes. For each frame's timestamp, the function checks if it falls within the start and end times of any relevant scene in the metadata DataFrame and removes the frame if this is not the case. This filtering process reduced the total number of frames from 68,338 to 38,790.

Ultimately, the *extract_frames* function returns a Pandas DataFrame containing NumPy array representations of all relevant frames and their timestamps as well as a count of all frames extracted.

Duplicate Removal

Despite the decision to extract frames on a per-second basis, a significant portion of the provided videos featured relatively static scenes (with minimal or no movement). Analyzing each frame of a scene in such cases would be computationally inefficient. To eliminate these redundancies, an optional third preprocessing function, *drop_duplicates*, was introduced (see Appendix F.1.2).

This function accepts the previously extracted frames DataFrame and a user-defined threshold as inputs. It employs an unbiased, pixel-based image comparison between all images and their subsequent images using the Mean Squared Error (MSE) method, as described by H. Zhou et al. (2002). If the MSE between two images surpasses the given threshold, the second image is deemed a duplicate and subsequently removed.

Additionally, a new "Counter" column is added to the frames DataFrame. This column includes a count for each frame, indicating the number of duplicates removed for that specific frame. The function returns the updated DataFrame, without the removed duplicates, but extended with the new column "Counter".

To examine the influence of duplicate removals on both performance and time, the pipeline was run with five different duplicate removal thresholds: no duplicate removal (i.e., no use of the function or an MSE value of 0), light duplicate removal (MSE = 250), medium duplicate removal (MSE = 500), strong duplicate removal (MSE = 750), and extreme duplicate removal (MSE = 1000). Table 4.3 illustrates the impact of duplicate removal on frame counts. Note that the impact of duplicate removal is not that it makes the individual models run faster, it merely decreases the overall runtime since fewer frames are analyzed per scene.

4.4.2 Tasks

Following the extraction of metadata and video frames, as well as the possible elimination of duplicates, a variety of individual models were applied to all video frames. The individual models were selected both on their performance in other projects, and on what dataset they were trained on, where the latter was important to ensure sufficient diversity for the ensemble approach (Section 3.7). The individual

Duplicate Removal Threshold	Raw Frame Count	Unique Frame Count (after Duplicate Removal)	Change (in %)
0	38,790	38,790	0.00
250	38,790	27,764	-28.42
500	38,790	22,945	-40.85
750	38,790	19,654	-49.33
1000	38,790	16,871	-56.51

Table 4.3: Duplicate Removal Effect on Frame Count.

models can be divided into three main categories: Frame Description, Company-Specific, and Text Recognition. The Frame Description category consists of models performing the image captioning, image segmentation, object detection and multilabel classification tasks. The Company Specific category consists of the model performing the scene recognition and night recognition tasks. Finally, the Text Recognition category consists of the model performing the optical character recognition task. These categories and their individual models will be elaborated upon in the following sections, and Table 4.4 provides a brief overview.

Category	Task	Model	Training Dataset	Platform
Frame Description	Image Captioning (IC)	vit-gpt2-image-captioning	COCO dataset	Hugging Face
	Postprocessing (tokenize_IC)	en_core_web_sm	OntoNotes 5	SpaCy
	Image Segmentation (IS)	maskformer-swin-tiny-ade	ADE20K	Hugging Face
	Object Detection (OD)	openimages_v4/ ssd/mobilenet_v2	Open Images V4	TensorflowHub
	Multilabel Classification (MC)	bit/m-r101x3/ imagenet21k _classification	ImageNet-21k	TensorflowHub
Company-Specific	Scene Recognition (SR)	Places365	Places365	GitHub
	Night Recognition (NR)	-	-	-
Text Recognition	Text Recognition (TR)	PaddleOCR	SynthText, ICDAR, COCO-Text, ...	GitHub
	Postprocessing: Named Entity Recognition	nbailab-base-ner-scandi	DaNE, NorNE, SUC 3.0, WikiANN (Icelandic and Faroese parts)	Hugging Face

Table 4.4: Overview of Applied Tasks & Models.

Frame Description Tasks

As explained in Chapter 2 and Chapter 3, the selected frame description tasks comprise Image Captioning, Image Segmentation, Object Detection, and Multilabel Classification. Their integration within the pipeline will be described in the subsequent sections.

Image Captioning For image captioning the pretrained *vit-gpt2-image-captioning* model from Hugging Face was utilized (Singh, 2022). Hugging Face, a leading AI research institution, focuses on the creation and distribution of state-of-the-art machine learning models while advancing the field through open-source platforms and fostering a dynamic community of researchers and practitioners. The *vit-gpt2-image-captioning* model is a hybrid architecture that combines Vision Transformer (ViT) and GPT-2 to generate descriptive captions for images. It was trained on the COCO dataset, an extensive, diverse collection of images depicting intricate everyday scenes and their respective captions.

Consequently, a function named *IC* was developed, taking the frames DataFrame resulting from section 4.4.1 as input (see Appendix F.1.3). Within the *IC* function, frames are converted from NumPy arrays into a format suitable for the model. No further model-specific preprocessing is needed, as the model pipeline (provided by Hugging Face) encompasses all additional preprocessing steps.

The frame captions are presented as sentences, however, all other models generate keywords, and the later summarization process 4.4.3 exclusively utilizes keywords. As a result, the predicted captions require conversion. A supplementary function, *tokenize_IC*, was created to perform this postprocessing transformation. This function takes the updated frames DataFrame as input and removes all stopwords (e.g., "and", "but", "on") using the Python library NLTK (Natural Language Toolkit). Subsequently, the sentences are tokenized, meaning they are broken down into words.

When dividing sentences into unigrams (i.e., isolating each word), it quickly became apparent that this method would lead to a considerable loss of information. For example, if a frame caption states "A yellow surfboard on the beach", removing stopwords and performing unigram extraction would produce the following result: "yellow", "surfboard", "beach". Although this seems to capture the most essential information, it neglects the relationship between "yellow" and "surfboard." To remedy this problem, an additional pretrained NLP model called *en_core_web_sm* was implemented (Explosion.ai, 2023). This small English language model, developed by SpaCy - an open-source library for natural language processing - is trained on the OntoNotes 5 dataset, a comprehensive corpus containing various text genres, such as news, conversations, and weblogs. The *en_core_web_sm* model can discern relationships between words and, when required, extract associated words collectively. In the case of the provided example, the model's output would be "yellow surfboard" and "beach", ensuring that the extra information discovered by the Image Captioning model is not lost during postprocessing.

Ultimately, the *tokenize_IC* function returns the same frames DataFrame with a new "IC" column, containing all tokenized predictions of the image captioning model. Additionally, the total runtime for one video is extracted after applying the image captioning model.

Image Segmentation For image segmentation, the *facebook/maskformer-swin-tiny-ade* model hosted on Hugging Face, which is a semantic segmentation model developed by Facebook AI, was used (Cheng et al., 2021). It employs the MaskFormer architecture in combination with the Swin Transformer (Swin-T) as the backbone. The model is trained on the ADE20K dataset, a diverse and large-scale collection of images encompassing a broad range of scenes, objects, and parts for

semantic segmentation tasks. Similar to the image captioning model, this model also did not require any additional preprocessing.

For this project, a function named *IS* was developed to apply image segmentation to all frames of the input DataFrame, append the predictions to a new column (called "IS"), and extract the model runtime for subsequent comparison (see Appendix F.1.3).

Object Detection Employing object detection as the third task in the frame description category, an additional function named *OD* was developed to first read in all frames from the provided frames DataFrame and subsequently convert them into suitable formats for the chosen *openimages_v4/ssd/mobilenet_v2/1* model (TensorFlowHub, 2023).

Created by Google and hosted on TensorFlow Hub, this object detection model employs the Single Shot MultiBox Detector (SSD) framework along with the MobileNetV2 architecture as its backbone. TensorFlow Hub is a wide-ranging repository of pretrained machine learning models and reusable components, aimed at accelerating the deployment and experimentation of TensorFlow-centric applications in various fields. The model is trained on the Open Images V4 dataset, a diverse and expansive collection of images featuring millions of annotated objects across more than 600 classes.

Following the object detection process, a column ("OD") is appended to the frames DataFrame, which contains all detected objects per frame, and the model's runtime is extracted. The relevant code for this function can be found in Appendix F.1.3.

Multilabel Classification Multilabel classification, the final frame description task, was implemented using the *bit/m-r101x3/imagenet21k_classification/1* model (Kolesnikov et al., 2020). Like the preceding object detection model, this model is also a product of Google and can be found on TensorFlow Hub. It incorporates the BiT (Big Transfer) technique and utilizes a ResNet-101x3 architecture as its foundation. The model is trained on the ImageNet-21k dataset, a diverse and large-scale collection of images covering a wide array of objects, animals, and scenes.

The *MC* function, tailored specifically for this model, reads the frames DataFrame in and preprocesses all frames by first resizing and then normalizing them (see Appendix F.1.3). The labels are afterwards classified on a frame-by-frame basis, with results added to a new "MC" column, and the model runtime is recorded.

Company-Specific Tasks

As discussed in Chapter 4.3.3, the project revealed that TelevisionCo sometimes requires more thorough descriptions and adopts its own abbreviations and fixed terms for video descriptions. In response to this context, both company-specific lists were assessed, and the potential of including additional models to address these TelevisionCo-specific needs was explored. As a result, two additional individual models were employed to generate metadata.

Scene Recognition Following the analysis of the list of valuable object types (Appendix B.2), it became evident that TelevisionCo is particularly focused on de-

tailed scene descriptions. Thus, a new task, scene classification, was implemented and the associated function *SR* was created (see Appendix F.1.3). This function inputs the frames DataFrame, preprocesses all frames to convert them into a compatible format, and then applies the Places365 model to each frame (B. Zhou et al., 2018).

The *Places365* model, hosted on GitHub by the CSAILVision research team (affiliated with the Computer Science and Artificial Intelligence Laboratory (CSAIL) at the Massachusetts Institute of Technology (MIT)), is a deep learning-centric image classification model tailored for scene recognition tasks. It has been trained on the *Places365* dataset, which contains over 10 million images spanning 365 diverse scene categories (B. Zhou et al., 2018). Although scene recognition is technically a multilabel classification task, this particular model's specialization in detecting scenes justifies labeling it as scene recognition.

When analyzing the list of abbreviations and fixed terms (Appendix B.1), the prevalence of abbreviations in the existing dataset was explored. As illustrated by the Abbreviation Count in Figure 4.3, the abbreviations "Ext. opt." and "Int. opt." were commonly used, signifying outdoor and indoor recordings, respectively. The Places365 model is conveniently capable of predicting these scene types as well. The model generates a metric based on the average probability of the 10 most probable sceneries being indoor or outdoor. The outcome of this metric varies between 0 and 1. An explanation of this value's interpretation can be found in Chapter 4.4.3.

In summary, the *SR* function employs the *Places365* model to predict one or more scenes, and an indoor/outdoor-related value. Scene predictions are stored in the "SR: Scene Categories" column, and values are recorded in the "SR: Scores" column. The model runtime is extracted, and both columns are appended to the frames DataFrame, which is then passed on to the next function in the pipeline.

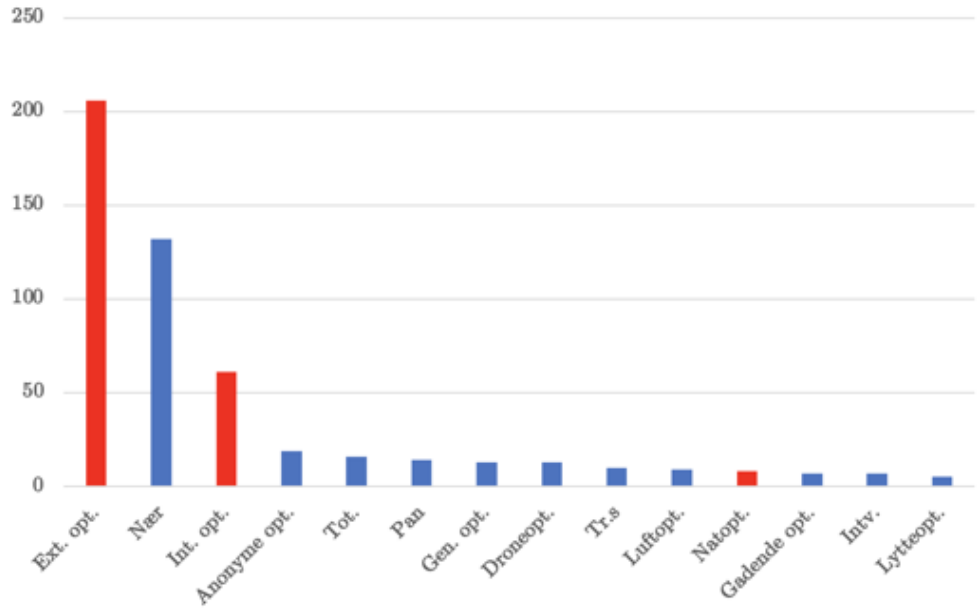


Figure 4.3: Occurrence of Company-Specific Abbreviations in 952 Scenes.

Night Recognition As other abbreviations proved to be inadequately predictable using machine learning models, the researchers chose to create their own method for predicting the "Natopt." abbreviation (English: Night Recording). This task is

unique within this thesis, as it does not utilize a pretrained machine learning model. Instead, the custom function *NR* processes the forwarded frames DataFrame and calculates an average pixel value for every frame (see Appendix F.1.3). These pixel values are then appended to the frames DataFrame in an "NR" column. During later summarization, the values are evaluated to ascertain whether a scene can be identified as a night recording (see Chapter 4.4.3). As with the other functions, this function extracts the model runtime for subsequent comparison.

Text Recognition

The final task performed in this project is text recognition. A function, named *TR*, was developed to accept the updated frames DataFrame as the sole argument and detect text in all frames using the Danish version of the *PaddleOCR* model (PaddlePaddle, 2020).

The *PaddleOCR* model, created by PaddlePaddle (Parallel Distributed Deep Learning, an open-source deep learning platform) and hosted on GitHub, is an Optical Character Recognition system engineered to detect and recognize text within images. The model utilizes the PaddlePaddle deep learning framework and integrates cutting-edge detection and recognition algorithms, such as DB (Differentiable Binarization) for text detection and Recurrent CNN for optical character recognition. It is trained on a variety of extensive optical character recognition datasets, including SynthText, ICDAR, and COCO-Text.

After assessing the preliminary text recognition outcomes, it was determined that a supplementary step was necessary to detect properly extracted text, in addition to the later described frequency-based summarization (4.4.3). Consequently, the *TR* function also incorporates named entity recognition as a postprocessing step for the recognized text.

The *saattrupdan/nbailab-base-ner-scandi* model, available on Hugging Face, is an named entity recognition model developed by researchers from the NBI AI Lab (a research group at the Niels Bohr Institute, University of Copenhagen). This model is specifically tailored for Scandinavian languages and is based on the BERT (Bidirectional Encoder Representations from Transformers) architecture, pretrained on a vast corpus of Scandinavian text and fine-tuned on a custom dataset of annotated text, concentrating on the identification of named entities in Scandinavian languages.

In the *TR* function, the named entity recognition model serves to detect locations and organizations. If any organization or location is identified, it is added to a new column. Consequently, the *TR* function outputs the model runtime and the frames DataFrame extended with a new "TR" (with raw text recognition results) and "NER" column. The *TR* function code is documented in Appendix F.1.2).

Thresholds

Most of the employed individual models not only generate predictions but also compute probabilities for each prediction, signifying their confidence in a specific prediction. During a meeting with TelevisionCo, it was emphasized that noise should be minimized in the predicted metadata. Consequently, it was determined to filter out results during the Model Applications phase of the pipeline (Figure 4.1).

The researchers conducted a multitude of small tests to identify suitable probability thresholds for various models.

Since all models exhibit different levels of certainty, distinct thresholds were applied. For instance, image segmentation outputs (4.4.2) were filtered for predictions with a probability greater than 90.00%, while the object detection task (4.4.2) employed a filter greater than 10.00%. Predictions from the multilabel classification model (4.4.2) were filtered for probabilities higher than 1.00%, and recognized scenes from the Places365 model (4.4.2) required surpassing a 50.00% threshold. To reduce noise from the text recognition model (4.4.2), it was additionally decided to exclude any text with a single-character length. These filters ensured that only relevant predictions proceeded through the pipeline.

4.4.3 Summarization

After applying all individual models to each frame of a video, it was necessary to consolidate the frame-wise predictions into a single scene prediction for each scene. To accomplish this, a final function called *summarize_predictions* was integrated into the pipeline. This function takes as input the frames DataFrame, now containing all results from the different models, the earlier extracted metadata DataFrame, and the respective video ID. First, it calculates the scene ranges for all pertinent scenes of a video using the start and end times from the corresponding metadata DataFrame. Subsequently, it determines which frames belong to each scene and consolidates these multiple frame predictions into a single scene prediction.

In this thesis, six out of the seven individual models generate keywords as their output, with the Image Captioning model being the only one producing sentences. The Ground Truth metadata provided by TelevisionCo is also in the form of sentences. Consequently, it was initially opted to employ a separate AI model to convert the keyword-based outputs from the six individual models into sentences, and use these sentences as the final metadata predictions. However, this approach yielded unsatisfactory results. Upon presenting these findings to TelevisionCo, the company indicated their preference for a keyword-based output in the ultimate chosen model or ensemble, rather than sentences. As a result and as discussed in Section 4.4.2, the output of the Image Captioning model required additional post-processing compared to the other techniques before it could be summarized. The summarization procedures for each task will be detailed in the subsequent sections.

Summarization of Frame Description Models

For all individual models from the frame description category, the summarization technique is uniform: for each scene, the *summarize_predictions* function examines if the selected model has produced a prediction. If a prediction is made, the predicted keyword(s) (or phrases, referred to as words for simplicity) are added to a list of scene keywords. Moreover, the frequency of predictions made by the specific model per scene, is counted.

After incorporating all relevant results into the list of scene keywords, the occurrences of all unique words are analyzed. Subsequently, two filters are employed. Firstly, the extraction of only the n most frequent words of a scene is performed. The parameter n was identified for each model individually through extensive word count delimitation tests. In these tests, the results of all models from the frame

description category were assessed (using the function from Section 4.6) with word counts ranging from 1 to 10 per scene. The word count delivering the optimal performance was ultimately selected as n for each individual model from the frame description category: 6 words for image captioning and image segmentation and 4 words for the object detection and multilabel classification models. The outcomes of these word count delimitation tests can be observed in Appendix C while the relevant code is documented in Appendix F.3.

The second filter was implemented to address TelevisionCo's request for more robust noise removal. This filter takes into account the relative frequency of words by calculating the frequency of a word being predicted in a scene relative to the total number of predictions made for that specific scene. Consequently, only words that occurred in more than 10.00% of all predictions were considered in the summarization. This approach prevented the extraction of words that might be most frequent in absolute terms but still relatively insignificant.

In conclusion, all predictions of the models from the frame description category are summarized by extracting the n most frequent (unique) words that exceed a relative frequency threshold of 10.00%.

Summarization of Company-Specific Tasks

The summarization strategy for the individual models in the company-specific category differs from that of the models from the frame description category and will be explained in the following two sections.

Summarization of Scene Recognition As discussed in Chapter 4.4.2, the scene recognition (*SR*) function adds two columns to the frames DataFrame: one column for scene categories ("SR: Scene Categories") and another for scores ("SR: Scores"). While the scene categories are summarized similarly to how the keywords from the individual models in the frame description category were summarized, the scores require an alternative approach.

To recap, scores determined in scene recognition range from 0 to 1, denoting the likelihood of a frame scene being indoors (or outdoors). The developers of the *Places365* model recommended using a single 0.5 threshold, with values below 0.5 indicating indoor scenery and values between 0.5 and 1 suggesting outdoor scenery. However, the data in this project was classified into three categories: scenes marked with the abbreviation "Int. opt." (indoor recording), scenes with a description including "Ext. opt." (outdoor recording), and scenes without any specific indication of being inside or outside. Thus, it was essential to identify two new thresholds to determine whether a scene was indoors, ambiguous, or outdoors. In a scene score threshold delimitation test, scores for every frame of those scenes with Ground Truth descriptions including "Ext. opt.", "Int. opt.", or neither of those were computed. Subsequently, these values were aggregated scene-wise by calculating their average, and the averages were stored alongside the respective scene descriptions. Categorizing scene descriptions into these three groups enabled the analysis of value ranges assignable to each category.

As illustrated in Figure 4.4, a distinct separation between the distributions of the "Ext. opt." and "Int. opt." categories can be observed. Regrettably, the "Unspecific" category did not exhibit a discernible pattern. The researchers attributed this anomaly to inconsistent descriptions or the subjective nature of determining

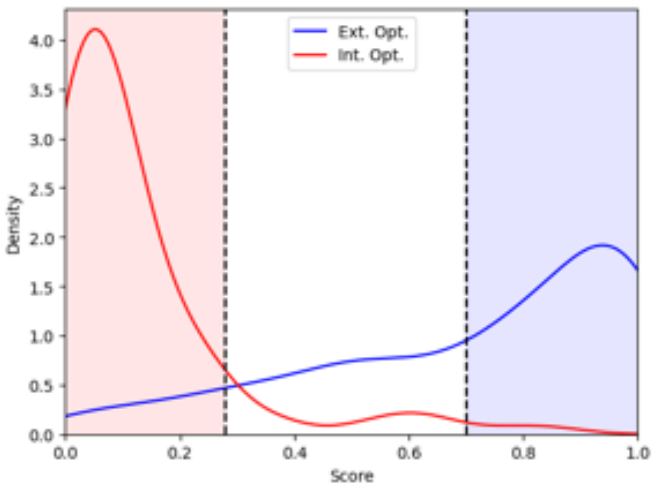


Figure 4.4: Occurrence Distribution of "Ext. opt." and "Int. opt." with Delimited Thresholds at x=0.28 and x=0.7, Respectively.

when a scene can be classified as unspecific. As displayed in Figure 4.4, the value of 0.28 was chosen as the first threshold (between "Int. opt." and the unspecific category), while the second threshold was established at 0.7 (separating the unspecific category and "Ext. opt.").

Within the *summarize_predictions* function, these thresholds are integrated accordingly. For each scene, the scores are aggregated by computing the average, and these average values are converted into either "Ext. opt.", "Int. opt.", or no output, depending on the value range they fall within. Thereafter, both elements of the scene recognition summarization (transformed scores and scene categories) are merged into one output.

Summarization of Night Recognition Comparable to the scene recognition scores, the night recognition approach also extracted values, specifically average pixel values. In a pixel value threshold delimitation test, a threshold was determined by initially calculating all average pixel values for all frames in the dataset, grouping these per scene and calculating the average, and then storing these findings together with the scene descriptions. The data was then partitioned into two categories: scenes of which the Ground Truth metadata contained "Natopt." ("night recording") and scenes of which the Ground Truth metadata did not contain "Natopt.". Figure 4.5 plots the distribution of these values per category and demonstrates a pronounced difference between categories, resulting in the establishment of 54.15 as the demarcating threshold.

In the summarization function, it is examined to which scene a frame belongs. The average pixel values of all such identified frames are added to a list. Subsequently, the mean of all values in this list is calculated and depending on whether the mean falls above or below the threshold, "Natopt." will be generated and extracted for the particular scene.

Summarization of Text Recognition

In the final stage of the summarization process, the text recognition results must be summarized. The *summarize_predictions* function begins by aggregating the

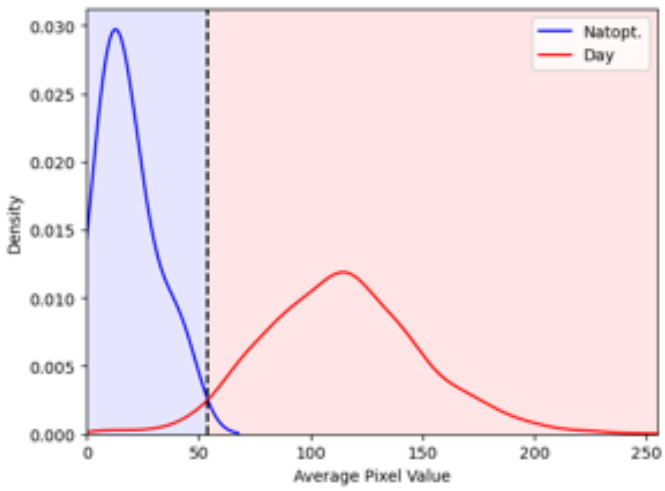


Figure 4.5: Occurrence Distribution of "Natopt." and without ("Day") with Delimited Threshold at x=54.15.

detected named entities from the "NER" column. For this, all output for one scene are added to a list, and subsequently, the unique entities are extracted per scene. No additional filters were applied in this summarization, as it was assumed that if the NER model identifies an organization or location, it can be considered accurately extracted text and thus valuable information for the metadata.

Following this, the "TR" column is analyzed. The summarization approach for the recognized text resembles that of the models in the frame description category but differs in three aspects: (1) no maximum word count is defined, and (2) no relative frequency threshold is implemented. These thresholds were excluded at this stage since optical character recognition often captures a snapshot of a scene. Frequently, the text is recognized in a few frames or even just one frame, and therefore, many individual predictions could exist, all of which might be of interest. Instead of these frequency-based filters, (3) the summarized unique words are filtered for already extracted named entities to avoid duplicates.

After merging the "NER" and "TR" summarizations, a word limit is established. The optimal word count was determined in the same manner as for the models in the frame description category. However, an initial test indicated an optimal word count of 10, which was also equal to the maximum amount of words tested. Consequently, a second test was conducted, testing word counts from 1 to 20. This test confirmed that indeed, 10 words would be the optimal amount for this specific summarization. Both test results can be observed in Appendix C.

Duplicate Removal Impact in the Summarization Function

As outlined in section 4.4.1, the pipeline was executed with five distinct duplicate removal variations: none (MSE=0), light (MSE=250), medium (MSE=500), strong (MSE=750), and extreme removal (MSE=1000). To recap, when a frame was identified as a duplicate and subsequently removed, a value was assigned to the remaining frame in the "Counter" column, indicating the number of duplicates represented by that specific frame. The *summarize_predictions* function takes this value into account by assigning higher weight to these duplicated frames, which has various impacts on the summarization approaches.

In the summarization approach of the models in the frame description category, a prediction is added to the list of scene keywords as many times as the number of duplicates it represents, instead of just once. Additionally, this prediction will be counted for as many times as it represents duplicated frames. This ensures that duplicate frames receive the appropriate attention, as they represent multiple frames. Since the summarization methods for the categorization part of the scene recognition model, and the optical character recognition part of the text recognition model are nearly identical, the effect of duplicate removal is the same in these models.

In the process of summarizing the named entities as part of text recognition, the potential duplicate removal does not have an impact, as no quantity-based rankings or filters are employed in this task. For summarization methods of the value-driven part of the scene recognition model (scene scores) and of the night recognition model (average pixel values), the duplicate removal affects the frequency with which a single frame's value is considered in calculating the scene-wise average.

After finalizing the summarization for the specific models, a new column featuring the moldel's name is appended to the metadata DataFrame, encompassing all summarized predictions for each scene. Moreover, the summarizing function adds a new column to the metadata DataFrame that includes the video ID.

As the concluding step in both the summarization and the overall algorithmic workflow, the metadata DataFrame containing all summarized predictions is saved and stored as a CSV file in the following format: "Video ID_Summarizations.csv". Furthermore, a "Runtime.csv" is generated, encompassing the runtimes for all models and videos, along with the raw video frame count (derived from the *extract_frames* function, as described in Chapter 4.4.1) and the unique video frame count (following duplicate removal, as detailed also in Chapter 4.4.1).

4.5 Ensemble Method

4.5.1 Model Ensembles

After each video and metadata file had been processed through the entire pipeline, all generated output was combined into one Pandas DataFrame, which included all relevant scenes from all videos (as denoted by two columns: "Video ID" and "Scene ID"), accompanied by their specific start time, end time, and Ground Truth description. Additionally, the DataFrame also featured the summarized predictions of all seven individual models, with each model's results stored in its own column. The consolidated DataFrame contained 952 scenes. However, since the DataFrame also encompassed scenes without available video material, 32 prediction rows were empty and had to be eliminated. This process ultimately reduced the overall number of scenes from 952 to 920.

In order to leverage the ensemble method, it was necessary to combine the results from the employed individual models. Considering that company-specific terms and abbreviations are predominantly found at the start of TelevisionCo's descriptions, the decision was made to restructure the model columns such that the SR and NR models consistently serve as the first individual model(s) in an ensemble. Then, all unique model combinations were determined. The total number of possible combinations for this project equaled the sum of the binomial coefficient $\sum_{k}^{n} n!k!(n-k)!$ with $n = 7$ (the number of models) and k ranging from 1 (employing

only one model) to 7 (merging all 7 models). This could also be computed by $n^2 - 1$ yielding 127 unique combinations (120 ensembles and 7 individual models) with each individual model appearing in 64 combinations. "Unique" implies that a combination of several models was established in a single, unambiguous way. For instance, the results of the image aptioning and image segmentation models would only be combined once, either as "IC_IS" or "IS_IC", but not in both forms.

After calculating these 127 combinations using Python's itertools library, the predictions of all individual models were combined, all ensembles were added to the DataFrame as new columns with their predictions as rows. This process was replicated for all duplicate removal variations, and the resulting five DataFrames were merged to produce a comprehensive "All_Results" DataFrame containing a total of 635 unique result columns (in addition to the "Video ID", "Scene ID", "Start-Time", "End-Time", and "Ground Truth" columns).

4.5.2 Runtime Ensembles

As all individual model results were combined, it was necessary to create artificial ensemble runtimes for comparing the ensembles' runtimes. To achieve this, all five "Runtime.csv" files were imported, the models' runtimes were aggregated, and an "average runtime per frame" was computed. Following this, artificial runtimes for all ensembles were generated by adding together the runtimes of all models included in an ensemble. In a new "Runtime_Evaluation.csv" file, all 635 ensembles and their runtimes were assembled in the "Model/Ensemble" and "Runtime per Frame" columns, respectively.

The developed and applied code for the Ensemble Method can be found in Appendix F.4.

4.6 Evaluation

With an "All_Results.csv" DataFrame containing 7x5 model results and 120x5 additional ensemble results for 920 scenes, the next step was to evaluate the quality of these predictions. To accomplish this, another Python function, *evaluate_predictions*, was developed (see Appendix F.5). This function accepts the final results DataFrame as input and computes a sentence similarity score using the *sentence-transformers/all-mpnet-base-v2* model.

Hosted on Hugging Face, this model is a sentence embedding model developed by the Sentence Transformers team (a group of researchers and developers led by Nils Reimers). The model is trained on various datasets, including NLI (Natural Language Inference) and STS (Semantic Textual Similarity). Utilizing this model, a sentence similarity score is derived by calculating the cosine similarity between each scene description (serving as Ground Truth) and all corresponding model or ensemble results. These similarity scores span between 0 (no similarity) and 1 (complete similarity). Upon inspecting initial results, it was observed that, for short descriptions (or Ground Truths), empty-string model and ensemble results received unexpectedly high scores (greater than 0.25). Since a lack of prediction would be of no value to TelevisionCo, these elevated scores would skew the overall results for this project. Consequently, similarity scores in such cases

were set to 0. Ultimately, the scores for all 920 scenes are stored in 635 new score columns within a Pandas DataFrame, and this DataFrame is extracted and saved as "All_Results_Evaluated.csv" by the function.

After applying the evaluation function, all scores were aggregated by calculating a mean value and stored in a "Scores_Evaluation.csv" file. Both the "Runtime_Evaluation.csv" (from Chapter 4.5.2) and "Scores_Evaluation.csv" files were merged to create a "Total_Evaluation.csv" DataFrame, consisting of 635 rows for each model or ensemble and 3 columns: "Model/Ensemble" for the combination names, "Scores" containing the respective average score values, and "Runtime per Frame" containing the runtime one ensemble requires to perform on a single frame. This final DataFrame was employed to evaluate the performances in terms of quality and speed.

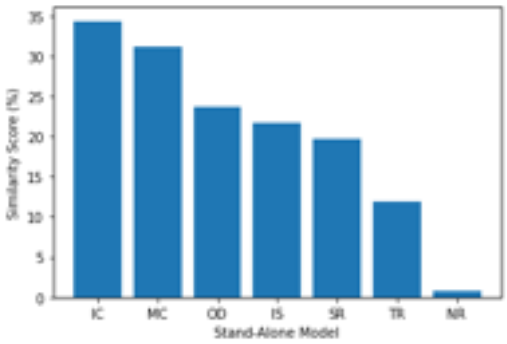
Chapter 5

Results

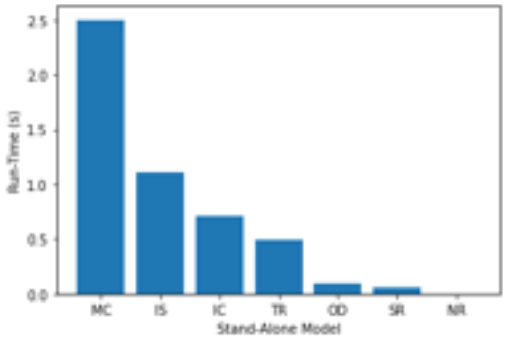
The purpose of this chapter is to present the results of the research. Those findings relevant to formulating an answer to the research questions will be highlighted, and in Chapter 6 these will be discussed. Specifically, this chapter will report on results concerning the performance of the different individual models, the different ensembles, and the optimal size of an ensemble. Additionally, the effect of duplicate removal and raw examples from the data will be presented. This chapter will round up by presenting example predictions of the models and ensembles, and their corresponding scenes.

5.1 Models and Ensembles

Figure 5.1a and 5.1b display the average performance and runtime, respectively, of the seven individual models across all scenes, in a setting with no duplicate removal (i.e., an MSE=0). In terms of performance, Figure 5.1a provides valuable insights into the quality of each individual model in generating metadata. It shows that image captioning and multilabel classification are the two highest performing individual models, reaching an average similarity score of 34.33% and 31.15%, respectively. Text recognition (TR) and night recognition (NR) are the two worst performing models with average similarity scores of 11.90% and 0.85%, respectively.



(a) Average Performance per Individual Model.



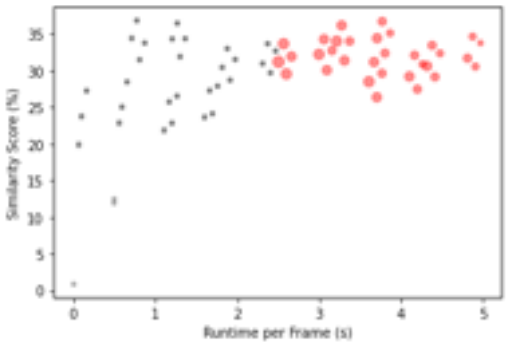
(b) Average Runtime per Frame per Individual Model.

Figure 5.1: Performance and Runtime Overview for each Individual Model.

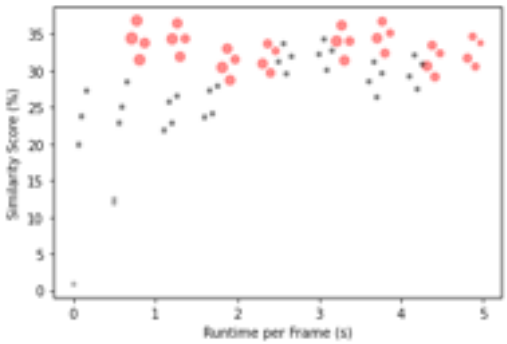
In terms of runtime, Figure 5.1b provides valuable insights into the time it takes for each individual model to generate metadata for one frame. It shows that

there is a big range between the fastest and the slowest model: multilabel classification requires 2.501 seconds per frame, whereas night recognition requires 0.002 seconds per frame. Image Captioning, the best model performance-wise, needs 0.709 seconds per frame to generate metadata. Figure 5.1a and 5.1b together reveal the trade-offs between performance and efficiency of each model, which could influence the future selection and optimization of the models.

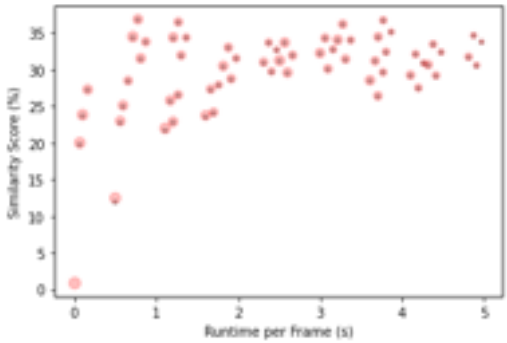
To further investigate the performance of the different methods, Figure 5.2a, 5.2b and 5.2c display scatter plots, each providing insights into the performance of the ensembles, and highlighting one specific model in red. Figure 5.2a shows the scatter plot of the average performance of all ensembles with no duplicate removal (i.e., MSE=0), with the red markers indicating the ensembles that include the multilabel classification model. Similarly, the red marked data points in Figures 5.2b and 5.2c represent ensembles that include the image captioning and the night recognition models, respectively.



(a) Scatterplot Highlighting Ensembles with MC (MSE=0).



(b) Scatterplot Highlighting Ensembles with IC (MSE=0).



(c) Scatterplot Highlighting Ensembles with NR (MSE=0).

Figure 5.2: Scatterplots with Specific Models Highlighted in Red.

In Figure 5.2a it can be observed that, in general, the ensembles of which multilabel classification is a part of, perform relatively well, but at the same time have a relatively long runtime. This observation is consistent with the results in Figures 5.1a and 5.1b

Figure 5.2b, on the other hand, sheds light on the ensembles that include image captioning models. The figure shows that all highest performing ensembles include the image captioning model. Although there are a multitude of ensembles with image captioning that reach the highest average performance, the figure also

shows that some ensembles have a considerably lower runtime than others, revealing that the trade-offs between performance and efficiency in ensembles with image captioning models can be assigned to other models in the ensemble, and not image captioning.

Finally, Figure 5.2c reveals that the impact of night recognition on the performance of the ensembles it belongs to is relatively limited. Its ensembles are dispersed throughout the entire figure, and in each ensemble, night recognition only marginally enhances the performance compared to the same ensemble without night recognition. However, it is important to note that the time required to achieve this minor improvement is minimal, as can be seen in Figure 5.1b. Consequently, it can be argued that the added value of night recognition is realized at virtually no cost.

Moving on, Figure 5.3 presents an overview of the performance of all ensembles in the case of no duplicate removal (i.e., MSE=0). Figure 5.3 is an upset plot, which is a plot that provides two key pieces of information: 1) the upset plot displays the performance in terms of average similarity score of each ensemble across all scenes, 2) the upset plot indicates which models are included in each ensemble.

Figure 5.3 shows us that the best performing ensemble is one that includes image captioning, scene recognition, and night recognition models, yielding an average similarity score of 36.85%. Conversely, the worst performing ensemble is the individual night recognition model, with an average similarity score of only 0.85%. This particular finding goes hand in hand with the bar graph presented in Figure 5.1a which highlights the relatively poor performance of the night recognition model compared to other models.

The analysis of Figure 5.3 also reveals that, in general, individual models tend to have lower performance compared to other ensembles. Specifically, the individual models of night recognition, text recognition, scene recognition, image segmentation, and object detection find themselves among the bottom ten percent (114-127). Following this, multilabel classification also ranks in the bottom half of all ensembles. Notably however, the individual image captioning model exhibits exceptional performance, ranking in the top 20 among all ensembles. This re-demonstrates the dominance of the image captioning method.

To further investigate the top performing ensembles, Figure 5.4 displays the top 10 best performing ensembles in a setting of no duplicate removal (i.e., MSE=0). Figure 5.4 plots the average performance across all scenes for each ensemble, as well as the average runtime per frame for each ensemble. The analysis of this figure reveals that the two best performing ensembles are also relatively fast compared to other models in the top 10. The eight other models in the top 10, are considerably slower compared to the top two models, while performing rather similar.

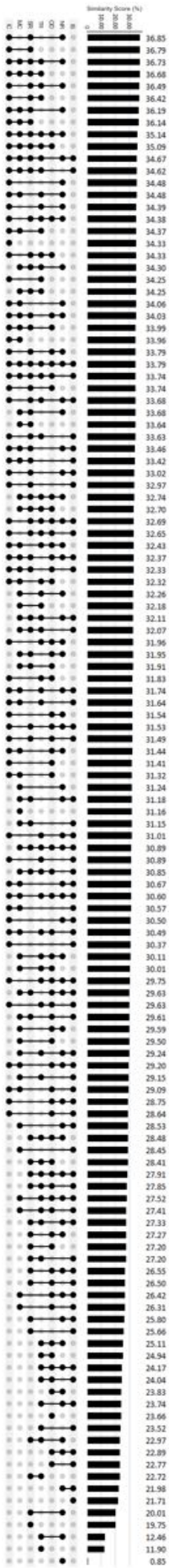


Figure 5.3: Average Performance per Ensemble at MSE=0

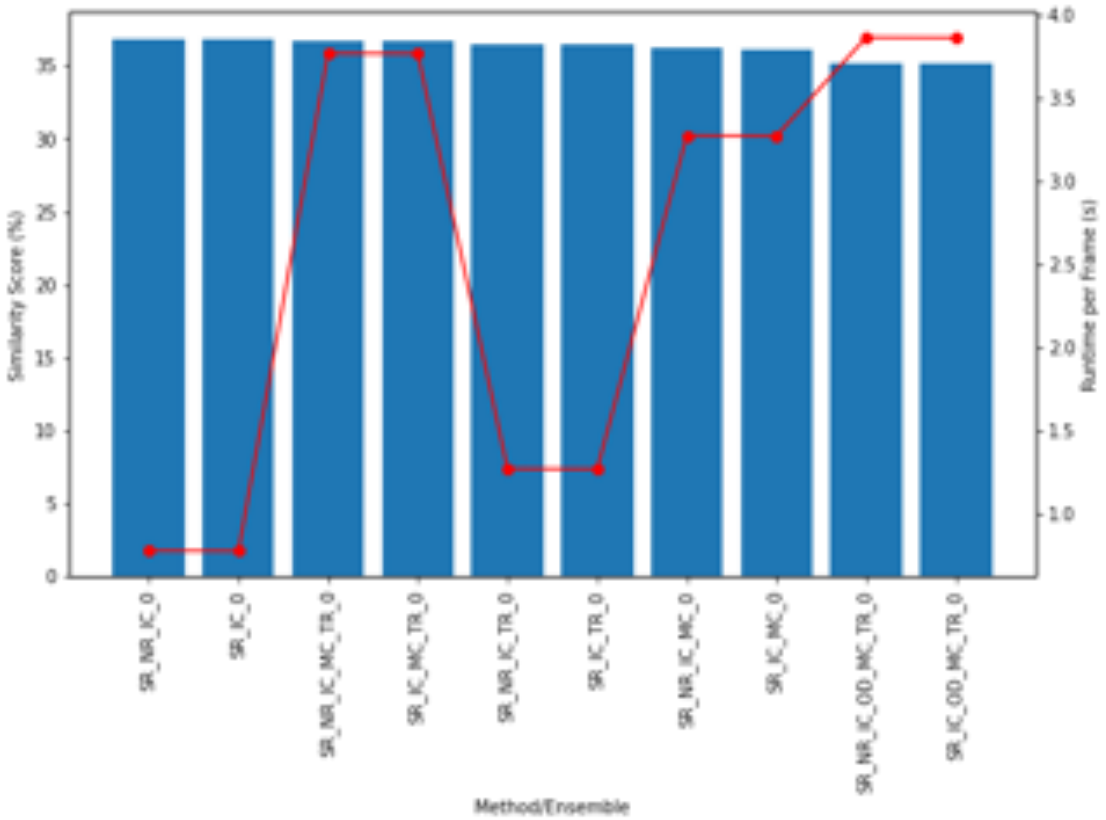


Figure 5.4: Average Similarity Score and Average Runtime per Frame for Top 10 Performing Ensembles.

Lastly, this study examined the possibility of a correlation between performance and scene length within the dataset (Appendix F.6). It is expectable that the models' performance may be influenced by the duration of a scene, either positively or negatively. Improved performance could result from the availability of more information to analyze, while a decline in performance might stem from information loss due to the summarization approach. Nevertheless, the findings of this investigation, presented in Appendix D.1, indicate that there is no apparent correlation between performance and scene length.

5.2 Duplicate Removal

In the following paragraphs, the performance of the ensembles with different degrees of duplicate removal will be reported on. This topic is explored to assess the necessity of evaluating all frames in a scene for accurate metadata generation, and to assess the possibilities for relying on key frames only. To this end, five different gradients of duplicate removal were tested, corresponding to five different Mean Squared Error values (MSE): 0, 250, 500, 750, and 1000. By exploring the performance of the models at the different degrees of duplicate removal, we can assess the trade-offs between accuracy and computational efficiency in metadata generation for digital video archives.

Figure 5.5 provides an overview of the effect of duplicate removal on the data quantity. The figure illustrates that at an MSE value of 250, approximately 5.00% of

the total scenes are lost. At the highest MSE value (i.e., MSE=1000) approximately 12.00% of the total scenes are lost. These 12.00% of scenes correspond to a loss of 56.51% of frames compared to the number of frames in the setting of no duplicate removal (MSE=0).

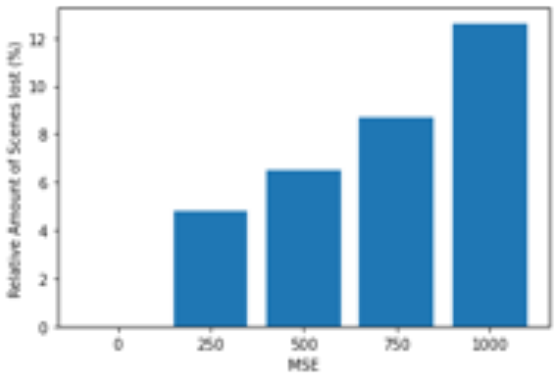
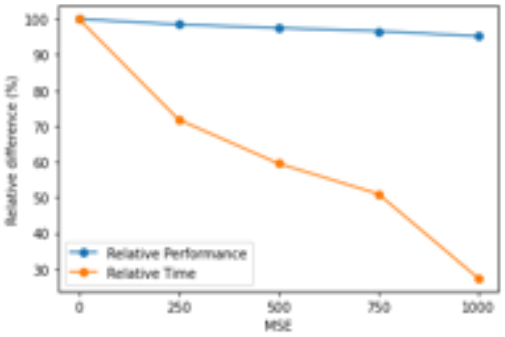


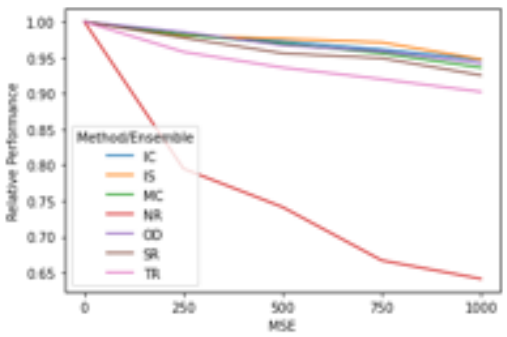
Figure 5.5: Information Loss per MSE Value.

To further investigate the impact of different levels of duplicate removal on the performance of AI models, the average performance across all scenes at different MSE values was analyzed. Figure 5.6 presents the findings of this analysis. Figure 5.6a plots the relative performance and relative runtime per frame, with the situation of no duplicate removal (MSE=0) as the baseline. Figure 5.6b shows the same relative performance, but then across the seven different individual models.

Figure 5.6a shows that, while the relative runtime decreases considerably with each degree of duplicate removal, the average performance across all scenes does not suffer of more than approximately 5.00% performance loss. Figure 5.6b presents the average performance across all MSE values for each individual model. This figure reveals that the trend observed in the previous figure holds for most individual models across all different MSE values. For the individual model of night recognition however, the figure shows a different trend. It can be seen that the performance of night recognition suffers considerably, the higher the MSE values are. Moreover, the individual text recognition model also shows relatively poor performance compared to other models, the higher the MSE value gets.

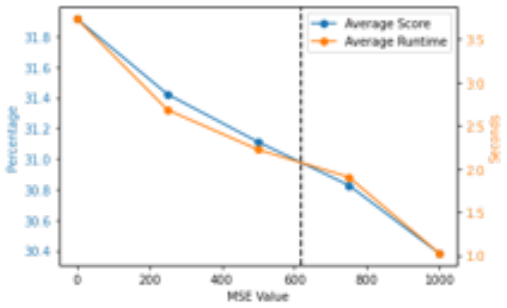


(a) Relative Performance vs. Relative Runtime per Frame.

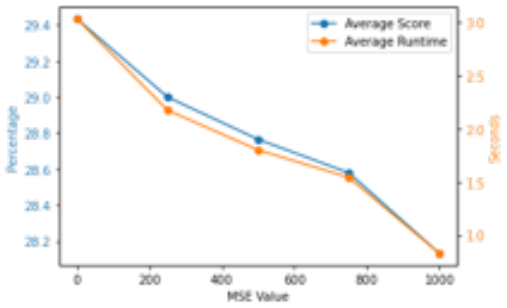


(b) Relative Performance per Individual Model and MSE.

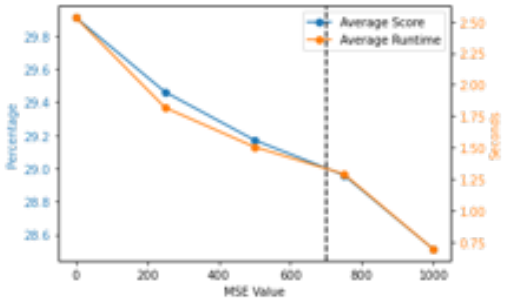
Figure 5.6: Relative Performance and Relative Runtime



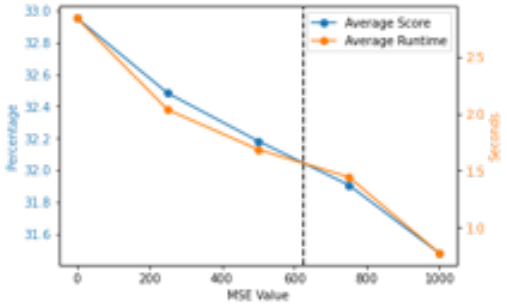
(a) Multilabel Classification



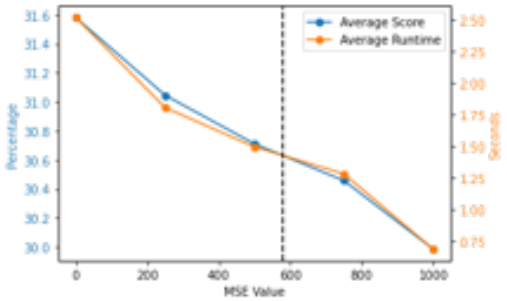
(b) Image Segmentation



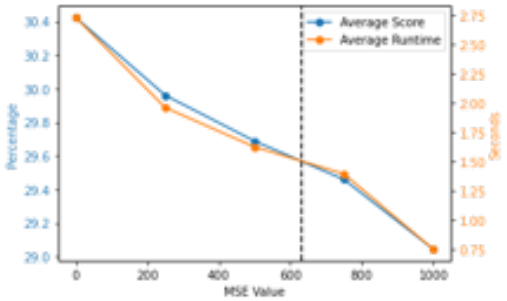
(c) Object Detection



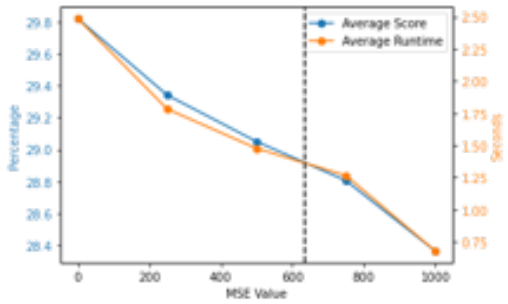
(d) Image Captioning



(e) Scene Recognition



(f) Text Recognition



(g) Night Recognition

Figure 5.7: Absolute Performance versus Absolute Runtime per Frame for each Individual Model.

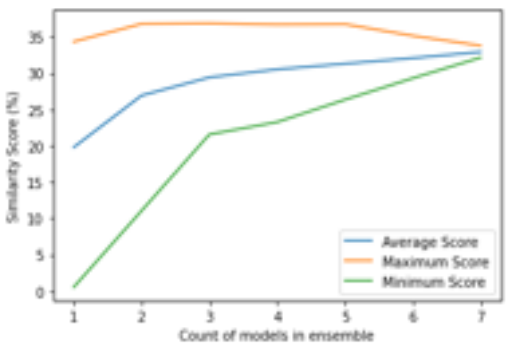
Moving on from the relative performance, Figure 5.7 shows a collection of line plots. Each line plot displays the absolute performance against the absolute runtime per frame for one of the methods employed in this thesis. It can be derived that for six out of the seven individual models (multilabel classification, image captioning, object detection, scene recognition, text recognition and night recognition), performance and time cross each other at an MSE value of approximately 640. In the case of image segmentation, performance and time do not cross.

5.3 Ensemble Size

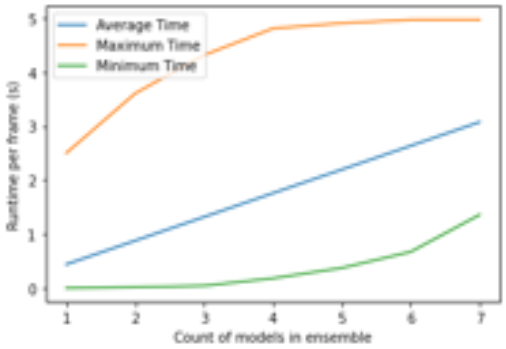
In addition to the previous analyses, further investigation was conducted to gain a deeper understanding of the impact of ensemble size on performance. For each method employed in this thesis an upset plot was created displaying all ensembles of which a specific method belonged to, leading to seven different upset plots (Appendix E). Then, the plots were sorted on ensemble size, with the smallest size on the most left. After sorting, a trend can be observed across the different plots: in general, the performance of an ensemble increases the bigger the ensemble size. However, when the ensemble reaches or goes beyond a size of three or four, the performance stays the same, increases only marginally, or even decreases.

Figure 5.8 displays these observed trends in a clearer way, providing a more detailed understanding of the impact of ensemble size on performance. It presents the minimum, maximum, and average performance across all scenes by ensemble size. The analysis of this figure reveals that the maximum score changes only marginally after an ensemble size of two. Once the size of the ensemble exceeds five, the maximum performance even decreases. Additionally, Figure 5.8a shows that the biggest improvement in minimum performance occurs between an ensemble size of one and three.

Figure 5.8b displays the minimum, maximum, and average runtime per frame by ensemble size. The figure displays that the line of average runtime per frame experiences a constant increase for each model added to the ensemble. This implies that the runtime is affected heavily by the number of models in an ensemble.



(a) MAX, AVG, and MIN Performance by Ensemble Size.



(b) MAX, AVG, and MIN runtime per Frame by Ensemble Size.

Figure 5.8: Performance and Runtime by Ensemble Size.

5.4 Examples and Scene Specific Analysis

In the following paragraphs the ten scenes with the highest achieved similarity score are described in order to give the reader a better image of how the models and ensembles work, and what their output looks like. Figure 5.9 displays a screenshot from each scene in the top 10, and Table 5.1 shows the generated metadata by the best performing ensemble, the Ground Truth, and their similarity score.

An initial observation of Table 5.1 is that the scene recognition model is included in nine out of ten ensembles achieving the top 10 scores. Additionally, it can be observed that text recognition and night recognition are not part of any ensemble present in the Top 10.

Starting off with the scene that achieved the highest overall result (Figure 5.9a), it can be derived from Table 5.1 that for this scene a similarity score of 78.45% was achieved by an ensemble including the methods image segmentation, object detection and scene recognition. What is striking about the ensembles achieving this similarity score for this scene, is that all include duplicate removal. The same ensemble without duplicate removal does not reach the same similarity score.

The scene with the second highest similarity score is presented in Figure 5.9b. Table 5.1 shows that its prediction was made by an ensemble including object detection and scene recognition without any duplicate removal, and the ensemble achieved a similarity score of 78.35%. Interesting in this scene is that in contrast to the best performing scene, the highest similarity score is in this case only achieved by the ensemble in the setting of no duplicate removal, and is not achieved by the same ensemble in the setting of duplicate removal.

Ranking	Ground Truth	Prediction	Similarity Score	Model
1	Near skull of dinosaur on display	natural history museum, wall, skull, dinosaur, footwear, poster	**78.45%**	SR_ID_OD_250 SR_ID_OD_500 SR_ID_OD_750 SR_ID_OD_1000
2	Ext. opt. swan on the sea	Ext. opt. iceberg, swan, bird, boat	**78.35%**	SR_OD_0
3	Int. opt. metro train passengers sit in metro trains	Int. opt. train interior, people, subway train, bus	**78.16%**	SR_IC_0 SR_IC_250 SR_IC_500 SR_IC_750 SR_IC_1000
4	Ext. opt. small boats are moored, the harbor basin can be seen, houses can be seen by the harbour	Ext. opt., boats, harbor	**77.98%**	SR_IC_0 SR_IC_250 SR_IC_500 SR_IC_750 SR_IC_1000
5	The horse-drawn carriage starts and enters the picture from the right. Walking up the cobbled street, around a street corner marked by a large crooked tree, and past a large yellow half-timbered house	horse drawn carriage, front, building, carriage, street, horse	**77.54%**	IC_1000
6	Near skull of dinosaur on display	natural history museum, skull, dinosaur, poster, footwear	**77.08%**	SR_OD_0
7	Ext. opt. swan on the sea	Ext. opt. iceberg, water, swan, bird, boat	**76.95%**	SR_IS_OD_0
8	Ext. opt. swan on the sea	Ext. opt. iceberg, water, swan's down, tundra swan, whistling swan, Bewick's swan	**76.90%**	SR_ID_MC_0
9	Natopt. windmill	Natopt., windmill, wind turbine	**76.79%**	NR_MC_0 NR_MC_250 NR_MC_500
10	Near skull of dinosaur on display	natural history museum, wall, skull, dinosaur, footwear, poster, allosaur, tyrannosaur, tooth, ostracoderm	**76.75%**	SR_IS_OD_MC_250 SR_IS_OD_MC_500 SR_IS_OD_MC_750 SR_IS_OD_MC_1000

Table 5.1: Top 10 Scenes with Highest Similarity Scores

(a) Top 1, 6, 10

(b) Top 2, 7, 8

(c) Top 3

(d) Top 4

(e) Top 5

(f) Top 9

Figure 5.9: Screenshots from Top 10 Performing Scenes in Dataset.

The third best described scene is presented in Figure 5.9c with a similarity score of 78.16%. The ensembles yielding this score include the image captioning and scene recognition models. Striking is that in this specific case, duplicate removal does not seem to have any effect on the prediction of the ensemble, since every ensemble generates the same prediction for each MSE value, and thus the same similarity score. The same holds for the next best performing scene depicted in Figure 5.9d, a scene for which the best performing ensembles also include image captioning and scene recognition, and where the ensembles achieve a similarity score of 77.98% at all different MSE levels.

The fifth top-described scene is presented in Figure 5.9e. Looking at Table 5.1, it can be derived that the best performing ensemble for this scene is the individual model of image captioning with the strongest degree of duplicate removal (MSE=1000). In other words, the model with the fewest frames outperforms all other versions of the model.

The next interesting finding is that the next three best-described scenes, as well as the last best-described scene, are all scenes that already appeared in the Top 10, but then at a higher ranking. This concerns the scenes in Figure 5.9a and Figure 5.9b. The reason why the scenes appear in the Top 10 more than once is that there are multiple different ensembles good at predicting this scene.

For the scene in Figure 5.9a, Table 5.1 shows that in two out of the three cases, the ensembles with duplicate removal outperform the ensembles without duplicate removal. However, for the case of the ensemble with scene recognition and object detection, duplicate removal seems to decrease the similarity score of the prediction.

The last scene that should be discussed is depicted in Figure 5.9f, for which a similarity score of 76.79% is achieved. For this scene multilabel classification combined with night recognition are the best performing ensembles. It makes sense that this is the only scene where night recognition plays a significant role, since this is a scene which has been shot at night. An additional finding is that in the setting of duplicate removal, the ensemble can still perform well until MSE=500, after which the ensemble seems to suffer from the removal of more duplicates.

Chapter 6

Discussion

Now that the results have been presented, they will be thematically categorized and discussed in the following chapter. For this purpose, the discussion is divided as follows: first, the five sub questions of the thesis are answered by referring to the project results. Thereafter, all results are combined to give an answer to the research question. Finally, academic and business implications are presented as well as recommendations specifically for TelevisionCo.

6.1 Sub Questions Answering

6.1.1 Sub Question 1

Which AI models can be used to automatically generate descriptive metadata for videos?

To address the first sub question of this thesis, the findings from the literature review (2.3) are revisited. Video frames need to be extracted and described for video metadata generation. Five primary computer vision tasks are performed for this purpose: multilabel classification, object detection, image segmentation, image captioning, and optical character recognition.

In this project, different AI models were utilized to perform these tasks along with two additional models. Figure 5.1a reveals that the individual image captioning (IC) model outperformed all other individual models with an average similarity score of 34.33%. The IC model (*vit-gpt2-image-captioning*) demonstrated a strong ability to contextualize. Unlike other models that only identified objects in the scenes, the IC model provided more specific and detailed descriptions, such as recognizing a "little girl" (compared to only "person"). This comprehensive understanding is reflected in its superior performance.

The lower performance of the individual night recognition (NR), scene recognition (SR), and text recognition (TR) models was anticipated, as these models were designed to enhance the understanding of specific attributes rather than to analyze frames holistically. Despite being penalized for empty results in the evaluation, the SR model performed relatively well, highlighting its value for TelevisionCo. In contrast, the TR model often generated excessive noise due to its inability to extract relevant information from the detected text (even after applying the NER algorithm), achieving low similarity scores. Lastly, the reason for the low performance

of the NR model is that it simply generates output in such few cases, that the amount of zeroes dominates in the final average performance.

In conclusion, all applied AI models can be used to automatically generate descriptive metadata for videos. However, they differ in their effectiveness.

6.1.2 Sub Question 2

Can an ensemble of AI models outperform individual AI models and produce synergies?

The second sub question can be answered by examining Figure 5.3. The figure demonstrates that ensembles generally outperform individual models. Notably, the best "SR-NR-IC" ensemble achieves an average similarity score of 36.85%, representing a 7.34% improvement over the individual image captioning model. This shows that the scene and night recognition methods, originally intended as supplementary techniques, indeed bring added value.

It is worth noting that an average similarity score of 36.85% across all scenes is a respectable achievement. Given that the analyzed videos encompass a diverse array of topics, an overall performance approaching 37.00% signifies a high degree of generalization in the models, demonstrating their adaptability to various subjects. Furthermore, as highlighted in Chapter 2, research indicates that automatically generated metadata does not need to be flawless to be valuable (Witbrock & Hauptmann, 1997). This notion was corroborated by TelevisionCo during the initial brainstorming session, where they expressed that even a model capable of providing a preliminary understanding of a video's content would significantly streamline their work and reduce the time required for metadata generation (Appendix A). Therefore, the achieved results can be considered a success.

The results show, however, that ensembles combining different frame description models, do not exhibit substantial improvement. This may be attributed to the absence of complementary results. Even if all models are trained on different datasets and apply different methods, their extracted information does not seem to complement each other. This implies that the tasks performed by the individual frame describing models are too similar to complement each other.

As stated in Section 5.1, no evident synergistic effects were observed: none of the 120 ensembles achieved a higher average similarity score than the sum of their individual model scores. This could be partially attributed to the fact that merging extracted information does not produce additional insights.

In summary, ensembles of AI models outperform individual models. However, synergies were not detected when comparing ensemble scores with the sum of their individual scores using the hard-voting ensemble approach.

6.1.3 Sub Question 3

To what extent does a trade-off exist between performance and runtime for the individual models and ensembles utilized?

In this study, the multilabel classification (MC) model was found to be the slowest of all models, taking 2.501s per frame. One possible explanation for this is that the model was trained on a large number of labels, precisely 21,841, and therefore

requires more time to assess a frame against all these labels. Conversely, the object detection (OD) model, trained on only 600 labels, is considerably faster, taking 0.096s per frame. However, the OD model ranks only one place behind the MC model in terms of similarity scores. It is possible that the additional details provided by the MC model do not significantly enhance the description's performance. For example, while the OD model detects a swan in an image (see 5.1, Top 2), the MC model attempts to identify the swan's species (5.1, Top 8). Often, the evaluation model penalizes rather than rewards this extra information.

Furthermore, the image captioning (IC) model's speed of 0.709s per frame, much faster than the second-best performing MC model (2.501s per frame), could be attributed to the runtime advantage of Vision Transformers over CNNs. This advantage is due to their self-attention attribute (see Chapter 3.4).

The scene recognition (SR) and night recognition (NR) models, both faster than any other model, likely achieve their speed due to their specialization, which requires examining fewer frame properties compared to the frame description models. The NR model is a unique case, as it is not a pretrained machine learning model that processes and analyzes images, but rather only performs mathematical calculations of pixel values.

Interestingly, the three fastest models (IC, SR, NR) are exclusively found in the two highest-performing yet notably quick ensembles (0.773s and 0.771s, respectively). As demonstrated in Figure 5.4, the subsequent two ensembles achieve similar similarity scores but take five times longer (3.768s and 3.767s). Thus, for ensembles as well, faster runtime does not necessarily imply inferior results.

These findings suggest that there is no evident speed/performance trade-off when employing AI models for video description in this project.

6.1.4 Sub Question 4

Is it necessary to analyze every frame within a scene, or can the examination of a reduced set of extracted frames also provide sufficient information?

As discussed in Section 4.4.1, key frames are commonly analyzed for video scene description instead of several frames per scene. Nevertheless, this project adopted a per-second frame analysis approach to accommodate TelevisionCo's distinct requirements. Furthermore, the effects of different levels of duplicate removal on runtime and performance were explored. The results depicted in Figures 5.6 and 5.7 will be interpreted in the following paragraphs.

Figure 5.6a shows that the (relative) impact of duplicate removal on the models' runtime is substantial. It must be clarified that duplicate removal does not make the individual models run faster, it merely leads to decreased overall runtime since fewer frames per scene are being analysed. The impact of duplicate removal on the (relative) performance of the models however, is limited. This initial observation suggests that analyzing more frames does not necessarily yield additional information for the data in this thesis. Upon closer examination of Figure 5.6b, it becomes apparent that the impact varies among the models. The two outliers, text recognition and night recognition, warrant particular attention.

The performance decline of the night recognition model could be attributed to the reduced number of considered frames, which significantly affects the average

pixel values. Although unique frames may still identify the same objects or scenes, more granular information about their average pixel value is lost during the duplicate removal process. This seems to have a great impact on the NR model's performance.

A less pronounced outlier observed in Figure 5.6b is the text recognition (TR) model. As this model heavily relies on the presence of text within a scene, it is hypothesized that duplicate removals led to the elimination of similar frames containing text. Additionally, since neither the frame extraction nor the duplicate removal process took frame quality into account, it is possible that the discarded duplicates were of higher quality than the remaining unique frame. If the extracted unique frame is relatively blurred, it may not hinder frame description models from detecting objects, but it could have a more significant impact on text recognition, as these models perform better when the text is depicted more sharply.

Figure 5.7 suggests that, for nearly all models, there is a point where the absolute impact of duplicate removals on quality surpasses its impact on runtime. From these results, it can be inferred that a duplicate removal with a mean squared error (MSE) threshold of approximately 640 might represent the optimal balance between the two extremes (using every frame per second on one side and considering key frames only on the other). Implementing a higher threshold results in an (absolute) performance loss that cannot be compensated by the runtime gain, while using lower thresholds might reveal more information at the expense of higher runtime.

Considering all these observations, it can be summarized that using a frame-per-second basis for scene description yields the best performance. Nevertheless, utilizing fewer frames in the process does not severely impact performance. A turning point could be a duplicate removal with an MSE threshold of about 640, implying that the time gained from removing additional frames does not justify the subsequent performance reduction.

6.1.5 Sub Question 5

What potential business value might a pipeline of AI models, designed to create metadata for digital video archives, have for a broadcasting company?

In order to address the final sub question, it is essential to first discuss how the business value of this project can be determined. The researchers opted to approximate the business value using the following formula:

$$V_m = \min\{C_{TV} \times Q_M \times \frac{T_{TV}}{T_M}; C_{TV}\}$$

The rationale behind this function is as follows: TelevisionCo associates specific costs C_{TV} with accurately describing its entire video archive within an appropriate time frame. Consequently, the value of a model that generates descriptions equivalent to TelevisionCo's current approach would equal these costs C_{TV}. However, a model's performance can be influenced by two distinct dimensions: quality Q_M and time T_M.

The concept of quality's impact is relatively intuitive and can be illustrated by three examples: (1) The best performing model (with a quality Q_M of 100%) would describe scenes precisely as TelevisionCo expects. In this case, the model's predictions fulfill all expectations, enabling it to take over the entire process of

describing videos, and its value would equal C_{TV}. (2) The worst performing model (with a quality Q_M of 0%) would not produce any valuable descriptions, requiring TelevisionCo to continue manual video annotation. In this scenario, the model would be valueless, or in other words, its value amounts to zero. (3) It is assumed that in cases where a model's quality falls between 0% and 100%, TelevisionCo employees must bridge the quality gap by editing or supplementing missing content themselves. For instance, if a model achieves results with 70.00% similarity to the ideal, the remaining 30.00% must still be manually produced (or updated). As a result, 30.00% of costs are still incurred by TelevisionCo, and the model's value would equate to the remaining 70.00% only. In summary, the quality Q_M directly affects the value of the model. This variable is determined by the average similarity score of a model or ensemble. As TelevisionCo's descriptions served as Ground Truth in this study, the similarity score represents the company-specific quality of the proposed model or ensemble.

Addressing the influence of time is slightly more complex. However, the underlying assumption is similar: the costs TelevisionCo associates with video metadata creation also correspond to certain temporal expectations. For instance, if TelevisionCo aimed to accelerate the manual video description process, it would need to employ more personnel for this task. Consequently, reducing the required time by half would double the costs. The same principle applies to a proposed model. If the model requires half the time of the original procedure, this would double the value of that specific model. Hence, two more variables are incorporated into the formula in the following: T_{TV}/T_M. T_{TV} represents the time variable of TelevisionCo and T_M denotes the time variable of the model. If the model needs half the time, the fraction equates to 2. These variables themselves are comprised out of the respective speed variables (S_{TV} and S_M) and their utility factors (U_{TV} and U_M). This will be elaborated on in the later paragraphs.

Bringing everything together, a model's or ensemble's business value can be determined by multiplying the existing costs C_{TV} with the model's quality Q_M and its time coefficient T_{TV}/T_M. However, the business value of a proposed model can never surpass the costs TelevisionCo currently allocates to this task. Thus, the formula was adjusted to produce the minimum of the current attributed costs C_{TV} and the otherwise calculated value $C_{TV} \times Q_M \times \frac{T_{TV}}{T_M}$.

As outlined in the introduction (Chapter 1), TelevisionCo revealed that its archive contains roughly 460,229 hours of video content. Employing the current manual approach, they manage to annotate 1,718 hours of video material per year, with an associated cost of 360,000 DKK. Assuming TelevisionCo aims to annotate the entire archive using this method, the total expense would amount to 96,439,139 DKK. Thus, C_{TV} = 96,439,139 DKK.

The best model's average similarity score represents Q_M, equating to 0.38. The time variables present a more complex situation. The model's speed variable, S_M, is 0.7730, as it takes 0.773 seconds to annotate a single frame (or 0.773 hours for one hour, and so forth). In contrast, TelevisionCo's speed can be determined by the time required to annotate a specific volume of videos. Based on their data, they need 910 hours to annotate 1,718 hours of material. Consequently, S_{TV} equals 0.5297. However, comparing these values would fail to consider that the proposed model can operate 365 days a year, whereas humans do not work every day. Neglecting this aspect would undermine the model's ability to annotate the entire archive seven

times faster than TelevisionCo.

To account for this discrepancy, the speed variables must be adjusted by a utility factor $U = \frac{T_{usable}}{T_{available}}$. For TelevisionCo, this means that on an annual basis, they have 8,766 hours available (365.25 * 24) but only utilize 910 of them, resulting in $U_{TV} = \frac{910h}{8{,}766h} = 10.38\%$. With $T_{TV} = \frac{S_{TV}}{U_{TV}}$, TelevisionCo's time variable ultimately amounts to 5.1031. Since the proposed model can theoretically utilize every hour of a year, its utilization variable U_M is 100% ($\frac{8{,}766h}{8{,}766h}$). As a result, T_M equates to S_M, which is 0.7730.

Given all of the above, the overall value of the model can now be computed as follows:

$$V_m = \min\{96,439,139DKK \times 0.38 \times \frac{5.1031}{0.7730}; 96,439,139DKK\}$$

$$= \min\{241,930,992DKK; 96,439,139DKK\}$$

$$= 96,439,139DKK$$

Hence, it can be argued that the proposed solution delivers a business value large enough to outweigh all costs linked to video metadata creation at TelevisionCo. This reemphasizes the positive impact this project has on TelevisionCo and could have on similar companies in the industry.

However, it must be acknowledged that this calculation is an approximation. Furthermore, certain relevant cost and time factors associated with model implementation, such as computing and maintenance costs and pre- and postprocessing times, were not included for simplicity. On the other hand, this calculation does not take into account that TelevisionCo obtains 22 times more video material per year than they can annotate, suggesting that continuing their current process would also escalate costs over time.

Besides the monetary value, the proposed solution of this thesis offers TelevisionCo a scalable solution. One of the primary benefits of using machine learning models for metadata generation is their scalability. When using a manual appraoch to create metadata, the process is limited by the number of employees available and the amount of time it takes them to complete the task. For example, if a digital video archive has a large number of videos that require metadata, it would take a significant amount of time and resources to manually create metadata for each one. In contrast, an automated pipeline of machine learning models can easily handle archives of any size. As the size of the archive grows, more computational resources can simply be allocated to the task, which is not as easily done with manual work.

6.2 Research Question Answering

To what extent can AI play a role in the process of generating metadata for digital video archives in broadcasting corporations?

In conclusion, taking into account all findings discussed, the research question can be addressed as follows. AI can significantly contribute to the metadata generation process for digital video archives in broadcasting corporations by providing a scalable, cost-effective, and efficient solution. The ensemble of AI models employed

in this thesis demonstrated the potential to outperform individual models, offering more accurate and consistent metadata generation. By leveraging open-source resources and customizing AI models to the specific needs of broadcasting corporations, the use of AI can lead to considerable cost reductions, resource reallocations, and greater overall profitability. However, it is important to acknowledge the need for ongoing collaboration between different departments within the organization in order to successfully implement AI.

6.3 Academic Implications

From the outcomes of the research conducted in this thesis, three implications for researchers can be deduced.

6.3.1 Optimal Duplicate Removal Threshold

Section 5.2 demonstrates that for six of the seven models employed in this thesis, there is a turning point at which the performance decline due to duplicate removal can no longer be reasonably compensated by runtime improvements. The observation that this intersection point lies roughly at the same mean squared error value (MSE) in each figure implies that an optimal value for the MSE can be determined.

This research has shown that the crossover point corresponds to an MSE value of approximately 640. Future researchers investigating a similar issue or task can use this insight as a basis for their research. This approach will save time, as they will not need to determine whether duplicate removal is necessary nor spend time deriving the optimal MSE value for the most efficient duplicate removal process.

6.3.2 Optimal Ensemble Size & Approach

Section 5.2 showcases figures that imply an optimal ensemble size for this particular approach to ensemble learning (hard voting). The observed pattern suggests that incorporating more than three distinct models into an ensemble does not markedly enhance the average performance, while the runtime per frame substantially increases. Given the importance of both time and performance, this thesis proposes an ideal ensemble size of three.

Furthermore, an academic implication of this study concerns the importance of selecting and adapting ensemble learning methods to better suit the specific task at hand. The hard-voting method employed in this thesis did not generate the expected synergies that ensemble learning is known for, even though it outperformed individual models.

6.3.3 Custom Models and Task-Specific Training

This thesis also highlights the academic implication of the benefits of developing and training custom models tailored to a specific task, rather than relying on pretrained models. The pretrained models used in this study struggled to generate metadata that met TelevisionCo's specific requirements, such as determining when to provide detailed versus more high-level descriptions.

This finding underscores the importance of researchers investing in the development and training of custom models that can address the nuances of the metadata generation tasks more effectively, thereby advancing the understanding and capabilities of AI models in the context of domain-specific metadata generation.

6.4 Business Implications

Moving beyond the academic implications, this thesis also identifies three business implications for multimedia corporations.

6.4.1 Adoption of Artificial Intelligence

The ultimate implication of this thesis is the demonstrated potential of incorporating AI models into businesses to tackle the escalating demand for metadata generation in multimedia corporations. With the constant growth of digital video archives, scalable solutions are needed to accommodate the increasing demand for metadata. This thesis showcases the potential for transforming the metadata generation process by employing AI models, minimizing reliance on manual labor, and allowing businesses to adjust to the expanding scale of their digital video archives. In a recent meeting, TelevisionCo indicated plans to implement the proposed pipeline as an additional feature in their systems, reflecting the potential they recognize in the conducted research and its business impact.

6.4.2 Proven Potential of Open-Source Technologies

The second implication arising from this thesis is the proven potential of utilizing open-source solutions in developing AI-powered metadata generation tools. The models, libraries, and architectures used in this thesis were all accessible without cost, demonstrating the capability of open-source resources generating metadata for multimedia corporations. This discovery highlights the importance of exploiting these readily available resources to foster innovation and devise effective solutions tailored to the specific difficulties encountered by businesses in the multimedia sector.

6.4.3 Cost Savings and Resource Reallocation

As illustrated in Section 6.1.5, the business value of the recommended solution is approximately 96,439,139 DKK. By implementing AI models for metadata generation, multimedia businesses like TelevisionCo can substantially decrease the costs associated with manual metadata production. By saving these costs, companies can become able to reallocate saved resources to other value-added tasks or expand the company's operations. This cost reduction can contribute to a stronger market presence and higher overall profitability.

6.5 Recommendations for TelevisionCo

This section will discuss the authors' recommendations for TelevisionCo based on the findings of this thesis. The initial recommendation is to begin incorporating

AI into their metadata creation process. Familiarizing themselves with artificial intelligence within their business operations is crucial, and this presents an ideal opportunity to demonstrate the value of AI to other parts of the company.

Implementation of Proposed Pipeline

From a technical standpoint, the optimal ensemble for generating metadata consists of the scene recognition, night recognition, and image captioning models. However, when presenting this ensemble's results and predictions to TelevisionCo, their primary concern was the omission of text recognition. After discussing, TelevisionCo was willing to accept a lower performance to ensure the inclusion of text recognition. Consequently, from a business perspective, it is advisable to employ an ensemble comprising the scene recognition, night recognition, image captioning, and text recognition models.

Implementation Plan

Upon deciding to implement the proposed pipeline from this thesis, it is vital for TelevisionCo to develop a comprehensive implementation plan. The authors have primarily interacted with the metadata department of TelevisionCo. However, when aiming to implement the proposed solution, it is essential to involve the company's IT department to determine the most effective implementation strategy. The authors lack knowledge about TelevisionCo's specific technical aspects, such as data structures, data streams, and data storage methods. Nonetheless, it is advisable to integrate a new data pipeline into their existing local or cloud-based infrastructure. Clear communication between departments about their requirements and desired formats is crucial before implementing any solution.

Development of Metadata Framework

A further recommendation relates to the standardization of manual metadata generation and correction following the implementation of the suggested pipeline. The Ground Truth for this thesis was produced by three employees. Engaging multiple individuals in a semantically sensitive task may lead to variations in model evaluations depending on the Ground Truth creator. To ensure consistency in all future annotations and potential re-annotations of existing archive scenes, it is recommended that there should be a clear framework with explicit rules instructing employees how to correct the metadata, limiting the room for personal touch. This way, the task can easily be done by multiple employees and easily be taken over by other employees over the years, while ensuring consistency.

Training of Models

The concluding recommendation for TelevisionCo, which was also discussed in Section 6.3.3, entails examining the potential enhancement of metadata quality by training custom models. This should be treated as a distinct project running parallel to the current solution. It is also important to assign a dedicated team to this project and acknowledge that it may demand a considerable time investment.

Chapter 7

Limitations & Further Research

In this chapter, the limitations of the study are discussed, providing insights into potential areas for improvement and addressing challenges faced during the research. Additionally, by exploring further research avenues, the groundwork is laid for the development of more sophisticated methods and models in the field.

7.1 Limitations

This section investigates the limitations encountered in this study, focusing on model and data limitations, evaluation metric limitations, and ensemble method limitations. By acknowledging these limitations, a more comprehensive understanding of the research outcomes can be achieved, and guidance for future work in the field can be provided.

7.1.1 Model and Data Limitations

One major limitation of this research is the loss of information during the translation of descriptions from Danish to English. Although the AI models were trained on English data, the original metadata provided by TelevisionCo was in Danish. As a result, the translation process may have led to a loss of context or nuance, which could impact the evaluation score of the models. This is especially the case for company-specific abbreviations and terms.

Additionally, the models are unable to predict specific abbreviations and terms that are unique to TelevisionCo's domain. This limitation suggests that fine-tuning or training a machine learning model specifically for TelevisionCo's context could lead to better results.

Potential overfitting is another limitation of this study. Some threshold determinations were based on the entire dataset, which could lead to an overfitting issue. A potential mitigation of this limitation is to determine thresholds by, for example, applying cross validation methods.

Finally, the generalizability of the results from this thesis is rather limited due to the fact that the models are optimized for TelevisionCo's specific requirements. While the approach might work well for the Danish TV company, it may not necessarily perform as well in other contexts or for other organizations with different metadata needs.

7.1.2 Evaluation Metric Limitations

In terms of the evaluation metric, it is a limitation that the performance of the individual models and of the ensembles is merely based on its similarity score with the Ground Truth. For example, it was observed that in certain cases, the metadata generated by the individual models and the ensembles could arguably be considered superior to the Ground Truth. However, since the evaluation metric focuses solely on comparing to the Ground Truth, it cannot represent this superiority. Instead, it often even penalized the model for describing more than the Ground Truth, leading to a lower similarity score.

An example can be found in Figure 7.1, where the Ground Truth described "Ext. opt. lifeguard seen from the back", while the best-performing ensemble predicted "Ext. opt., water, beach, yellow shirt, woman, man, surfboard". Although the model failed to identify the woman as a lifeguard, it offered a more detailed description of the scene, including the presence of both the woman and the man, the surfboard, the beach setting, water, and even the woman's yellow shirt. However, as TelevisionCo's descriptions serve as the Ground Truth, the similarity score for this specific scene was (merely) 36.25%.

Figure 7.1: Screenshot of Scene where AI Ensemble Potentially Outperforms Ground Truth.

Another limitation is the sensitivity of the chosen cosine similarity metric to word order. Although unique model combinations were tested, it is possible that ensembles with different ordering could achieve varying scores or even higher results.

Additionally, the decision to set similarity scores to zero when no prediction was made penalizes models like the night and scene recognition models, which were designed to occasionally produce no output. Although this decision may not have a substantial impact on the overall project, as these models were considered supplementary, it is worth noting that the performance of these models alone might be better than the results suggest.

7.1.3 Ensemble Method Limitations

With regards to the ensemble method, only unique combinations of models were considered. The evaluation metric's sensitivity to word order implies that ensembles arranged in different orders could produce different scores or even higher results. Furthermore, the ensemble method employed in this study, hard voting, combined models without taking into account their individual strengths and weaknesses.

7.2 Further Research

In this section, potential research opportunities are identified, including data and model improvements, ensemble method refinements, efficiency and performance enhancements, and an extensive evaluation and generalization. Following these directions can help mitigate the identified limitations and advance the field of automatic metadata generation for video archives.

7.2.1 Dataset and Model Improvements

During the analysis of various company-specific abbreviations and terms, it became evident that many of these terms were scarcely (or never) present in the existing dataset. As a result, one promising direction for future research involves broadening the dataset used for analysis. By adding a more considerable number of videos, a more in-depth understanding of company-specific terms and context can be attained. This would likely improve the AI models' performance and increase the precision and relevance of metadata generation for TelevisionCo's domain.

Another direction for improvement is to train or fine-tune AI models on TelevisionCo's extensive video archive. By developing and training own models, these models can become more specialized and attuned to the specific requirements and nuances of TelevisionCo's metadata, potentially leading to better performance and more accurate metadata generation.

Lastly, utilizing more sophisticated approaches for determining the optimal thresholds is another area worth exploring. This could address possible overfitting concerns and help finding the most effective parameters for each model.

7.2.2 Ensemble Method Refinements

Continuing research on the improvement of ensemble methods could encompass investigating the effects of different orders, or permutations, within ensembles, rather than merely assessing unique model combinations. As the similarity score is sensitive to word orders, this approach might reveal new optimal configurations that lead to superior performance and results.

Another promising direction for advancing ensemble methods is the implementation of weighted ensembles. By attributing different weights to individual models within an ensemble (e.g., a 40.00% contribution from one model and 60.00% from another), it is feasible to more effectively capitalize on the unique competencies of each model.

7.2.3 Efficiency and Performance Enhancements

One potential for further research in respect to enhancing efficiency and performance lies in the utilization of key frames. It was shown that using less frames does not considerably impact the performance of a model, but it does impact its speed. By concentrating on key frames, it might be possible to maintain a balance between performance impact and time efficiency. This approach could streamline the process of generating metadata, without substantially sacrificing quality.

Another interesting area for investigation is the advantages of specific models in relation to different delimitable aspects. By identifying and understanding the strengths and weaknesses of individual models in specific contexts or topics, it may be possible to better tailor their application to TelevisionCo's metadata generation requirements, ultimately improving overall performance.

7.2.4 Extensive Evaluation and Generalization

As previously mentioned, one significant limitation of this thesis may be the restricted generalizability of its findings. Consequently, future research could involve evaluating the approach's efficacy on an independent dataset. Investigating the models' adaptability to various contexts or organizations with distinct metadata needs would establish the flexibility of the ensemble method and AI models in different scenarios and assess their appropriateness for a range of settings.

Lastly, exploring alternative evaluation metrics is imperative for a more all-encompassing understanding of model performance. This examination would reveal deeper insights into the ensemble method and AI models' strengths and weaknesses, ultimately contributing to the advancement of more refined metadata generation techniques.

An overview of the five most important limitations of this thesis with their respective implications on future research can be seen in Table 7.1 .

Limitations of Thesis	**Implications for Further Research**
Company-specific abbreviations and terms poorly depicted	Training or fine-tuning on TelevisionCo's data
Potential overfitting of thresholds	Analysis of larger sample of videos
Limited generalizability	Validation on independent dataset
Evaluation metric's sensitivity to word order	Investigation of alternative evaluation metrics
	Test influence of different orders in ensembles (permutations)
Ensembles not flexibly leveraging individual strengths	Implement weighted ensembles

Table 7.1: Overview of Five Most Important Limitations & Further Research Implications.

Chapter 8

Conclusion

This project set out to investigate the role of AI in the process of generating metadata for digital video archives in broadcasting corporations, using the business case of TelevisionCo as an example. The primary academic goal of this thesis was to explore the potential of AI in the field of metadata for digital video archives, and specifically, whether an ensemble of AI models can outperform individual models in the quality of the metadata. The primary business goal of this thesis was to evaluate how such AI models can create business value for broadcasting corporations such as TelevisionCo and from there derive implications for businesses alike.

Following these goals, this thesis demonstrated that AI can, and should, play a considerable role in the process of generating metadata for digital video archives. The results of this thesis reveal that the quality of the metadata generated by ensembles of AI models is not yet at a perfect state but is still decent enough for companies to want to use it. Moreover, this thesis shows that generating metadata with AI models is more time-efficient than when creating it manually, and it offers the potential to scale, which is of high importance with the ever growing multimedia content.

Throughout the research, a thorough analysis of the performance of both the seven employed individual models and the ensembles was conducted. The findings revealed that the best performing ensemble, when it comes to the similarity score between the artificially generated metadata and the Ground Truth, consisted of the models scene recognition, night recognition, and image captioning. This ensemble yielded an average similarity score of 36.85%.

Additionally, this study highlights the significance of utilizing readily accessible resources such as the open-source models employed in the research. Companies should seize the opportunity to leverage these models, as they offer valuable benefits at minimal costs. Moreover, an important academic contribution of this thesis is that the results suggest an optimal threshold for frame reduction of MSE = 640.

Finally, the last main finding of this thesis is that the value of the proposed solution outweighs the costs TelevisionCo currently incurs from generating metadata. Following a self-developed formula, the final value of the proposed model to TelevisionCo is estimated to be 96,439,139 DKK.

Despite the promising results of this thesis, there remain limitations and solid opportunities for further research. The biggest opportunities for future work lie in experimenting with different individual models specialized in different tasks, different ensemble methods and different evaluation metrics, as well as conducting a more general study with a "one-size-fits-all" solution that is not tailored to solely

one company.

In conclusion, this thesis has demonstrated the potential of AI in transforming metadata generation processes for broadcasting corporations. By implementing an ensemble of AI models, companies like TelevisionCo can effectively address the growing need for metadata in their digital video archives while simultaneously benefiting from cost reduction, resource allocation advantages, and scaling possibilities. As the field of AI continues to evolve, further improvements in AI capabilities can be anticipated, leading to even more efficient and sophisticated metadata generation solutions for the multimedia industry.

Bibliography

Aiforia. (2021). *Instance segmentation for digital pathology tissue image analysis - what is it and why do we need it?* https://digitalpathologyplace.com/instance-segmentation-for-digital-pathology-tissue-image-analysis-what-is-it-and-why-do-we-need-it/

Asha Paul, M. K., Kavitha, J., & Jansi Rani, P. A. (2018). Key-frame extraction techniques: A review. *Recent Patents on Computer Science*, *11*(1), 3–16. https://www.ingentaconnect.com/content/ben/cseng/2018/00000011/00000001/art00004

Barla, N. (2023). *The beginner's guide to semantic segmentation.* https://www.v7labs.com/blog/semantic-segmentation-guide#:~:text=Semantic%5C%20Segmentation%5C%20follows%5C%20three%5C%20steps%5C%3A%5C%20Classifying%5C%3A%5C%20Classifying%5C%20a,a%5C%20localized%5C%20image%5C%20by%5C%20creating%5C%20a%5C%20segmentation%5C%20mask.

Chaudhuri, A., Mandaviya, K., Badelia, P., & Ghosh, S. K. (2017). Optical character recognition systems. In *Optical character recognition systems for different languages with soft computing* (pp. 9–41). Springer Verlag. https://doi.org/10.1007/978-3-319-50252-6_2

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2014). Semantic image segmentation with deep convolutional nets and fully connected crfs. http://arxiv.org/abs/1412.7062

Chen, X., Girshick, R., He, K., & Dollár, P. (2019). Tensormask: A foundation for dense object segmentation. https://openaccess.thecvf.com/content_ICCV_2019/papers/Chen_TensorMask_A_Foundation_for_Dense_Object_Segmentation_ICCV_2019_paper.pdf

Cheng, B., Schwing, A. G., & Kirillov, A. (2021). Per-pixel classification is not all you need for semantic segmentation. http://arxiv.org/abs/2107.06278

Dai, J., He, K., & Sun, J. (2016). Instance-aware semantic segmentation via multi-task network cascades. https://doi.org/10.48550/arXiv.1512.04412

Deci. (2021). *A guide to common object detection algorithms and implementations.* https://medium.com/deci-ai/a-guide-to-common-object-detection-algorithms-and-implementations-455757ac9e20#:~:text=A%5C%20Guide%5C%20to%5C%20Common%5C%20Object%5C%20Detection%5C%20Algorithms%5C%20and,...%5C%207%5C%20Get%5C%20started%5C%20using%5C%20these%5C%20models%5C%20

Devkar, R., & Shiravale, S. (2017). A survey on multi-label classification for images. *International Journal of Computer Applications*, *162*, 39–42. https://doi.org/10.5120/ijca2017913398

Dorai, C., Farrell, R., Katriel, A., Kofman, G., Li, Y., & Park, Y. (2006). Magical demonstration: System for automated metadata generation for instructional content. https://dl.acm.org/doi/pdf/10.1145/1180639.1180740

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. http://arxiv.org/abs/2010.11929

Esri. (2023). *How single-shot detector (ssd) works?* https://developers.arcgis.com/python/guide/how-ssd-works/

Explosion.ai. (2023). *Spacy models documentation: En_core_web_sm.* https://spacy.io/models/en#en_core_web_sm

Farhadi, A., Hejrati, M., Sadeghi, M. A., Young, P., Rashtchian, C., Hockenmaier, J., & Forsyth, D. (2010). Every picture tells a story: Generating sentences from images. https://www.cs.cmu.edu/~afarhadi/papers/sentence.pdf

Fernández, D., Varas, D., Espadaler, J., Masuda, I., Ferreira, J., Woodward, A., Rodríguez, D., Giró-i-Nieto, X., Riveiro, J. C., Bou, E., & Alto, P. (2017). Vits: Video tagging system from massive web multimedia collections. https://openaccess.thecvf.com/content_ICCV_2017_workshops/papers/w5/Fernandez_ViTS_Video_Tagging_ICCV_2017_paper.pdf

Fischler, M. A., & Elschlager, R. A. (1973). The representation and matching of pictorial structures representation. *IEEE Transactions on Computers*, *C-22*, 67–92. https://doi.org/10.1109/T-C.1973.223602

Géron, A. (2019). *Hands-on machine learning with scikit-learn, keras, and tensorflow: Concepts, tools, and techniques to build intelligent systems.* O'Reilly Media.

Girshick, R. (2015). Fast r-cnn. *Proceedings of the IEEE international conference on computer vision*, 1440–1448. https://openaccess.thecvf.com/content_iccv_2015/papers/Girshick_Fast_R-CNN_ICCV_2015_paper.pdf

Grover, A., & Ermon, S. (2018). Boosted generative models. www.aaai.org

He, X., & Deng, L. (2017). Deep learning for image-to-text generation: A technical overview. *IEEE Signal Processing Magazine*, *34*, 109–116. https://doi.org/10.1109/MSP.2017.2741510

Hodosh, M., Young, P., & Hockenmaier, J. (2013). Framing image description as a ranking task: Data, models and evaluation metrics. *Journal of Artificial Intelligence Research*, *47*, 853–899. https://www.jair.org/index.php/jair/article/view/10833

Hossain, M. D. Z., Sohel, F., Shiratuddin, M. F., & Laga, H. (2019). A comprehensive survey of deep learning for image captioning. *ACM Computing Surveys*, *51*. https://doi.org/10.1145/3295748

IBM. (2022). *Optical character recognition (ocr).* https://www.youtube.com/watch?v=or8AcS6y1xg

Islam, N., Islam, Z., & Noor, N. (2017). A survey on optical character recognition system. *Journal of Information Communication Technology-JICT*, *10*. https://arxiv.org/pdf/1710.05703.pdf

Jurafsky, D., & Martin, J. (2022). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition.* Pearson Prentice Hall. https://books.google.dk/books?id=fZmj5UNK8AQC

Kass, M., Witkin, A., & Terzopoulos, D. (1988). Snakes: Active contour models. *International journal of computer vision*, *1*(4), 321–331. https://membres-ljk.imag.fr/Emmanuel.Maitre/pdfs/ijcv98kass.pdf

Kiros, R., Salakhutdinov, R., & Zemel, R. (2014). Multimodal neural language models. http://proceedings.mlr.press/v32/kiros14.pdf

Kiros, R., Salakhutdinov, R., & Zemel, R. S. (2014). Unifying visual-semantic embeddings with multimodal neural language models. http://arxiv.org/abs/1411.2539

Kolesnikov, A., Beyer, L., Zhai, X., Puigcerver, J., Yung, J., Gelly, S., & Houlsby, N. (2020). Big transfer (bit): General visual representation learning. http://arxiv.org/abs/1912.11370

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. https://dl.acm.org/doi/pdf/10.1145/3065386

Kulkarni, G., Premraj, V., Ordonez, V., Dhar, S., Li, S., Choi, Y., Berg, A. C., & Berg, T. L. (2013). Baby talk: Understanding and generating simple image descriptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*, 2891–2903. https://doi.org/10.1109/TPAMI.2012.162

Lanchantin, J., Wang, T., Ordonez, V., & Qi, Y. (2021). General multi-label image classification with transformers. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 16478–16488. https://openaccess.thecvf.com/content/CVPR2021/papers/Lanchantin_General_Multi-Label_Image_Classification_With_Transformers_CVPR_2021_paper.pdf

Laven, P., & Hopper, R. (2000). Confused by metadata? *EBU Technical Review*, *284*.

Lee, Y., & Park, J. (2020). Centermask : Real-time anchor-free instance segmentation. https://openaccess.thecvf.com/content_CVPR_2020/papers/Lee_CenterMask_Real-Time_Anchor-Free_Instance_Segmentation_CVPR_2020_paper.pdf

Li, S., Kulkarni, G., Berg, T., Berg, A., & Choi, Y. (2011). Composing simple image descriptions using web-scale n-grams. *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, 220–228. https://aclanthology.org/W11-0326.pdf

Li, X., Wang, L., & Sung, E. (2004). Multilabel svm active learning for image classification. *2004 International Conference on Image Processing, 2004. ICIP '04.*, *4*, 2207–2210 Vol. 4. https://doi.org/10.1109/ICIP.2004.1421535

Liddy, E. D., Allen, E., Harwell, S., Corieri, S., Yilmazel, O., Ozgencil, N. E., Diekema, A., Mccracken, N., Silverstein, J., & Sutton, S. (2002). Automatic metadata generation evaluation. https://dl.acm.org/doi/pdf/10.1145/564376.564464

Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., & Pietikäinen, M. (2020). Deep learning for generic object detection: A survey. *International Journal of Computer Vision*, *128*, 261–318. https://doi.org/10.1007/s11263-019-01247-4

Liu, S., Qi, L., Qin, H., Shi, J., & Jia, J. (2018). Path aggregation network for instance segmentation. https://openaccess.thecvf.com/content_cvpr_2018/papers/Liu_Path_Aggregation_Network_CVPR_2018_paper.pdf

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. https://openaccess.thecvf.com/content_cvpr_2015/papers/Long_Fully_Convolutional_Networks_2015_CVPR_paper.pdf

Maratea, A., Petrosino, A., & Manzo, M. (2013). *Generation of description metadata for video files*. https://dl.acm.org/doi/pdf/10.1145/2516775.2516795

Mikolajczyk, K., & Schmid, C. (2005). A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *27*, 1615–1630. https://doi.org/10.1109/TPAMI.2005.188

Minaee, S., Boykov, Y., Porikli, F., Plaza, A., Kehtarnavaz, N., & Terzopoulos, D. (2022). Image segmentation using deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*, 3523–3542. https://doi.org/10.1109/TPAMI.2021.3059968

Najman, L., Schmitt, M., & Watershed, M. S. (1994). Of a continuous function. *Signal Processing*, *38*. https://hal.science/hal-00622129/file/lpe.pdf

Nock, R., & Nielsen, F. (2004). Statistical region merging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *26*, 1452–1458. https://doi.org/10.1109/TPAMI.2004.110

Ordonez, V., Kulkarni, G., & Berg, T. L. (2011). Im2text: Describing images using 1 million captioned photographs. https://proceedings.neurips.cc/paper/2011/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf

Osuna, E., Freund, R., & Girosit, F. (1997). Training support vector machines: An application to face detection. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=609310

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, *9*(1), 62–66. https://cw.fel.cvut.cz/b201/_media/courses/a6m33bio/otsu.pdf

PaddlePaddle. (2020). *Github - paddlepaddle/paddleocr: Awesome multilingual ocr toolkits based on paddlepaddle (practical ultra lightweight ocr system, support 80+ languages recognition, provide data annotation and synthesis tools, support training and deployment among server, mobile, embedded and iot devices)*. https://github.com/PaddlePaddle/PaddleOCR

Plath, N., Toussaint, M., & Nakajima, S. (2009). Multi-class image segmentation using conditional random fields and global classification. https://dl.acm.org/doi/pdf/10.1145/1553374.1553479

Rafferty, J., Nugent, C., Liu, J., & Chen, L. (2015). Automatic metadata generation through analysis of narration within instructional videos. *Journal of Medical Systems*, *39*. https://doi.org/10.1007/s10916-015-0295-2

Raj, B. (2019). *A simple guide to semantic segmentation*. https://rb.gy/t9nye

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. https://pjreddie.com/media/files/papers/yolo_1.pdf

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, *28*. https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf

Rosenfeld, A., & Kak, A. (1976). *Digital picture processing*. Academic Press. https://doi.org/https://doi.org/10.1016/C2009-0-21955-6

Rowley, H. A., Baluja, S., & Kanade, T. (1998). Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*, 23–38. https://doi.org/10.1109/34.655647

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4510–4520. https://openaccess.thecvf.com/content_cvpr_2018/papers/Sandler_MobileNetV2_Inverted_Residuals_CVPR_2018_paper.pdf

Shakurov, M., Korobkin, D., Fomenkov, S., & Golovanchikov, A. (2021). Software for generation of video files metadata. *Journal of Physics: Conference Series*, *2060*. https://doi.org/10.1088/1742-6596/2060/1/012029

Singh, A. (2022). *Nlpconnect/vit-gpt2-image-captioning*. https://huggingface.co/nlpconnect/vit-gpt2-image-captioning

Siva, G. (2021). *What is transformer model? how does it work?* https://rb.gy/ff1ap

Snoek, C. G. M., Sande, K. E. A. V. D., Rooij, O. D., Huurnink, B., Uijlings, J. R. R., Liempt, M. V., Bugalho, M., Trancoso, I., Yan, F., Tahir, M. A., Mikolajczyk, K., Kittler, J., Rijke, M. D., Geusebroek, J. M., Gevers, T., Worring, M., Koelma, D. C., & Smeulders, A. W. M. (2009). The mediamill trecvid 2009 semantic video search engine. https://s3.eu-central-1.amazonaws.com/eu-st01.ext.exlibrisgroup.com/44SUR_INST/storage/alma/20/55/3A/DE/68/3E/0B/FF/F4/7A/F3/01/E2/1A/CD/40/mediamill-TRECVID2009-final.pdf?response-content-type=application%5C%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20230506T085336Z&X-Amz-SignedHeaders=host&X-Amz-Expires=119&X-Amz-Credential=AKIAJN6NPMNGJALPPWAQ%5C%2F20230506%5C%2Feu-central-1%5C%2Fs3%5C%2Faws4_request&X-Amz-Signature=86636731fe1b3f44777a64856a284f4d50fdbd6be5e97681ad76f5a0a131f706

Starck, J. L., Elad, M., & Donoho, D. L. (2005). Image decomposition via the combination of sparse representations and a variational approach. *IEEE Transactions on Image Processing*, *14*, 1570–1582. https://doi.org/10.1109/TIP.2005.852206

Strehl, A., Ghosh, J., & Mooney, R. (2000). Impact of similarity measures on web-page clustering. https://www.researchgate.net/publication/2367438_Impact_of_Similarity_Measures_on_Web-page_Clustering

TensorFlowHub. (2023). *Openimages$_v$4/ssd/mobilenet_v2*. https://tfhub.dev/google/openimages_v4/ssd/mobilenet_v2/1

Thompson, V. U., Panchev, C., & Oakes, M. (2015). Performance evaluation of similarity measures on similar and dissimilar text retrieval; performance evaluation of similarity measures on similar and dissimilar text retrieval. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7526978

Tidake, V. S., & Sane, S. S. (2018). Multi-label classification: A survey. *International Journal of Engineering Technology*, *7*, 1045. https://doi.org/10.14419/ijet.v7i4.19.28284

Tsoumakas, G., & Katakis, I. (2007). Multi-label classification. *International Journal of Data Warehousing Mining*, *3*, 1–13. https://www.igi-global.com/gateway/article/full-text-pdf/1786&riu=true

Vaidya, J., Yu, H., & Jiang, X. (2008). Privacy-preserving svm classification. *Knowledge and Information Systems*, *14*, 161–178. https://link.springer.com/content/pdf/10.1007/s10115-007-0073-7.pdf

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. http://arxiv.org/abs/1706.03762

Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2017). Show and tell: Lessons learned from the 2015 mscoco image captioning challenge. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*, 652–663. https://doi.org/10.1109/TPAMI.2016.2587640

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *1*. https://doi.org/10.1109/cvpr.2001.990517

Von Platen, P. (2020). *Transformers-based encoder-decoder models*. https://www.huggingface.co/blog/encoder-decoder

Wactlar, H. D., & Christel, M. G. (2002). Digital video archives: Managing through metadata, 80–101. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=fd1a16b783ddaa201aa796a90ab73c28cc7443c3#page=84

Wactlar, H. D., Christel, M. G., Gong, Y., & Hauptmann, A. G. (1999). Lessons learned from building a terabyte digital video library. *Computer*, *32*, 66–73. https://doi.org/10.1109/2.745722

Wang, C.-Y., Bochkovskiy, A., & Liao, H.-Y. M. (2022). Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. http://arxiv.org/abs/2207.02696

Wang, J., Yang, Y., Mao, J., Huang, Z., Huang, C., & Xu, W. (2016). Cnn-rnn: A unified framework for multi-label image classification. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2285–2294. https://openaccess.thecvf.com/content_cvpr_2016/papers/Wang_CNN-RNN_A_Unified_CVPR_2016_paper.pdf

Wang, W., Dai, J., Chen, Z., Huang, Z., Li, Z., Zhu, X., Hu, X., Lu, T., Lu, L., Li, H., Wang, X., & Qiao, Y. (2023). Internimage: Exploring large-scale vision foundation models with deformable convolutions. http://arxiv.org/abs/2211.05778

Wei, Y., Xia, W., Huang, J., Ni, B., Dong, J., Zhao, Y., & Yan, S. (2014). Hcp: A flexible cnn framework for multi-label image classification. https://doi.org/10.1109/TPAMI.2015.2491929

Wendler, R. (1999). Ldi update: Metadata in the library. *Library Notes*, *1286*.

Witbrock, M. J., & Hauptmann, A. G. (1997). Using words and phonetic strings for efficient information retrieval from imperfectly transcribed spoken documents. https://doi.org/10.1145/263690.263779

Xiao, R., Zhu, L., & Zhang, H. J. (2003). Boosting chain learning for object detection. *Proceedings of the IEEE International Conference on Computer Vision*, *1*, 709–715. https://doi.org/10.1109/iccv.2003.1238417

Yu, J., Li, J., Yu, Z., & Huang, Q. (2019). Multimodal transformer with multi-view visual representation for image captioning. *IEEE Transactions on Circuits and Systems for Video Technology*, *30*, 4467–4480. https://doi.org/10.1109/TCSVT.2019.2947482

Zhang, M. L., & Zhou, Z. H. (2014). A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, *26*, 1819–1837. https://doi.org/10.1109/TKDE.2013.39

Zheng, S., Lu, J., Zhao, H., Zhu, X., Luo, Z., Wang, Y., Fu, Y., Feng, J., Xiang, T., Torr, P. H. S., & Zhang, L. (2021). Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers. https://fudan-zvg.github.io/SETR

Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., & Torralba, A. (2018). Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *40*(6), 1452–1464. https://doi.org/10.1109/TPAMI.2017.2723009

Zhou, H., Chen, M., & Webster, M. F. (2002). Comparative evaluation of visualization and experimental results using image comparison metrics. https://cs.swan.ac.uk/reports/yr2002/CSR28-2002.pdf

Appendix A

Overview of Communication with TelevisionCo.

Table A.1 provides a concise overview of the content of each in-person meeting with TelevisionCo. Besides these meetings, during the past six months, there has been constant e-mail communication between the authors of this thesis and their contact persons at TelevisionCo. Below, a few snippets from the selected meetings are provided that are relevant to the decisions made in this thesis.
Relevant statements by employees of TelevisionCo from the brainstorm session:

- "Creating the metadata takes a long time and is a very manual process."
- "Creating metadata feels like it is taking time from other tasks, but it is super important since it helps us navigate our archives."
- "It would be awesome if a model could do this for us."
- "Even if the output produced by such a model is not perfect, we can imagine it still helps us a lot and saves us time in our daily work."

Event	Topics	Date	Duration
Virtual Introduction	- Get-to-know the team	10-11-2022	30 mins
Brainstorm	- Explaining core machine learning capabilities - Brainstorming possible use cases (Figure A.1) - Collecting topic - Evaluating pro's and con's of each topic (Figure A.2) - Final decision: recognizing people and objects in videos	16-11-2022	120 mins
Topic Delimitation	- Inspecting raw data archive to get an impression of what is (not) important - Discussing access, legal requirements, etc. Drawing first architecture (Figure A.3)	07-12-2022	60 mins
Kick-Off	- Presentation of project plan (Figure A.4) - Discussion of time line	09-01-2023	60 mins
Archive Skifting	- Presentation and discussion of prototype results - Asking for 50 properly described videos - Discussion of possible expansions of AI architecture - Viewing raw archive material	27-01-2023	60 mins
Initial Results	- Presentation of results - Final input from TelevisionCo	15-03-2023	60 mins
Final Results	- Presentation of final results and workings of model(s)	TBD	60 mins
Implementation	- Discussing how to implement proposed solution in TelevisionCo.	TBD	120 mins

Table A1: Overview of Meetings Between Authors and TelevisionCo.

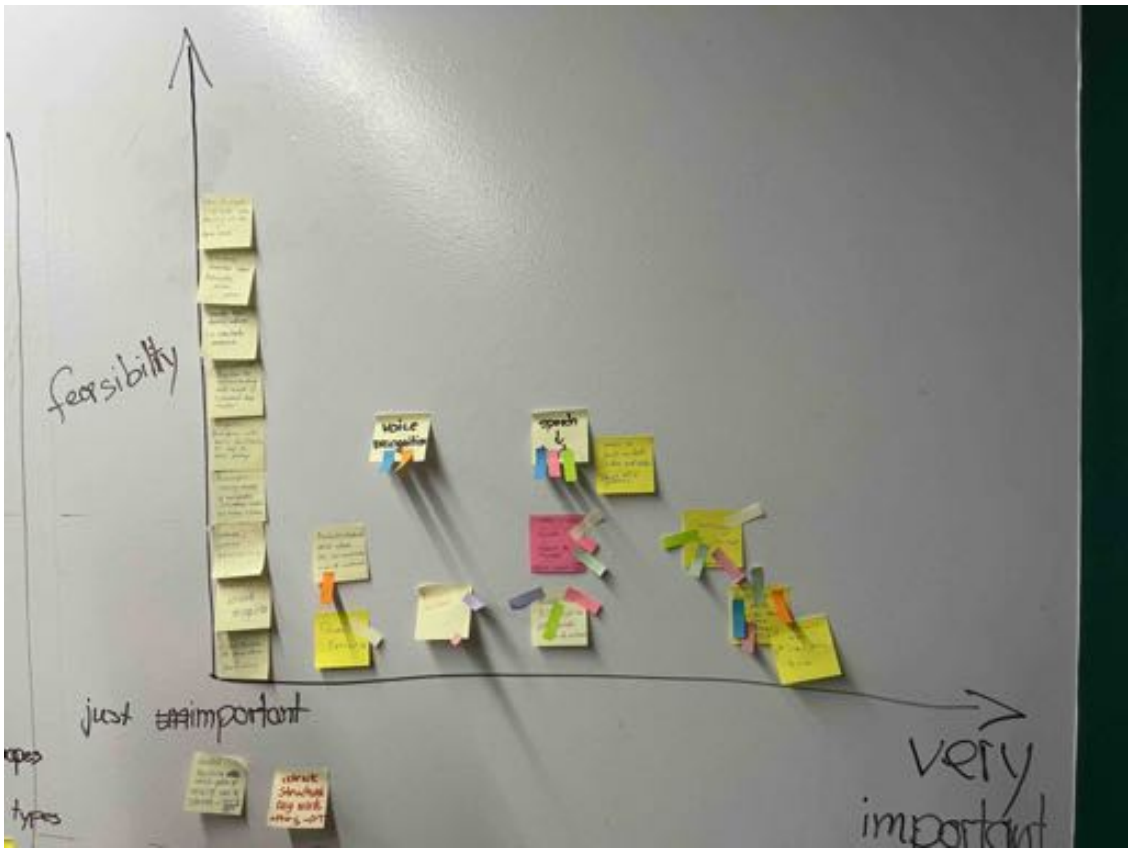


Figure A.1: Overview of Brainstorm Session with TelevisionCo.

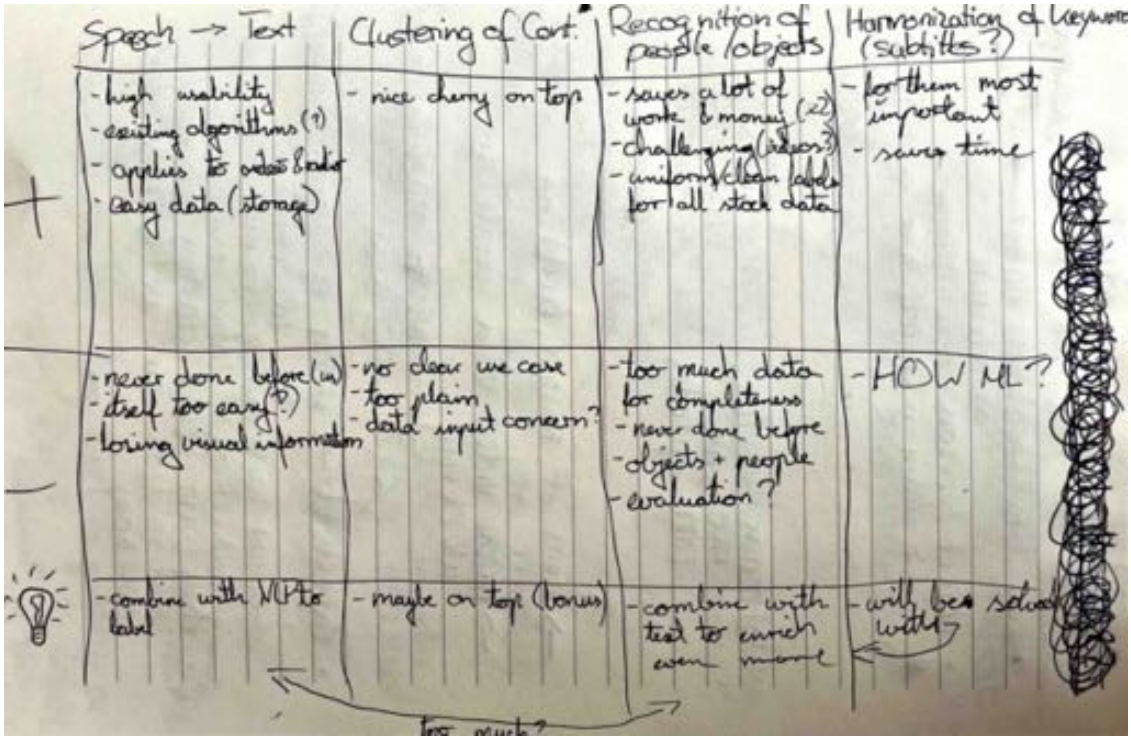


Figure A.2: Evaluation of Pro's and Con's per Topic from Brainstorm Session.

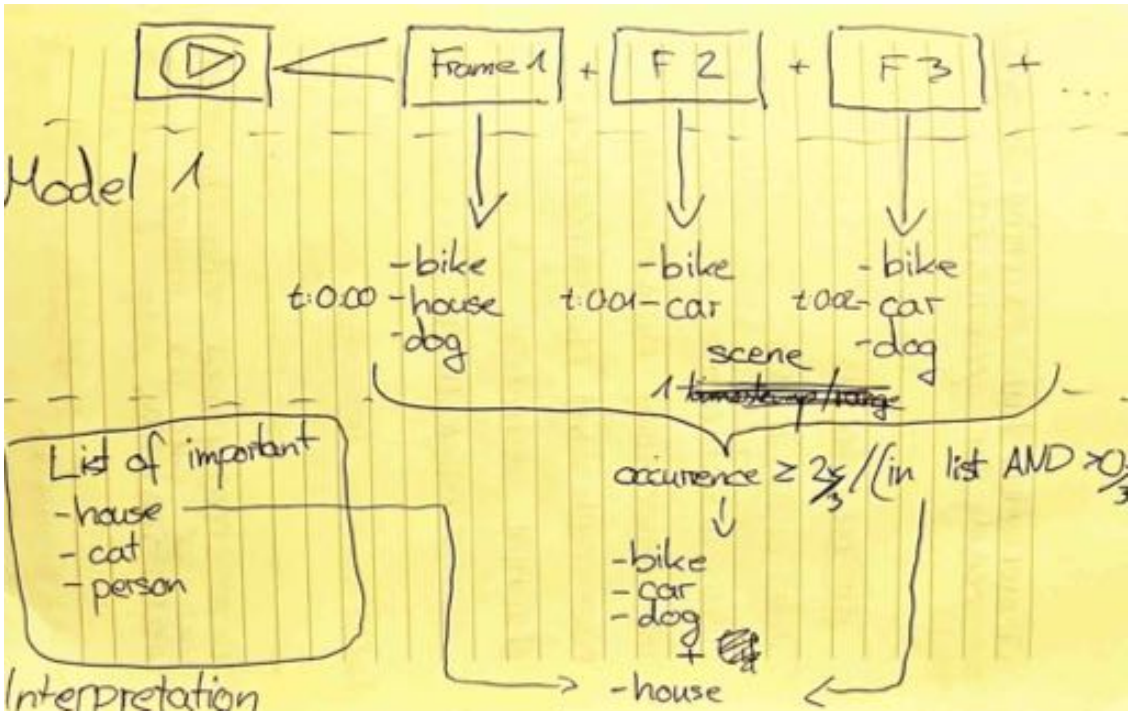


Figure A.3: Initial Architecture.

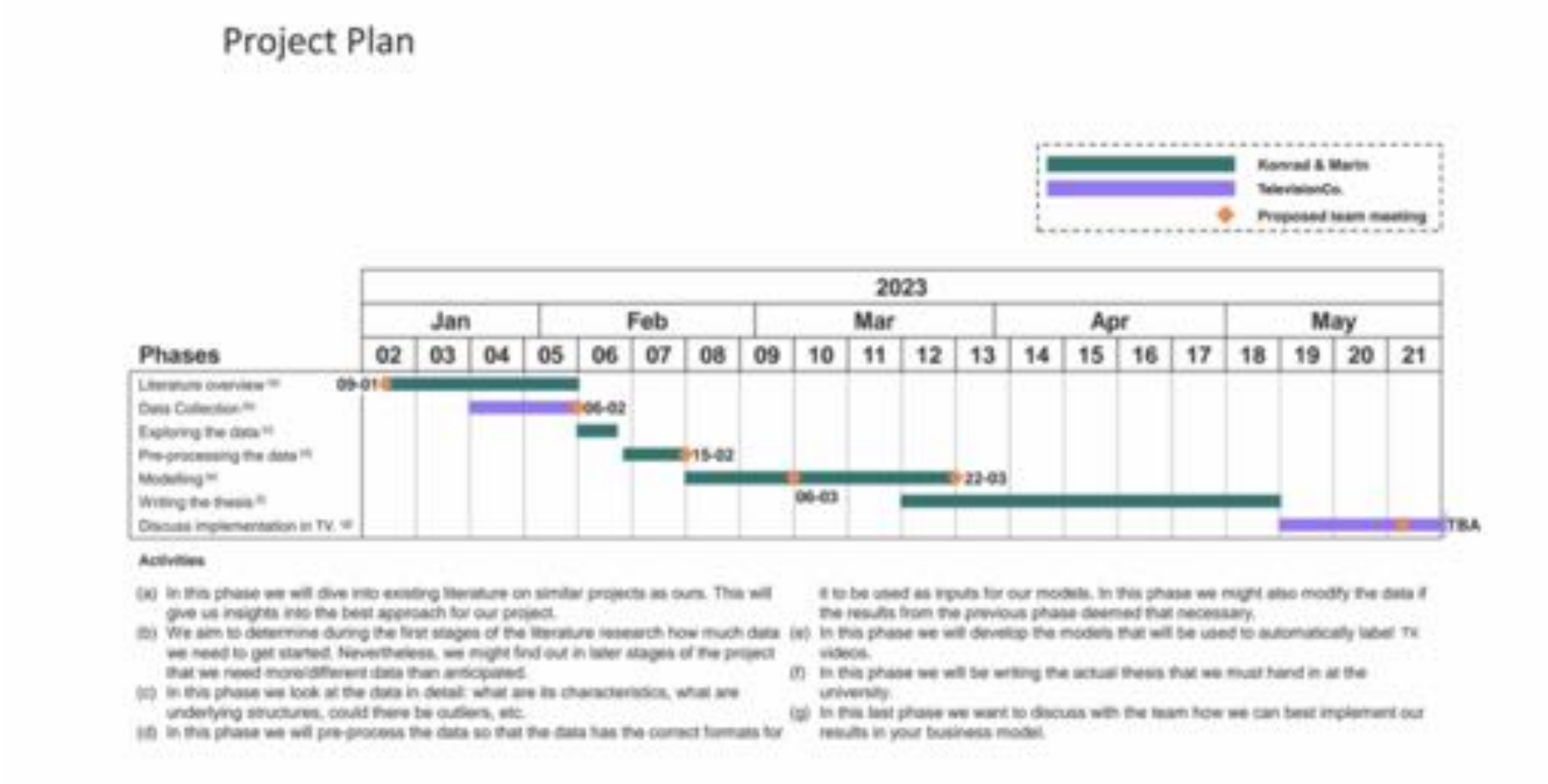


Figure A.4: Proposed Project Plan.

Appendix B

Supplementary Files from TelevisionCo.

B.1 List of Abbreviations & Fixed Terms

B.1.1 Abbreviations

opt. = optagelser

Aftenopt.

Anonyme opt.: Bruges om optagelser, typisk personer, som er ikke-genkendelige.

Delvist anonyme opt.

Droneopt.

Efterårsopt.

Ext.: Eksteriør-optagelser

Forårsopt.

Gadeopt.: Fodgængere, biler cyklister. Hvad man forventer at se på en gade.

Gen. opt.: Generelle optagelser

Gående opt.: Også kaldet håndholdt kamera. Kameraet føres af kameramanden (m/k) og følger vedkommendes bevægelser

Halvtot.: halvtotal - en del af personens torso i billedet.

Indendørs opt.: Interiør-optagelser

Int.: Interiør-optagelser

Intv.: Interview

Konstruerede opt.: En iscenesat genfremstilling af en ikke-specifik begivenhed.

Kan bruges mere bredt end "Rekonstruktion", da konstruerede optagelser ikke henviser til en bestemt begivenhed.

Liveintv.: live interview

Luftopt.

Lytteopt. (efterfulgt af navn på person): Bruges om kendte personer, der forekommer i genbrugsegnede optagelser, dvs. hvor personen ikke taler under interview. Personen skal være i fokus og alene i billedet.

Mikroskopopt.: Optagelser set gennem et mikroskop.

Natopt.

Neutrale opt.: Bruges om optagelser, typisk personer, som er ikke-genkendelige.

Nær: Næroptagelser

Næropt.

Pan.: panorering. En horisontal bevægelse omkring en akse. Kameraet flytter ikke position.

Sommeropt.

Sejlende opt.

Stemningsopt.

Supernæropt.

Tot.: Total - hele personen/objektet ses i billedet

Tr.s.: Kørende optagelser

Udendørs opt.: Eksteriør-optagelser

Undervandsopt.

Vinteropt.

B.1.2 Fixed Terms

Crane shot: kamera kører op/ned - monteret på en kran.

Defokus: objekter eller planer i billedet, som har den mindste skarphed

Fast indstilling: optagelser med et fast indstillet kamera

Fodgængere

Fokus: objekter eller planer i billedet, som har den største skarphed eller er centralt placeret i billedet

Foto/fotos: fotografi/fotografier

Frøperspektiv: kamera er placeret under øjenhøjde og ser altså begivenheden nedenfra

Frys: et enkelt billede fastholdes

Fugleperspektiv: kamera er placeret over øjenhøjde og ser altså begivenheden oppefra

Hovedsæde

Live: Når noget sendes direkte, en direkte transmission af en udsendelse eller dele heraf

Navnetræk

Rekonstruktion: En iscenesat genfremstilling af en specifik begivenhed.

Tilt: en vertikal bevægelse omkring en akse. Kameraet flytter ikke position.

Voxpop: vox populi ('folkets stemme') er interview med en række tilfældige personer om et givet emne.

Vue: optagelser hvor man fra et fast punkt ser ud over et større område.

Zoom: kameraet forbliver stationært mens linsens brændvidde forskydes. Bevægelsen af billede bliver fra det større til det mindre - eller omvendt.

B.2 List of Valuable Object Types

Buildings, Facilities and Constructions

- *Building*: A single building understood in the classical narrow sense. Free-standing, clearly defined (castles, churches, town halls, hotels, banks etc.)
- *Facility*: Comprises at least the following two definitions:
  - Complexes consisting of more than one building (factories, educational institutions, hospitals, residential complexes etc.)
  - Complexes consisting of different types of objects (a harbour consisting of ships, quays, decks, containers, cargo, cranes, office buildings etc. A park consisting of paths, lakes, trees, pedestrian bridges, statues, pedestrians, flowers etc.)
- *Construction*: A humanmade standing structure that is not a building or a facility (entrance portals, wooden beacons, colonnades, breakwaters etc.)

Infrastructure

Comprises of at least the following two types:

- Transportation (bridges, railways, roads etc.)
- Supply (wind turbines, water treatment plants, power plants etc.)

Nature and Natural Phenomena

- *Nature*: comprises at least the following types:
  - Animals
  - Types of nature (forests, lakes, hills, coasts, etc.)
- *Natural phenomena*: sunrises, snow, rain, icicles, etc.

Agriculture and Horticulture

- Fields
- Crops
- Hay bales
- Gardens
- Flower beds, etc.

Technology and Electronics

- Mobile phones
- Computer chips
- Screens, etc.

Machines

- Drones
- Lawn mowers
- Industrial robots, etc.

Vehicles

- Cars
- Trains
- Ferries
- Tanks, etc.

Medical Equipment and Pharmaceuticals

- Pills
- Syringes and needles
- Drug packaging with labelling, etc.

Art

- Sculptures
- Monuments
- Street art
- Fountains, etc.

People

- *Groups of people*: Characterized by different parameters such as group size, group affiliation, age composition, activity, setting etc. ("Small group of kindergarten children puddle jumping").
- *Individual persons*: Characterized by different parameters such as physical appearance, age, activity, setting etc. ("Middle aged man with long beard dancing on a boat". "Pregnant woman in an office").

Please keep in mind that this is just a list of objects we know typically would be of relevance to us. This does not mean that objects not covered by the list fundamentally has lesser relevance. Essentially it is not possible to make a prioritized list as every object has equal potential reuse value depending on the use case.

Appendix C

Word Count Delimitation

Below a collection of line plots is presented. These line plots represent the tests conducted in order to find the optimal number of words in the final output, per individual model in the frame description collection. The optimal number of words is found based on which output generated the highest performance.

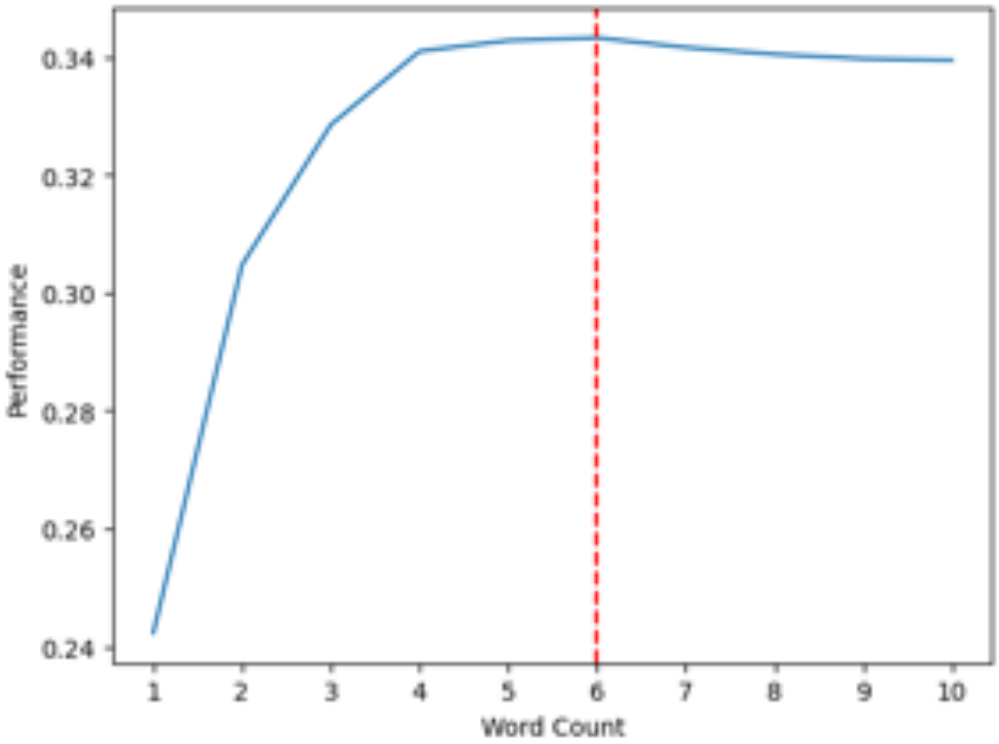


Figure C.1: Line Plot Highlighting Optimal Number of Words for (Individual) IC Model.

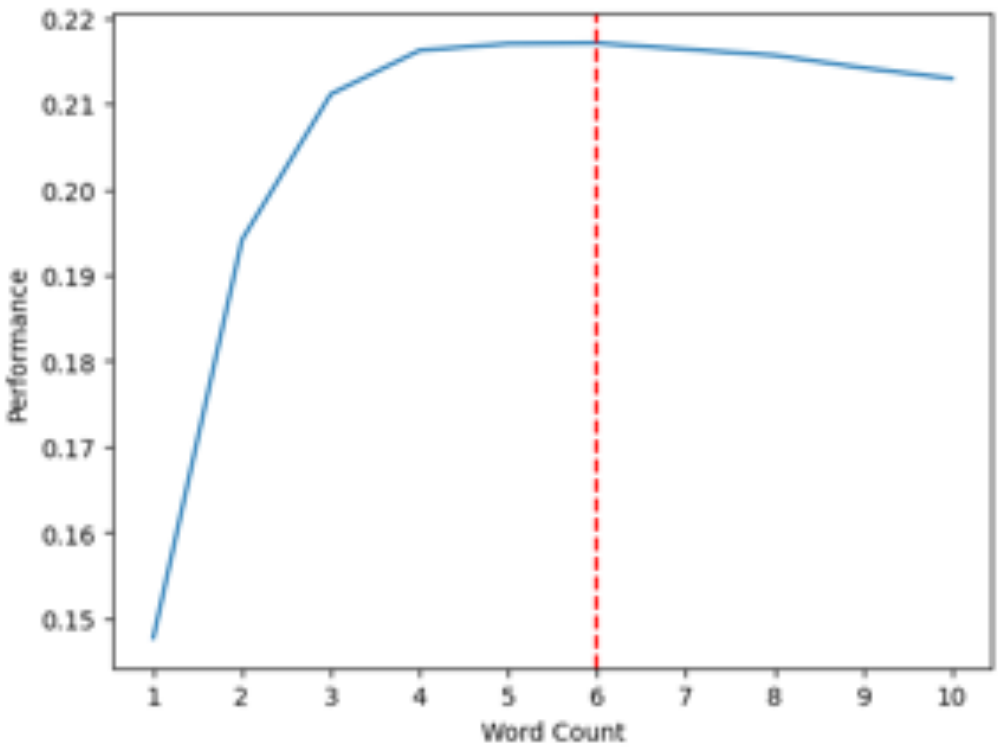


Figure C.2: Line Plot Highlighting Optimal Number of Words for (Individual) IS Model.

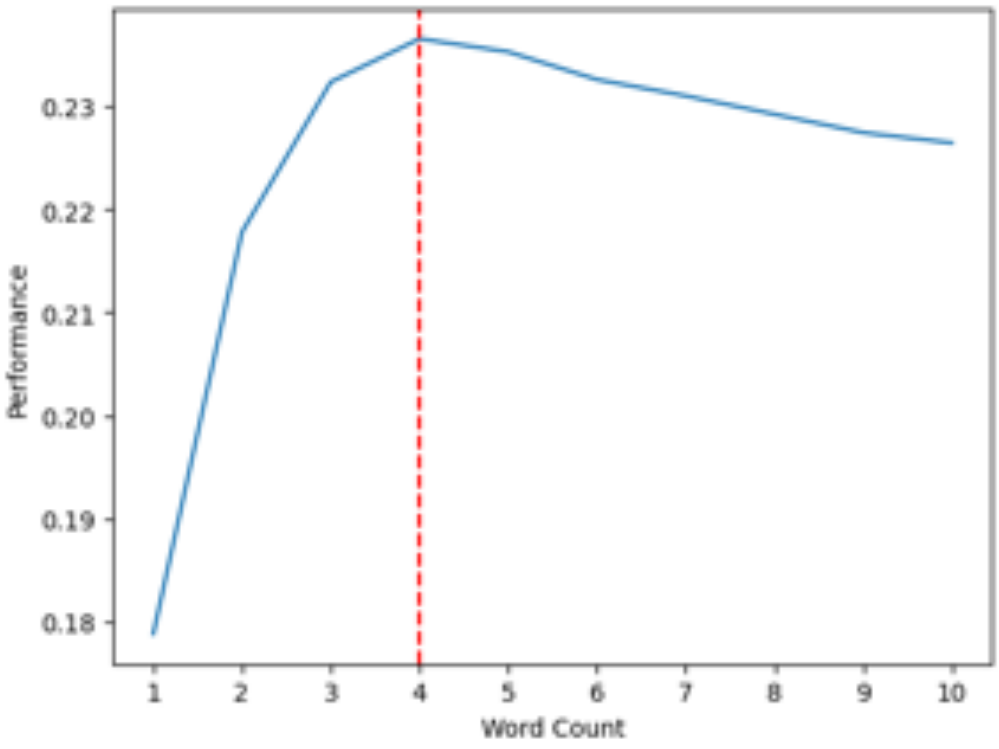


Figure C.3: Line Plot Highlighting Optimal Number of Words for (Individual) OD Model.

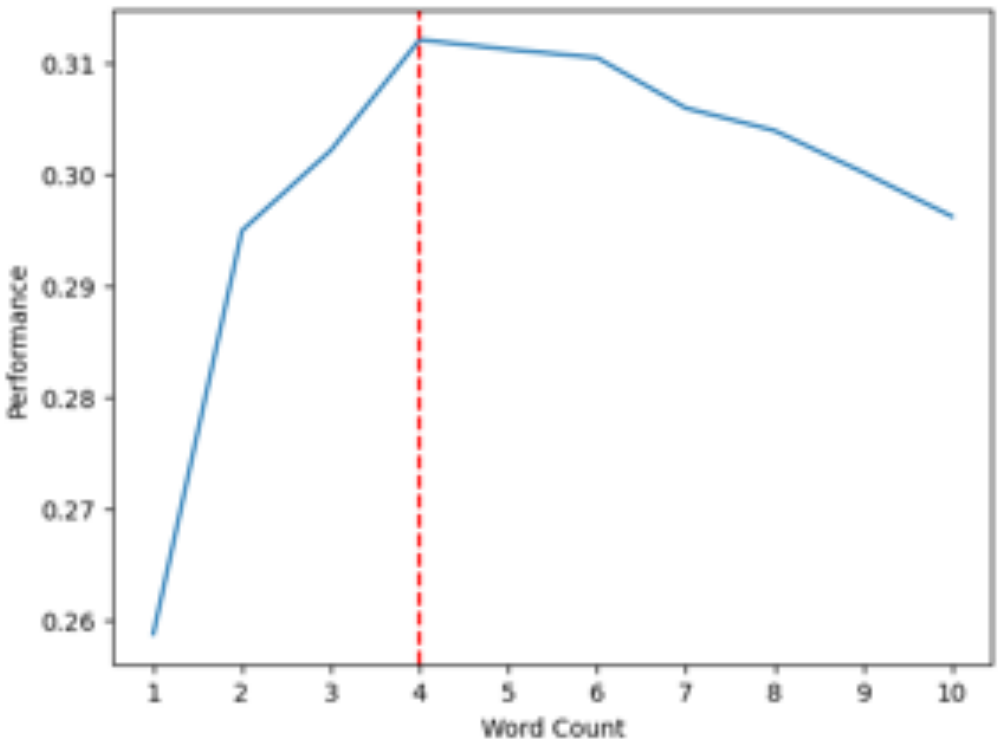


Figure C.4: Line Plot Highlighting Optimal Number of Words for (Individual) MC Model.

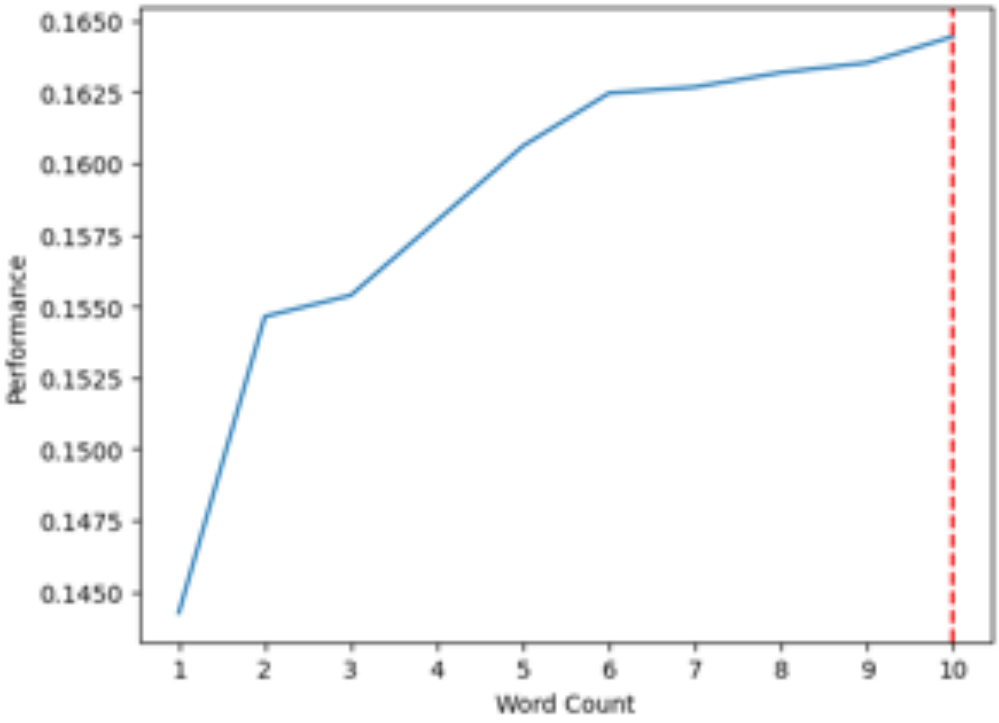


Figure C.5: Line Plot Highlighting Optimal Number of Words for (Individual) TR Model (Maximal 10 Words).

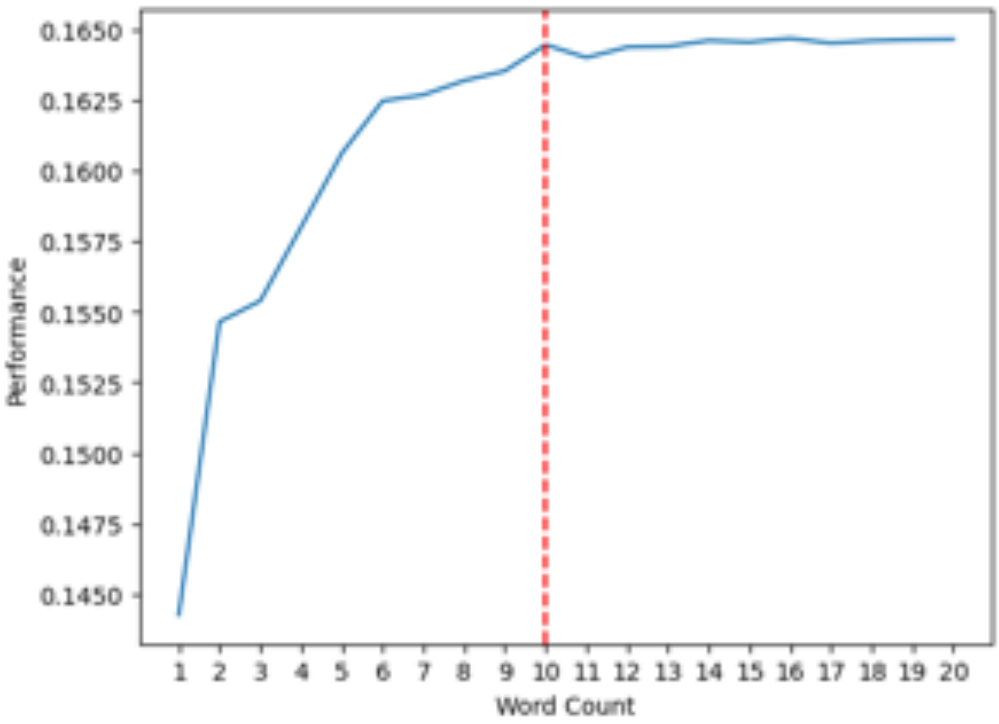


Figure C.6: Line Plot Highlighting Optimal Number of Words for (Individual) TR Model (Maximal 20 Words).

Appendix D

Correlation

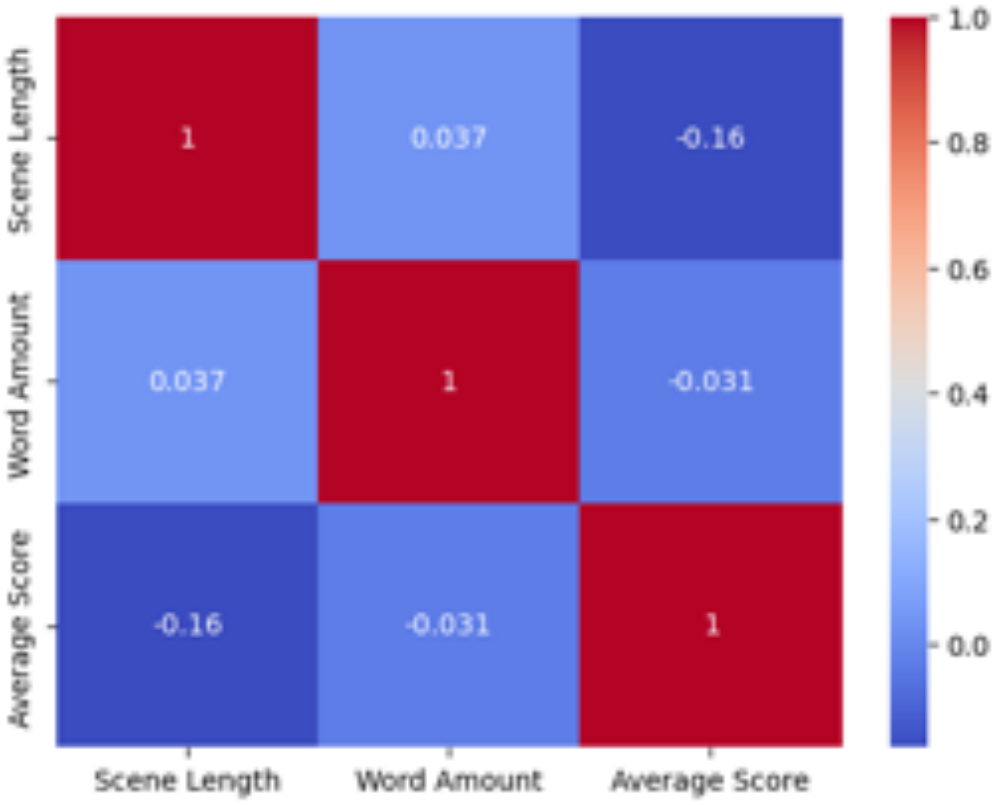


Figure D.1: Correlation Matrix for Correlation Between Scene Lengths and Performances.

Appendix E

Upset Plots

Below a collection of upset plots is presented. Each upset plot includes all ensembles containing a specific individual model. Note that the performance metric, similarity score, needs to be divided by 100 to for the correct number.

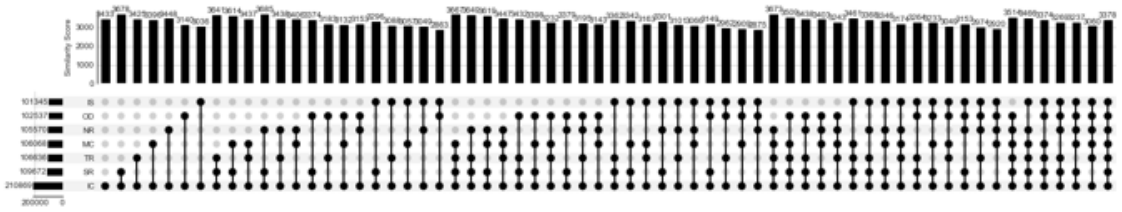


Figure E.1: Upset Plot Describing Performance of each Ensemble Including Image Captioning (IC), Sorted by Ensemble Size.

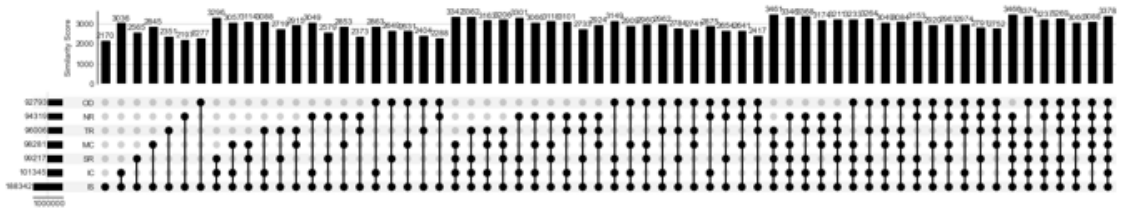


Figure E.2: Upset Plot Describing Performance of each Ensemble Including Image Segmentation (IS), Sorted by Ensemble Size.

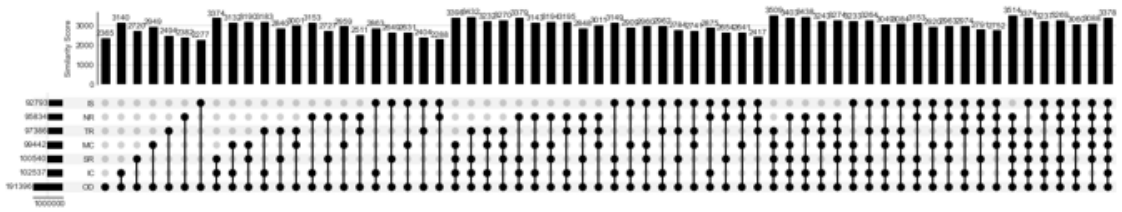


Figure E.3: Upset Plot Describing Performance of each Ensemble Including Object Detection (OD), Sorted by Ensemble Size.

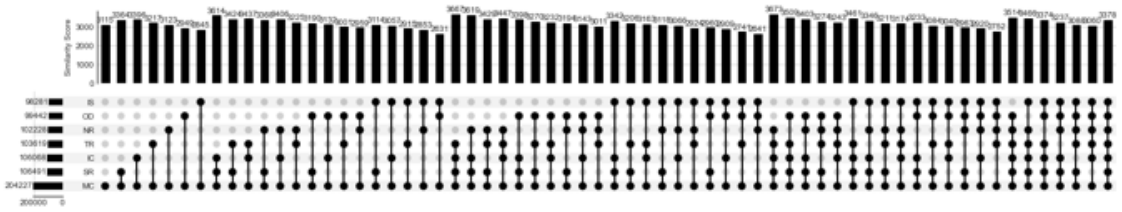


Figure E.4: Upset Plot Describing Performance of each Ensemble Including Multilabel Classification (MC), Sorted by Ensemble Size.

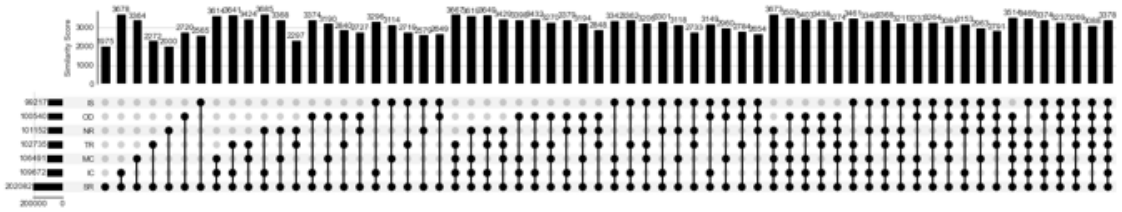


Figure E.5: Upset Plot Describing Performance of each Ensemble Including Scene Recognition (SR), Sorted by Ensemble Size.

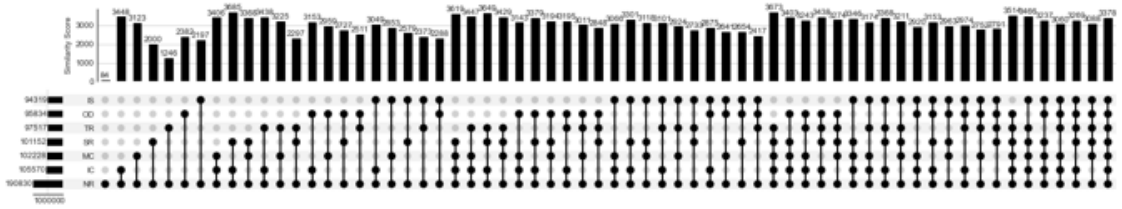


Figure E.6: Upset Plot Describing Performance of each Ensemble Including Night Recognition (NR), Sorted by Ensemble Size.

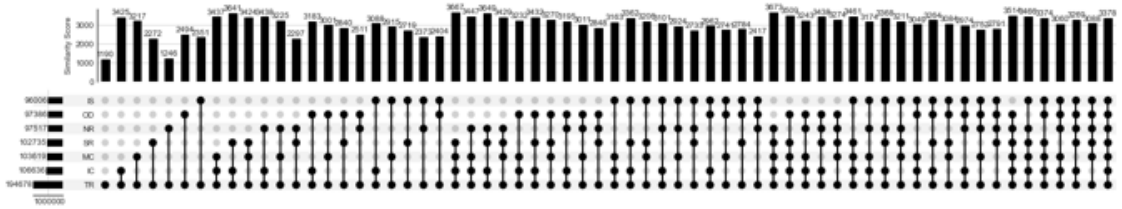


Figure E.7: Upset Plot Describing Performance of each Ensemble Including Text Recognition (TR), Sorted by Ensemble Size.

Appendix F

Code

F.1 Pipeline

F.1.1 Requirements

```
# General
import os
import time
from time import strftime
import pandas as pd
import numpy as np
import cv2
import datetime
from datetime import timedelta
from PIL import Image
import torch
import tensorflow as tf
import tensorflow_hub as hub
import torch.nn.functional as F
from transformers import pipeline

### 1 Data Extraction

# 1.A Video Metadata Extraction
import xml.dom.minidom
from deep_translator import GoogleTranslator

# 1.B Video Frame Extraction
import cv2

# 1.C Duplicate Removal
from sewar.full_ref import mse

### 2 Methods

# 2.A Frame Description
```

```
32
33  # 2.A.a Image Captioning (IC)
34  image_to_text = pipeline("image-to-text",
    ↪  model="nlpconnect/vit-gpt2-image-captioning")
35
36  # Tokenization
37  import nltk
38  import spacy
39  from nltk.corpus import stopwords
40  from nltk.tokenize import sent_tokenize
41  nlp = spacy.load("en_core_web_sm")
42
43  # 2.A.b Image Segmentation (IS)
44  from transformers import MaskFormerFeatureExtractor,
    ↪  MaskFormerForInstanceSegmentation
45  feature_extractor =
    ↪  MaskFormerFeatureExtractor.from_pretrained("facebook/
    ↪  maskformer-swin-tiny-ade")
46  segmentor =
    ↪  MaskFormerForInstanceSegmentation.from_pretrained("facebook/
    ↪  maskformer-swin-tiny-ade")
47
48  # 2.A.c Object Detection (OD)
49  detector = hub.load("https://tfhub.dev/google/
    ↪  openimages_v4/ssd/mobilenet_v2/1").signatures['default']
50
51  # 2.A.d Multilabel Classification (MC)
52  import requests
53  module = hub.KerasLayer("https://tfhub.dev/google/
    ↪  bit/m-r101x3/imagenet21k_classification/1")
54
55  # 2.B Scene Recognition (SR)
56  from torch.autograd import Variable as V
57  import torchvision.models as models
58  from torchvision import transforms as trn
59
60  # 2.C Night Recognition (NR)
61
62  # 2.D Text Recognition (TR)
63  from paddleocr import PaddleOCR, draw_ocr
64  paddle_ocr_model = PaddleOCR(use_angle_cls=True,lang='da')
65  ner = pipeline(task='ner',
    ↪  model='saattrupdan/nbailab-base-ner-scandi',
    ↪  aggregation_strategy='first')
66
67  ### 3. Summarization
68  from collections import Counter
```

F.1.2 Preprocessing

Metadata Extract

```
def extract_metadata(folder_path, xml_file, evaluation=False):

    """
    This function takes a path (folder_path) and xml file (xml_file)
    as inputs and outputs a dataframe containing the id of the scene,
    the start time of a scene, the end time of a scene and the
    description of the scene. There are no additional arguments.
    """

    if evaluation==True:
        print("[EVALUATION MODE] Extracting Metadata...")
    else:
        print("Extracting Metadata...")
    # Start timer
    start_timer = time.time()

    domtree=xml.dom.minidom.parse(folder_path+'/'+xml_file+'.xml')
    group = domtree.documentElement

    # Get duration of video
    duration =
    ↪  datetime.datetime.strptime(group.getElementsByTagName('media_
    ↪  object_group')[0].getElementsByTagName('mog_duration')[0]
    ↪  .childNodes[0].nodeValue,"%H:%M:%S.%f").strftime("%H:%M:%S")

    # Extract timecode information
    metadata = group.getElementsByTagName('timecode_description')

    # Create empty lists
    scene_id = []
    start_time = []
    descriptions = []

    # Looping through the .xml file
    for i, scene in enumerate(metadata):
        # Get the scene-id
        scene_id.append(scene.getAttribute('tid_id'))

        # Get the start-time
        timestamp = scene.getElementsByTagName('tid_timecode')[0]
        ↪  .childNodes[0].nodeValue
        timestamp = datetime.datetime.strptime(timestamp,
        ↪  "%H:%M:%S.%f")
        rounded_timestamp = timestamp.strftime("%H:%M:%S")
        start_time.append(rounded_timestamp)
```

```
38
39          # Get the description
40          try:
41              description =
                ↪  scene.getElementsByTagName('tid_description')[0]
                ↪  .childNodes[0].nodeValue
42          except IndexError:
43              description = ''
44          descriptions.append(description)
45
46      """Only necessary for evaluation phase!"""
47
48      # Create a list of abbreviations which shall not be translated
        ↪  and will therefore be removed to later be
49      # added (in their original form) again
50      abbreviations = ["Natop. ", "Natopt. ", "Næropt. ","Halvnær opt.
        ↪  ","Supernær opt. ","Ext. opt.", \
51                       "Ext. ","Int. opt.","Int.
                         ↪  ","Aftenopt.","Natopt.","Natop.","Anonyme
                         ↪  opt.","Gen. opt.","Gen. "]
52      # Create a ist for all removed abbreviations and the translations
        ↪  (without the removed abbreviations)
53      all_removals = []
54      raw_translations = []
55
56      # Loop through all descriptions
57      for sentence in descriptions:
58
59          # Create a list for all removed abbreviations for one
            ↪  specific sentence
60          removed = []
61
62          # Check for all abbreviations (also lower case) if they are
            ↪  contained in a sentence.
63          # If yes, remove them from this sentence and add them to the
            ↪  "removed" list
64          for word in abbreviations:
65              if word in sentence:
66                  sentence = sentence.replace(word,"")
67                  removed.append(word)
68              if word.lower() in sentence:
69                  sentence = sentence.replace(word.lower(),"")
70                  removed.append(word.lower())
71
72          # Add the shortened sentence to the to-be-translated list
73          # and the removed abbreviations to the "all_removals" list
74          raw_translations.append(sentence)
75          all_removals.append(removed)
```

```
76
77      # Translate all shortened sentences to english (for comparison)
78      raw_translations = GoogleTranslator('da',
        ↪  'en').translate_batch(raw_translations)
79
80      # Exchange all NoneTypes with ""
81      raw_translations = ["" if item is None else item for item in
        ↪  raw_translations]
82
83      # Add the removed abbreviations back to the beginning of each
        ↪  translated sentence
84      descriptions = [" ".join(all_removals[i]) + " " +
        ↪  raw_translations[i] for i in range(len(all_removals))]
85
86      # create dataframe and sort values (due to weird structure in
        ↪  DR's .xml files)
87      comparison_df = pd.DataFrame({'Scene ID': scene_id, 'Start-Time':
        ↪  start_time, 'Ground Truth Description': descriptions})
88      comparison_df = comparison_df.sort_values(by=['Start-Time'],
        ↪  ascending=True, ignore_index=True)
89
90      # create a list for the end time (we need start and end in order
        ↪  to know how to summarize our generated output)
91      end_time = []
92
93      # get the end-time based on the start-time
94      for i in range(len(comparison_df['Start-Time']) - 1):
95          start =
            ↪  datetime.datetime.strptime(comparison_df['Start-Time']
            ↪  [i+1], "%H:%M:%S")
96          end = start - datetime.timedelta(seconds=1)
97          end_time.append(end.strftime("%H:%M:%S"))
98      end_time.append(duration)
99
100     # Add the end-time to the dataframe
101     comparison_df['End-Time'] = end_time
102
103     # Re-order the columns of the dataframe and return the dataframe
104     comparison_df = comparison_df[["Scene
        ↪  ID","Start-Time","End-Time","Ground Truth Description"]]
105
106     # Remove uncommented scenes, scenes describing the whole video
        ↪  (starting with "Background:" or "Description")
107     # and scenes with incorrectly extracted End-Times (end time of
        ↪  23:59:59 or earlier than start-time) (only for evaluation)
108     if evaluation == True:
109         for row in range(len(comparison_df)):
```

```
110             if comparison_df["Ground Truth Description"][row] == " "
                ↪  \
111             or "Background: " in comparison_df["Ground Truth
                ↪  Description"][row] \
112             or "Description: " in comparison_df["Ground Truth
                ↪  Description"][row] \
113             or comparison_df["End-Time"][row] == "23:59:59" \
114             or (datetime.datetime.strptime(
                ↪  comparison_df["End-Time"][row],"%H:%M:%S")\
115                 - datetime.datetime(1900,1,1)).total_seconds() \
116                 - (datetime.datetime.strptime(
                    ↪  comparison_df["Start-Time"][row],"%H:%M:%S")\
117                 - datetime.datetime(1900,1,1)).total_seconds()  < 0:
118                 comparison_df.drop(row, axis=0, inplace=True)
119         comparison_df = comparison_df.reset_index(drop=True)
120
121     # Print duration of extraction
122     print("Metadata for video '" + str(xml_file) + "' was extracted
        ↪  in " + str(round(time.time()-start_timer,2)) \
123           + " seconds. (" +
              ↪  str(round(round(time.time()-start_timer,2)/
              ↪  len(comparison_df),2)) + "s/scene)")
124
125     return comparison_df
```

Frame Extract

```
1  def extract_frames(folder_path,video_id,metadata_dataframe,
   ↪  resize=False,fps_of_video=25,frame_frequency=1,
   ↪  evaluation=False):
2
3      """  This function takes a path and a video id (name) as inputs
       ↪  and outputs a dataframe containing the frames """
4      """  and their timestamp. Additional arguments are: fps of the
       ↪  video and frame frequency (defining in seconds """
5      """  in which interval a frame should be extracted).
       ↪  """
6
7      if evaluation==True:
8          print("[EVALUATION MODE] Extracting Frames...")
9      else:
10         print("Extracting Frames...")
11     # Start timer
12     start = time.time()
13
14     # Create two empty lists: one for the frames and one for their
       ↪  timestamps
15     frames = []
16     timestamps = []
```

```
17
18      # Create a list of all videos in folder
19      video_files = os.listdir(folder_path)
20
21      # Filter list for all videos containing this video ID (in case of
        ↪  splits possibly more than 1)
22      filtered_list = [name for name in video_files if video_id in
        ↪  name]
23
24      # Creating the folder
25      for video in filtered_list:
26          video_name = os.path.join(folder_path, video)
27          cap = cv2.VideoCapture(video_name)
28          total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
29          interval = fps_of_video*frame_frequency
30
31          # Extracting the frames
32          for i in range(0, total_frames, interval):
33              cap.set(cv2.CAP_PROP_POS_FRAMES, i)
34              ret, frame = cap.read()
35
36              # Get necessary delta from video title
37              if "-" not in filtered_list[0]:
38                  delta = 0
39              else:
40                  zero_time = datetime.datetime.strptime("00.00.00",
                    ↪  "%H.%M.%S")
41                  duration = video.split("-")[1][:-4]
42                  duration = datetime.datetime.strptime(duration,
                    ↪  "%H.%M.%S")
43                  delta_duration = duration-zero_time
44                  delta = delta_duration.total_seconds()
45
46              # Get the timestamp of the frame
47              if ret:
48                  timestamp = cap.get(cv2.CAP_PROP_POS_MSEC)/1000
49                  formatted_timestamp =
                    ↪  str(datetime.timedelta(seconds=timestamp+delta))
                    ↪  # add delta depending on earlier statement more
                    ↪  seconds!
50                  # Save the frame with the timestamp as the file name
51                  frames.append(frame)
52                  timestamps.append(formatted_timestamp)
53
54          # Release the video capture object
55          cap.release()
56
57      # Resize frames if resize factor is defined
```

```
58      if resize != False:
59          frames = [cv2.resize(frame, (round(frame.shape[1]*resize),
            ↪ round(frame.shape[0]*resize)),
            ↪ interpolation=cv2.INTER_CUBIC) for frame in frames]
60
61      # Create dataframe with frames and timestamps
62      dataframe = pd.DataFrame({"Timestamp":timestamps,"Frame":frames})
63
64      # Remove frames of uncommented scenes (only for evaluation)
65      if evaluation == True:
66          # Convert Timestamps of Frames to seconds
67          frames_timestamp_seconds = [(datetime.datetime.strptime(
            ↪ dataframe.iloc[row]["Timestamp"],"%H:%M:%S") -
            ↪ datetime.datetime(1900,1,1)).total_seconds() for row in
            ↪ range(len(dataframe))]
68
69          # Convert Start and End Times of Metadata to Scene Ranges
70          scene_ranges = [range(int((datetime.datetime.strptime(
            ↪ metadata_dataframe.iloc[row]["Start-Time"],"%H:%M:%S") -
            ↪ datetime.datetime(1900,1,1)).total_seconds()),
            ↪ int((datetime.datetime.strptime(
            ↪ metadata_dataframe.iloc[row]["End-Time"], "%H:%M:%S")
            ↪ -datetime.datetime(1900,1,1)).total_seconds()+1)) for row
            ↪ in range(len(metadata_dataframe))]
71
72          for row in range(len(frames_timestamp_seconds)):
73              if not [1 for scene in scene_ranges if
                ↪ frames_timestamp_seconds[row] in scene]:
74                  dataframe.drop(row, axis=0, inplace=True)
75          dataframe = dataframe.reset_index(drop=True)
76
77      # Print amount of extracted frames and duration of extraction
78      print(str(len(dataframe))+"(/" + str(len(frames)) + ") frames
        ↪ were extracted in " + str(round(time.time()-start,2)) + "
        ↪ seconds. (" \
79            + str(round(round(time.time()-start,2)/len(dataframe),2)) +
              ↪ "s/frame)")
80
81      # Store the amount of (extracted) frames we will be working with
        ↪ from now on
82      raw_frames = len(dataframe)
83
84      return dataframe, raw_frames
```

Duplicate Removal

```
1   def drop_duplicates(dataframe,mse_threshold=100):
2
3       """
```

```
4       This function takes a dataframe, a frame column name and a
    ↪   threshold for the MSE measurements as input,
5       filters out all duplicate frames (i.e. one of two frames too
    ↪   similar according to MSE measures, ergo below
6       the given threshold (default=100)) and adds a new column stating
    ↪   the amount of duplicates of one frame.
7       """
8
9       print("Finding and removing duplicates ...")
10      # Start timer
11      start = time.time()
12
13      # Create a list for counting the duplicates and another for
        ↪  dropping the duplicated frames
14      duplicate_counter = []
15      drop_indices = []
16
17      # Loop through all rows of the dataframe
18      for frame in range(len(dataframe)-1):
19
20          # Calculate MSE for every frame in comparison to its
            ↪  following (i.e. one second later) frame
21
22          # In case MSE is bigger than threshold, nothing is dropped
            ↪  and the frame's occurence is set to "1"
23          if mse(dataframe["Frame"][frame],dataframe["Frame"][frame+1])
            ↪  >= mse_threshold:
24              if frame == 0:
25                  duplicate_counter.append(1)
26                  duplicate_counter.append(1)
27              else:
28                  duplicate_counter.append(1)
29
30          # If MSE is smaller than threshold, the duplicated (next
            ↪  frame) will be dropped and the duplicate
31          # counter for this frame will be incresed by "1"
32          else:
33              drop_indices.append(frame+1)
34              if frame == 0:
35                  duplicate_counter.append(2)
36              else:
37                  duplicate_counter[-1] += 1
38
39      # Drop all duplicates from dataframe
40      dataframe = dataframe.drop(drop_indices)
41
42      # Add Counter column to dataframe
43      dataframe["Counter"] = duplicate_counter
```

```
44
45      print(str(len(drop_indices)) + " duplicates (" +
        ↪  str(round(len(drop_indices)/(len(duplicate_counter) +
        ↪  len(drop_indices))*100,2)) + "%) were found and removed in "
        ↪  + str(round(time.time()-start,2)) + " seconds. (" +
        ↪  str(round(round(time.time()-start,2)/(len(drop_indices) +
        ↪  len(dataframe)),2)) + "s/frame)")
46
47      unique_frames = len(duplicate_counter)
48
49      return dataframe.copy().reset_index(drop=True), unique_frames
```

F.1.3 Methods

Image Captioning (IC)

```
1   # Function to convert the numpy arrays to a PIL images
2   def convert_to_pil_image(np_array):
3       imageRGB = cv2.cvtColor(np_array, cv2.COLOR_BGR2RGB)
4       return Image.fromarray(np.uint8(imageRGB))
5
6   def extract_ngrams(text):
7       doc = nlp(text)
8       ngrams = []
9       for chunk in doc.noun_chunks:
10          chunk_text = chunk.text
11          # Remove the leading "a" if present
12          if chunk_text.startswith('a '):
13              chunk_text = chunk_text[2:]
14
15          # Remove the leading "the" if present
16          if chunk_text.startswith('the '):
17              chunk_text = chunk_text[4:]
18
19          # Keep only the first adjective and the noun if two
            ↪  adjectives are combined with "and"
20          tokens = chunk_text.split(' ')
21          if len(tokens) >= 4 and tokens[1] == 'and':
22              chunk_text = ' '.join([tokens[0], tokens[3]])
23
24          ngrams.append(chunk_text)
25      return ngrams
26
27  def tokenize_IC(IC_predictions):
28
29      """
30      This function takes a dataframe and a column name as inputs and
    ↪   outputs the dataframe with a tokenized
31      (and adjusted for stopwords) text column.
```

```
32      """
33
34      print("Removing Stopwords and Extracting Noun Chunks...")
35      # Start timer
36      start_timer = time.time()
37
38      # Specifying that we want to drop specific words
39      stop_words = set(stopwords.words("english"))
40
41      # Create an empty list for tokenized predictions
42      tokenized_IC = []
43      for row in range(len(IC_predictions)):
44          tmp = []
45          sentences = sent_tokenize(IC_predictions[row])
46          for sent in sentences:
47              sent = sent.lower()
48
49              # Extract noun chunks (including nouns with their
                ↪  adjectives) using SpaCy
50              noun_chunks = extract_ngrams(sent)
51
52              # Filter out stopwords
53              filtered_noun_chunks = [chunk for chunk in noun_chunks if
                ↪  chunk not in stop_words]
54
55              tmp.extend(filtered_noun_chunks)
56
57          # Update dataframe with tokenized and adjusted description
58          tokenized_IC.append(tmp)
59
60      # Print duration of extraction
61      print("Tokenizing Predictions took " +
        ↪  str(round(time.time()-start_timer,2)) + " seconds. (" +
        ↪  str(round(round(time.time()-start_timer,2) /
        ↪  len(dataframe),2)) + "s/frame)")
62
63      return tokenized_IC
64
65  def IC(dataframe):
66
67      """
68      This function takes a dataframe and the column name of the frames
    ↪   as input and outputs the same dataframe extended with the image
    ↪   caption prediction from the
    ↪   'nlpconnect/vit-gpt2-image-captioning' model.
69      """
70
71      print("Captioning frames ...")
```

```
    # Start timer
    start = time.time()

    # Convert all arrays to images
    images = [convert_to_pil_image(frame) for frame in
    ↪  dataframe["Frame"]]

    # Predict Frame Captions
    captions = [image_to_text(image)[0]['generated_text'] for image
    ↪  in images] # [0]['generated_text'] to extract captions only

    # Tokenize IC predictions and update prediction list accordingly
    tokenized = tokenize_IC(captions)

    # Add Predictions to Dataframe
    dataframe["IC"] = tokenized

    # Print amount of predicted captions and duration of prediction
    print(str(len(captions)) + " captions were predicted in " +
    ↪  str(round(time.time()-start,2)) + " seconds. (" \
          + str(round(round(time.time()-start,2)/len(dataframe),2)) +
          ↪  "s/frame)")

    model1_time = round(time.time()-start,2)

    return dataframe.copy(), model1_time
```

Image Segmentation (IS)

```
def IS(dataframe,probability_threshold=0.9):

    """ This function takes a dataframe as input and outputs the same
    ↪  dataframe """
    """    extended with the image segmentation prediction from the
    ↪  \facebook/maskformer-swin-tiny-ade' model.    """

    print("Segmenting frames ...")
    # Start timer
    start = time.time()

    # create an empty list
    segments = []

    # Uses a for-loop to iterate through the files in our folder.
    for frame in dataframe["Frame"]: # change to dataframe later
        inputs = feature_extractor(images=frame, return_tensors="pt")
        outputs = segmentor(**inputs)
        class_queries_logits = outputs.class_queries_logits
```

```
18          probabilities_tensor = F.softmax(class_queries_logits,
            ↪  dim=-1)
19
20          # Convert the probabilities tensor to a NumPy array
21          probabilities_np = probabilities_tensor.detach().numpy()
22
23          # Find the unique class indices and their maximum
            ↪  probabilities
24          unique_class_indices =
            ↪  list(range(probabilities_np.shape[-1]))
25          max_probabilities = probabilities_np.max(axis=(0, 1))
26
27          # Zip the class indices and their maximum probabilities
            ↪  together
28          class_probabilities = list(zip(unique_class_indices,
            ↪  max_probabilities))
29
30          # Filter the class_probabilities to only include those with
            ↪  probabilities higher than 90%
31          high_prob_class_probabilities = [(class_idx, probability) for
            ↪  class_idx, probability in class_probabilities if
            ↪  probability > probability_threshold]
32
33          # Sort the high_prob_class_probabilities list in descending
            ↪  order of probability
34          high_prob_class_probabilities.sort(key=lambda x: x[1],
            ↪  reverse=True)
35
36          # Extract the first word of each high-probability class and
            ↪  store them in a new list
37          sorted_high_prob_first_words =
            ↪  [segmentor.config.id2label[class_idx].split(",")[0] for
            ↪  class_idx, _ in high_prob_class_probabilities if
            ↪  class_idx in segmentor.config.id2label]
38
39          # Store the labels
40          segments.append(sorted_high_prob_first_words)
41
42      # Add Predictions to Dataframe
43      dataframe["IS"] = segments
44
45      # Print amount of predicted captions and duration of prediction
46      print(str(len(segments)) + " frames were segmented in " +
        ↪  str(round(time.time()-start,2)) + " seconds. (" \
47            + str(round(round(time.time()-start,2)/len(dataframe),2)) +
              ↪  "s/frame)")
48
49      model2_time = round(time.time()-start,2)
```

```
50
51      return dataframe.copy(), model2_time
```

Object Detection (OD)

```
1   def OD(dataframe,probability_threshold=0.1):
2
3       """ This function takes a dataframe and the column name of the
        ↪  frames as input and outputs the same dataframe """
4       """                          extended with the object detection
        ↪  prediction from the                                    """
5       """
        ↪  'https://tfhub.dev/google/openimages_v4/ssd/mobilenet_v2/1'
        ↪  model.                           """
6
7       print("Detecting objects ...")
8       # Start timer
9       start=time.time()
10
11      # Convert frames to tensors
12      converted = [tf.image.convert_image_dtype(tf.constant(frame),
        ↪  tf.float32)[tf.newaxis, ...] for frame in dataframe["Frame"]]
13
14      # Detect objects
15      prediction = [detector(image) for image in converted]
16      results = [{key: value.numpy() for key, value in part.items()}
        ↪  for part in prediction]
17
18      # Extract unique classes with their highest probability above the
        ↪  threshold
19      filtered_labels = [
20          [
21              tf.compat.as_str_any(class_entity).lower()
22              for class_entity in
                ↪  set(result["detection_class_entities"])
23              if max(score for class_entity_iter, score in
                ↪  zip(result["detection_class_entities"],
                ↪  result["detection_scores"]) if class_entity_iter ==
                ↪  class_entity) > probability_threshold
24          ]
25          for result in results
26      ]
27
28      # Add filtered labels to the DataFrame
29      dataframe["OD"] = filtered_labels
30
31      # Print amount of predicted captions and duration of prediction
32      print(str(len(dataframe)) + " frames were analyzed in " +
        ↪  str(round(time.time()-start,2)) + " seconds. (" \
```

```
33          + str(round(round(time.time()-start,2)/len(dataframe),2)) +
            ↪ "s/frame)")
34      model3_time = round(time.time()-start,2)
35
36      return dataframe.copy(), model3_time
```

Multilabel Classification (MC)

```
1   def preprocess_image(image, target_size=(224, 224)):
2       # Resize the image
3       resized_image = tf.image.resize(image, target_size)
4
5       # Normalize the pixel values to the range [0, 1]
6       normalized_image = tf.cast(resized_image, tf.float32) / 255.0
7
8       # Add a batch dimension
9       input_image = normalized_image[tf.newaxis, ...]
10
11      return input_image
12
13  def MC(dataframe, probability_threshold=0.01):
14
15      """
16      This function takes a dataframe and the column name of the frames
    ↪   as input and outputs the same dataframe extended with the
    ↪   multilabel classifications from the 'https://tfhub.dev/google/
    ↪   openimages_v4/ssd/mobilenet_v2/1' model.
17      """
18
19      # Print message indicating that label classification has started
        ↪   and record starting time
20      print("Classifying Labels ...")
21      start = time.time()
22
23      # Retrieve list of classes from a text file hosted on a remote
        ↪   server and format class names
24      lines = requests.get('https://storage.googleapis.com/
        ↪   bit_models/imagenet21k_wordnet_lemmas.txt').text.split("\n")
25      classes = [entity.split(",")[0].replace("_", " ") for entity in
        ↪   lines][:-1]
26
27      # Preprocess each frame in the input dataframe and obtain
        ↪   predictions from a TensorFlow model
28      converted = [preprocess_image(frame, target_size=(224, 224)) for
        ↪   frame in dataframe["Frame"]]
29      prediction = [tf.nn.sigmoid(module(image)) for image in
        ↪   converted]
30
31      classifications = []
```

```
32
33      # Iterate over predictions and identify the top classes with
        ↪ probabilities above the threshold
34      for probabilities in prediction:
35          top_x = []
36          prob_list = probabilities.numpy()[0].tolist()
37          sorted_results = pd.DataFrame({"classes": classes,
            ↪ "probabilities": prob_list}).sort_values("probabilities",
            ↪ ascending=False)
38
39          for index, row in sorted_results.iterrows():
40              if row["probabilities"] > probability_threshold:
41                  top_x.append(row["classes"])
42
43          classifications.append(top_x)
44
45      # Add new column to the dataframe with the predicted classes for
        ↪ each frame
46      dataframe["MC"] = classifications
47
48      # Print the number of frames classified and the duration of the
        ↪ classification process
49      print(str(len(classifications)) + " frames were classified in " +
        ↪ str(round(time.time()-start,2)) + " seconds. (" \
50            + str(round(round(time.time()-start,2)/len(dataframe),2)) +
              ↪ "s/frame)")
51
52      # Calculate the total time taken for classification and return a
        ↪ copy of the modified dataframe and the time taken
53      model4_time = round(time.time()-start,2)
54
55      return dataframe.copy(), model4_time
```

Scene Recognition

```
1   def load_labels():
2
3       """
4       The load_labels function reads the category labels from the
    ↪   Places365 dataset and returns them in the form of tuples. It also
    ↪   reads the indoor-outdoor labels, scene categories, and wideresnet
    ↪   model.
5       """
6
7       # 1. Prepare all the labels for scene category
8       file_name_category = 'categories_places365.txt'
9       classes = list()
10
11      # Read the category labels from file
```

```
12      with open(file_name_category) as class_file:
13
14          # For each line in the file, extract the label name and add
            ↪  to the list
15          for line in class_file:
16              classes.append(line.strip().split(' ')[0][3:])
17
18      # Convert the list to a tuple for immutability. By converting the
        ↪  classes list to a tuple, the code ensures that the labels
        ↪  cannot be accidentally modified or changed by other parts of
        ↪  the program.
19      classes = tuple(classes)
20
21      # 2. Prepare all the labels for indoor-outdoor
22      file_name_IO = 'IO_places365.txt'
23
24      # Read the indoor-outdoor labels from the labels file
25      with open(file_name_IO) as f:
26          lines = f.readlines()
27          labels_IO = []
28
29          # For each line in the file, extract the label name and add
            ↪  it to the list.
30          for line in lines:
31              items = line.rstrip().split()
32              # subtract 1 to map 1 to 0 and 2 to 1. For the final
                ↪  labels 0 is indoor, and 1 is outdoor.
33              labels_IO.append(int(items[-1]) -1)
34
35      # Convert the list to numpy array for efficiency
36      labels_IO = np.array(labels_IO)
37
38      return classes, labels_IO
39
40
41  def returnTF():
42
43      """
44      The returnTF function loads the image transformer, which applies
    ↪   a set of transformations (resize, convert to tensor, and
    ↪   normalize) to an image. The returnTF function uses the
    ↪   trn.Compose method from the torchvision.transforms module to
    ↪   define a set of transformations that will be applied to an
    ↪   image.
        """
45
46
47      # Load the image transformer and define a set of transformations
        ↪  to be applied to an image
```

```
48      TF = trn.Compose([
49
50          # Resize the image to (224, 224)
51          trn.Resize((224,224)),
52
53          # Convert the image to a PyTorch tensor
54          trn.ToTensor(),
55
56          # Normalize the image with mean and standard deviation
57          trn.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
58      ])
59
60      # Return the image transformer
61      return TF
62
63  def load_model():
64
65      """
66      The load_model function loads the pre-trained model, the weights,
    ↪   and hacks the model to handle batch normalization, average
    ↪   pooling, and other module issues in pytorch1.x.
67      """
68
69      # Download the model file
70      arch = 'resnet18'
71      model_file = f'{arch}_places365.pth.tar'
72
73      # import the model architecture
74      model = models.__dict__[arch](num_classes=365)
75      checkpoint = torch.load(model_file, map_location=lambda storage,
        ↪   loc: storage)
76      state_dict = {str.replace(k,'module.',''): v for k,v in
        ↪   checkpoint['state_dict'].items()}
77      model.load_state_dict(state_dict)
78
79      # set the model to evaluation mode
80      model.eval()
81
82      # register a forward hook on the last convolutional layer
        ↪   (layer4) and the average pooling layer (avgpool) of the
        ↪   model
83      features_names = ['layer4','avgpool']
84
85      return model
86
87
88  # Load all the labels we need for the scene recognition.
89  classes, labels_IO = load_labels()
```

```
90
91  # Create an empty list to store the features extracted from the input
    ↪  image and call it features_blobs
92  features_blobs = []
93
94  #Load the model
95  places_model = load_model()
96  #Load the image transformer
97  TF = returnTF() # image transformer
98  # Retrieve the parameters of the pre-trained model.
99  params = list(places_model.parameters())
100 # Retrieve the softmax weights of the pre-trained model.
101 weight_softmax = params[-2].data.numpy()
102 # This sets all negative values in the softmax weight to zero.
103 weight_softmax[weight_softmax<0] = 0
104
105 def SR(dataframe, frame_column_name="Frame"):
106     """
107     This function runs the places365 model on the videos.
108     Input: video number
109
110     PUT IN BY MARIN --> KONRAD CHECK
111
112     """
113
114     print("Predicting scenes ...")
115     # Start timer
116     start = time.time()
117
118     # Create empty list for predictions
119     places = []
120     all_scenes = []
121
122     for frame in dataframe[frame_column_name]:
123
124         # Create an empty list for all predictions
125         prediction = []
126
127         # Convert array to PIL format
128         img = Image.fromarray(frame)
129
130         # Pre-process the input image using the image transformer and
            ↪  convert it into a PyTorch variable.
131         input_img = V(TF(img).unsqueeze(0))
132
133         # Perform a forward pass through the pre-trained model to
            ↪  obtain the logit values.
134         logit = places_model.forward(input_img)
```

```
135
136             # Apply the softmax function to the logit values and remove
                ↪  the dimensions of size 1.
137             h_x = F.softmax(logit, 1).data.squeeze()
138
139             # This sorts the probabilities and the corresponding class
                ↪  indices in descending order.
140             probs, idx = h_x.sort(0, True)
141
142             # This converts the probabilities to a NumPy array
143             probs = probs.numpy()
144
145             # This converts the class indices to a NumPy array
146             idx = idx.numpy()
147
148             # This calculates the probability of the input image being
                ↪  indoors or outdoors
149             io_image = np.mean(labels_IO[idx[:10]])
150             prediction.append(io_image)
151             places.append(prediction[0])
152
153             # Create an empty list to store the scene categories which
                ↪  the model predicts
154             scene_categories = []
155
156             # We say we maximum want 5 scene categories (if they all meet
                ↪  the threshold)
157             for i in range(0, 5):
158
159                 # Define a variable for the classes (scene categories)
                    ↪  and for the probabilities of these scene categories.
160                 category = classes[idx[i]]
161                 probability = probs[i]
162
163                 # If the probability is higher than 60%, then add the
                    ↪  scene categorie to the list. If not, go on to the
                    ↪  next frame.
164                 if probability >= 0.5:
165                     scene_categories.append(category)
166                 else:
167                     continue
168
169             # Add the scene categories per frame to the list that will
                ↪  form the column in the next step
170             all_scenes.append(scene_categories)
171
172     dataframe["SR: Score"] = places
173     dataframe["SR: Scene Categories"] = all_scenes
```

```
174
175     # Print the number of frames classified and the duration of the
        ↪ classification process
176     print(str(len(all_scenes)) + " scenes were identified in " +
        ↪ str(round(time.time()-start,2)) + " seconds. (" \
177           + str(round(round(time.time()-start,2)/len(dataframe),2)) +
              ↪ "s/frame)")
178
179     model5_time = round(time.time()-start,2)
180
181     return dataframe.copy(), model5_time
```

Night Recognition

```
1   def NR(dataframe):
2
3       """
4       This function takes a dataframe as input where the values of the
    ↪   frames are represented in arrays. It accesses the arrays and
    ↪   calculates the average pixel value per frame. (Later, in the
    ↪   summarization it will be cehecked if the avg pixel value is below
    ↪   a certain previously defined threshold, and if yes, it determines
    ↪   the frame must be a Natopt). It outputs a new dataframe with the
    ↪   timestamp, frames, and a new column with the average pixel
    ↪   values.
5       """
6
7       print("Predicting time of day ...")
8       # Start timer
9       start = time.time()
10
11      # Create an empty list for all predictions
12      day_night = []
13
14      for frame in dataframe["Frame"]:
15
16          # Create empty list for avgerage pixel value
17          prediction_day_night = []
18
19          # Calculate the average pixel value
20          average_pixel_value = np.mean(frame)
21
22          day_night.append(average_pixel_value)
23
24      dataframe["NR"] = day_night
25
26      # Print the number of frames classified and the duration of the
        ↪ classification process
```

```
27      print(str(len(day_night)) + " scenes were analyzed in " +
        ↪  str(round(time.time()-start,2)) + " seconds. (" \
28            + str(round(round(time.time()-start,2)/len(dataframe),2)) +
              ↪  "s/frame)")
29
30      model6_time = round(time.time()-start,2)
31
32      return dataframe.copy(), model6_time
```

Text Recognition

```
1   def TR(dataframe):
2
3       print("Detecting text ...")
4       # Start timer
5       start = time.time()
6
7       # Create empty list for the detected text
8       tr_results = []
9
10      # Apply paddleocr model to recognize text from every frame
11      for frame in dataframe["Frame"]:
12
13          text = paddle_ocr_model.ocr(frame)[0]
14
15          if text:
16
17              # Extract text, remove one-letter-words and append to
                ↪  results
18              tr_results.append([word[1][0] for word in text if
                ↪  len(word[1][0]) > 1])
19
20          else:
21              tr_results.append("")
22
23      # Named Entity Recognition
24      list_ner = [ner(part) if part != "" and part != [] else [] for
        ↪  part in tr_results]
25
26      ner_results = []
27      for part in list_ner:
28
29          # Create an empty list to possibly add no, one or several
            ↪  recognized entities per frame
30          interim = []
31          for i in range(len(part)):
32
33              # If an entity was recognized and it belongs to the LOC
                ↪  or ORG group, add it to the empty interim list
```

```
            if part[i] != [] and part[i][0]['entity_group'] in
            ↪  ["LOC","ORG"]:
                interim.append(part[i][0]['word'])

        # Add the interim list to the ner_results as list if it
        ↪  contains NERs, otherwise append an empty string
        if interim:
            ner_results.append(interim)
        else:
            ner_results.append("")

    # Add the TR and NER results to a new column
    dataframe["NER"] = ner_results
    dataframe["TR"] = tr_results

    # Print amount of images with text and duration of prediction
    print(str(len(ner_results)) + " frames were analysed in " +
    ↪  str(round(time.time()-start,2)) + " seconds. (" \
          + str(round(round(time.time()-start,2)/len(dataframe),2)) +
          ↪  "s/frame)")

    model7_time = round(time.time()-start,2)

    return dataframe.copy(), model7_time
```

F.1.4 Summarization

```
def summarize_predictions(prediction_dataframe,metadata_dataframe,
↪  video_id=False,words_amount=10,threshold=0.1):

    """
    This function takes two dataframes and a word amount as arguments
↪   outputs an updated metadata dataframe containing the summarized
↪   predictions from all different models. The "words_amount"
↪   argument defines how many words should maximally be in the output
↪   and the threshold reduces the output further to only outputting
↪   words if they were predicted for x% of the frames.
    """

    # Save dataframe of evaluation results as csv
    if video_id != False:
        save = prediction_dataframe.copy()
        del save["Frame"]
        save.to_csv(video_id+"_Predictions.csv",index=False)
        print("Dataframe was successfully saved as '" + video_id +
        ↪  "_Predictions.csv'")

    print("Summarizing Predictions...")
```

```
15      # Start timer
16      start_timer = time.time()
17
18      # Convert Timestamps of Predictions to seconds
19      prediction_timestamp_seconds = [(datetime.datetime.strptime(
        ↪  prediction_dataframe.iloc[row]["Timestamp"],
        ↪  "%H:%M:%S")-datetime.datetime(1900,1,1)).total_seconds() for
        ↪  row in range(len(prediction_dataframe))]
20
21      # Convert Start and End Times of Metadata to Scene Ranges
22      scene_ranges = [range(int((datetime.datetime.strptime(
        ↪  metadata_dataframe.iloc[row]["Start-Time"],"%H:%M:%S") -
        ↪  datetime.datetime(1900,1,1)).total_seconds()),
        ↪  int((datetime.datetime.strptime(
        ↪  metadata_dataframe.iloc[row]["End-Time"], "%H:%M:%S") -
        ↪  datetime.datetime(1900,1,1)).total_seconds()+1)) for row in
        ↪  range(len(metadata_dataframe))]
23
24      # Summarize for every model separately
25      for model_column in prediction_dataframe.columns:
26
27          # A) The Frame Descriptions will be summarized in the same
            ↪  way
28          if model_column in ["IC","IS","OD","MC"]:
29
30              # Create empty list for all summarizations
31              prediction_summarized = []
32
33              # Summarize per scene from metadata extract
34              for scene in scene_ranges:
35
36                  # Create empty list for collecting all predicted
                    ↪  keywords and counter for threshold computation
37                  keywords = []
38                  counter = 0
39
40                  # Check for every frame if it was in the scene and if
                    ↪  so, add its keywords to the pool
41                  for timestamp in range(len(prediction_dataframe)):
42
43                      # Check if a specific frame belongs to a scene
44                      if prediction_timestamp_seconds[timestamp] in
                        ↪  scene:
45
46                          # If Duplicates were removed their
                            ↪  predictions get higher weight (by their
                            ↪  occurrence)
47                          if "Counter" in prediction_dataframe.columns:
```

```
48                              keywords +=
                                ↪ prediction_dataframe[model_column]
                                ↪ [timestamp] *
                                ↪ prediction_dataframe["Counter"]
                                ↪ [timestamp]
49                              counter += 1 *
                                ↪ prediction_dataframe["Counter"]
                                ↪ [timestamp]
50
51                          # If duplicates were not removed no occurence
                            ↪ will be considered for the summarization
52                          else:
53                              keywords +=
                                ↪ prediction_dataframe[model_column]
                                ↪ [timestamp]
54                              counter += 1
55
56                  # Adds n(= minimum of all words and "words_amount")
                    ↪ words
57                  # if they occured more often than a predefined
                    ↪ threshold
58                  prediction_summarized.append(',
                    ↪ '.join([Counter(keywords).most_common(
                    ↪ min(len(Counter(keywords)) ,
                    ↪ words_amount))[word][0] for word in
                    ↪ range(min(len(Counter(keywords)) , words_amount))
                    ↪ if Counter(keywords).most_common(
                    ↪ min(len(Counter(keywords)) ,
                    ↪ words_amount))[word][1]/counter > threshold]))
59
60              # Create a new column in the metadata dataframe with the
                ↪ summarized predictions for one model
61              metadata_dataframe[model_column] = prediction_summarized
62
63          # B) Scene Recognition - Indoor/Outdoor detection
64          if model_column == "SR: Score":
65
66              # Create empty list for the end result
67              int_ext = []
68
69              # Summarize per scene from metadata extract.
70              for scene in scene_ranges:
71
72                  # Create empty list for collecting the i/o score for
                    ↪ each frame
73                  score = []
74
75                  # For each timestamp within a scene:
```

```
76                  for timestamp in range(len(prediction_dataframe)):
77
78                      # If the timestamp belongs to the scene, add its
                        ↪  score to the score list for this particular
                        ↪  scene.
79                      if prediction_timestamp_seconds[timestamp] in
                        ↪  scene:
80
81                          # If Duplicates were removed their
                            ↪  predictions/scores get higher weight (by
                            ↪  their occurrence)
82                          if "Counter" in prediction_dataframe.columns:
83                              for duplicate in
                                ↪  range(prediction_dataframe["Counter"]
                                ↪  [timestamp]):
84                                  score.append(
                                    ↪  prediction_dataframe[model_column]
                                    ↪  [timestamp])
85
86                          # If duplicates were not removed no occurence
                            ↪  will be considered for the summarization
87                          else:
88                              score.append(
                                ↪  prediction_dataframe[model_column]
                                ↪  [timestamp])
89
90                  # Calculate average of the score list to find the
                    ↪  score for the overall scene.
91                  average_score = np.mean(score)
92
93                  # Set the thresholds for when we want to see which
                    ↪  output for a scene
94                  if average_score >= 0 and average_score <= 0.28:
95                      int_ext.append('Int. opt.')
96                  elif average_score > 0.7 and average_score <= 1.0:
97                      int_ext.append('Ext. opt.')
98                  else:
99                      int_ext.append("")
100
101             # Later, the int_ext list will be added to the dataframe
                ↪  together with the scene categories list
102
103         # Now we move on the the scene categories part of the
            ↪  places365 model
104         elif model_column == "SR: Scene Categories":
105
106             # Create an empty list which will later be the foundation
                ↪  for a new column in the dataframe
```

```
107             location  = []
108
109             # Summarize per scene from metadata extract
110             for scene in scene_ranges:
111
112                 # create empty list for collecting all scene
                    ↪ categories
113                 attributes = []
114                 counter = 0
115
116                 # For each timestamp in a scene, check if its part of
                    ↪ the scene and then:
117                 for timestamp in range(len(prediction_dataframe)):
118                     if prediction_timestamp_seconds[timestamp] in
                        ↪ scene:
119
120                         # If Duplicates were removed their
                            ↪ predictions get higher weight (by their
                            ↪ occurrence)
121                         if "Counter" in prediction_dataframe.columns:
122                             attributes +=
                                ↪ prediction_dataframe[model_column]
                                ↪ [timestamp] *
                                ↪ prediction_dataframe["Counter"]
                                ↪ [timestamp]
123                             counter += 1 *
                                ↪ prediction_dataframe["Counter"]
                                ↪ [timestamp]
124
125                         # If duplicates were not removed no occurence
                            ↪ will be considered for the summarization
126                         else:
127                             attributes +=
                                ↪ prediction_dataframe[model_column]
                                ↪ [timestamp]
128                             counter += 1
129
130                 # Add the end the division is compared to a threshold
                    ↪ 0.5 (hardcoded since by default our threshold is
                    ↪ 0.1)
131                 # But for this specific part of the model we want a
                    ↪ threshold of 0.5). We explain in the thesis how
                    ↪ we got
132                 # to this number.
```

```
133                 location.append(',
                    ↪  '.join([Counter(attributes).most_common(min(
                    ↪  len(Counter(attributes)) ,
                    ↪  words_amount))[word][0].replace("_"," ") for word
                    ↪  in range(min(len(Counter(attributes)) ,
                    ↪  words_amount)) if
                    ↪  Counter(attributes).most_common(min(
                    ↪  len(Counter(attributes)) ,
                    ↪  words_amount))[word][1] / counter > threshold]))
134
135             # Later, the scene categories list will be added to the
                ↪  dataframe together with the int_ext list
136
137         # C) Night Recognition
138         elif model_column == "NR":
139
140             # Create empty list for all summarizations
141             day_night = []
142
143             # Summarize per scene from metadata extract
144             for scene in scene_ranges:
145
146                 # Create empty list for collecting all predicted
                    ↪  keywords and counter for threshold computation
147                 pixel = []
148
149                 # For each timestamp within a scene:
150                 for timestamp in range(len(prediction_dataframe)):
151                     if prediction_timestamp_seconds[timestamp] in
                        ↪  scene:
152
153                         # If Duplicates were removed their
                            ↪  predictions/scores get higher weight (by
                            ↪  their occurrence)
154                         if "Counter" in prediction_dataframe.columns:
155                             for duplicate in
                                ↪  range(prediction_dataframe["Counter"]
                                ↪  [timestamp]):
156                                 pixel.append(
                                    ↪  prediction_dataframe[model_column]
                                    ↪  [timestamp])
157
158                         # If duplicates were not removed no occurence
                            ↪  will be considered for the summarization
159                         else:
160                             pixel.append(
                                ↪  prediction_dataframe[model_column]
                                ↪  [timestamp])
```

```
161
162                 # Calculate the average pixel value per scene
163                 average_pixel_values = np.mean(pixel)
164
165                 # Set thresholds for natopt.
166                 if average_pixel_values <= 54.15:
167                     day_night.append('Natopt.')
168                 else:
169                     day_night.append("")
170
171             # Create a new column in the metadata dataframe with the
                ↪  summarized predictions for one model
172             metadata_dataframe[model_column] = day_night
173
174         # D) Text Recognition
175         elif model_column == "NER":
176
177             # Create empty list for all summarizations which will be
                ↪  the foundations of the new column in the dataframe
178             prediction_summarized_ner = []
179
180             # Loop through all scenes in a video
181             for scene in scene_ranges:
182
183                 # Create necessary empty lists to collect results
184                 text_ner = []
185
186                 # Go through each frame (per scene)
187                 for timestamp in range(len(prediction_dataframe)):
188                     if prediction_timestamp_seconds[timestamp] in
                        ↪  scene:
189
190                         # Duplicate Removals are not considered as
                            ↪  all named entities will be extracted
                            ↪  either way
191                         if prediction_dataframe[model_column]
                            ↪  [timestamp] != "":
192                             text_ner +=
                                ↪  prediction_dataframe[model_column]
                                ↪  [timestamp]
193
194                 # Add the unique results to the predefined list
195                 prediction_summarized_ner.append(', '.join([word[0]
                    ↪  for word in Counter(text_ner).most_common()]))
196
197                 # Later, the prediction_summarized_ner list will be
                    ↪  added to the
```

```
198                 # dataframe together with the
                    ↪ prediction_summarized_no_ner list
199
200             elif model_column == "TR":
201                 prediction_summarized_no_ner = []
202
203                 # Loop through all scenes in a video
204                 for scene in scene_ranges:
205
206                     # Create necessary empty lists to collect results
207                     text_no_ner = []
208                     counter = 0
209
210                     # Go through each frame (per scene)
211                     for timestamp in range(len(prediction_dataframe)):
212                         if prediction_timestamp_seconds[timestamp] in
                            ↪ scene:
213
214                             # Remove all words from prediction which were
                                ↪ already recognized as NERs to avoid
                                ↪ duplicates
215                             cleaned_tr = [phrase for phrase in
                                ↪ prediction_dataframe[model_column]
                                ↪ [timestamp] if phrase not in
                                ↪ prediction_summarized_ner
                                ↪ [scene_ranges.index(scene)]]
216
217                             # If Duplicates were removed their
                                ↪ predictions get higher weight (by their
                                ↪ occurrence)
218                             if "Counter" in prediction_dataframe.columns:
219                                 text_no_ner += cleaned_tr *
                                    ↪ prediction_dataframe["Counter"]
                                    ↪ [timestamp]
220
221                             # If duplicates were not removed no occurence
                                ↪ will be considered for the summarization
222                             else:
223
224                                 if cleaned_tr != "":
225                                     text_no_ner += cleaned_tr
226
227                     # For the rest of the text that was not detected by
                        ↪ NER, sort words (descending) by their frequency
                        ↪ and add
228                     # them (uniquely) to the list. Here, no further
                        ↪ threshold will be applied (since the word count
                        ↪ filter will
```

```
229                 # be applied in a later step for the whole 'Text
                    ↪ Recognition' column).
230                 prediction_summarized_no_ner.append(',
                    ↪ '.join([word[0] for word in
                    ↪ Counter(text_no_ner).most_common()]))
231
232     # Merge the two text recognition lists together and add them to
        ↪ the dataframe (if both exist, connect them with a comma)
233     metadata_dataframe['TR'] = [prediction_summarized_ner[i] + ', ' +
        ↪ prediction_summarized_no_ner[i] \
234                                 if prediction_summarized_ner[i] and
                                    ↪ prediction_summarized_no_ner[i] \
235                                 else prediction_summarized_ner[i] +
                                    ↪ prediction_summarized_no_ner[i] \
236                                 for i in
                                    ↪ range(len(metadata_dataframe))]
237
238     # Merge the two scene recognition lists together and add them to
        ↪ the dataframe (if both exist, connect them with a space)
239     metadata_dataframe['SR'] = [int_ext[i] + ' ' + location[i] if
        ↪ int_ext[i] and location[i] \
240                                 else int_ext[i] + location[i] for i
                                    ↪ in
                                    ↪ range(len(metadata_dataframe))]
241
242     # Filter all summarizations for the first x words:
243     def get_first_x_words(text, x):
244         return ', '.join(text.split(',')[:x])
245     # 1) IC: x words:
246     metadata_dataframe['IC'] =
        ↪ metadata_dataframe['IC'].str.strip().apply(get_first_x_words,
        ↪ x=6)
247     # 2) IS: x words:
248     metadata_dataframe['IS'] =
        ↪ metadata_dataframe['IS'].str.strip().apply(get_first_x_words,
        ↪ x=6)
249     # 3) OD: x words:
250     metadata_dataframe['OD'] =
        ↪ metadata_dataframe['OD'].str.strip().apply(get_first_x_words,
        ↪ x=4)
251     # 4) MC: 4 words:
252     metadata_dataframe['MC'] =
        ↪ metadata_dataframe['MC'].str.strip().apply(get_first_x_words,
        ↪ x=4)
253     # 5) SR: Since Scene Categories are already strongly restricted,
        ↪ no further filters will be applied
254     # 6) NR: No further filters needed
255     # 7) TR: 10 words:
```

```
256     metadata_dataframe['TR'] =
        ↪ metadata_dataframe['TR'].str.strip().apply(get_first_x_words,
        ↪ x=10)
257
258     # Print duration of summarization
259     print("Summarization of " + str(len(scene_ranges)) + " Scenes for
        ↪ " + str(len(["yes" for column in
        ↪ list(prediction_dataframe.columns) if "Model" in column])) +
        ↪ " Models took " + str(round(time.time()-start_timer,2)) + "
        ↪ seconds. (" + str(round(round(time.time() - start_timer,2) /
        ↪ len(prediction_dataframe) , 2)) + "s/frame)")
260
261     # Add a new column to the dataframe with the video id
262     metadata_dataframe.insert(0,"Video ID",video_id)
263
264     # Save dataframe of evaluation results as csv
265     metadata_dataframe.to_csv(video_id+"_Summarizations.csv",
        ↪ index=False)
266     print("Dataframe was successfully saved as '" + video_id +
        ↪ "_Summarizations.csv'")
267
268     return metadata_dataframe.copy()
```

F.2 Master File

```
1  import Pipeline
2  import pandas as pd
3  import time
4  import os
5
6  # Define video and metadata paths
7  data_path = "D:/Thesis/Master Thesis Scripts/00_Data"
8  metadata_path = data_path + "/metadata"
9  video_path = data_path + "/video"
10
11 # Collect Video IDs
12 video_ids = os.listdir("D:/Thesis/Master Thesis
   ↪ Scripts/00_Data/metadata")
13 video_ids = [id.split(".")[0] for id in video_ids]
14
15 # Create several empty lists for counts
16 time_VID = []
17 time_IC = []
18 time_IS = []
19 time_OD = []
20 time_MC = []
21 time_SR = []
22 time_NR = []
```

```
time_TR = []
time_amount = []
time_unique = []

def loop(video_id):

    model_start = time.time()

    # Extract metadata
    metadata = Pipeline.extract_metadata(metadata_path,
    ↪  video_id,evaluation=True)

    # Extract Frames
    frames, frame_amount = Pipeline.extract_frames(video_path,
    ↪  video_id,metadata,evaluation=True)

    # Duplicate Removal
    frames, unique_amount =
    ↪  Pipeline.drop_duplicates(frames,mse_threshold=0)

    # IC (Image Captioning)
    frames, IC_time = Pipeline.IC(frames)

    # IS (Image Segmentation)
    frames, IS_time = Pipeline.IS(frames)

    # OD (Object Detection)
    frames, OD_time = Pipeline.OD(frames)

    # MC (Multilabel Classification)
    rames, MC_time = Pipeline.MC(frames)

    # SR (Scene Recognition)
    frames, SR_time = Pipeline.SR(frames)

    # NR (Night Recognition)
    frames, NR_time = Pipeline.NR(frames)

    # TR (Text Recognition)
    frames, TR_time = Pipeline.TR(frames)

    # Summarization of Predictions
    summarization = Pipeline.summarize_predictions(frames,metadata)

    # Append all video-related information to lists
    time_VID.append(video_id)
    time_IC.append(IC_time)
    time_IS.append(IS_time)
```

```
68      time_OD.append(OD_time)
69      time_MC.append(MC_time)
70      time_SR.append(SR_time)
71      time_NR.append(NR_time)
72      time_TR.append(TR_time)
73      time_amount.append(frame_amount)
74      time_unique.append(unique_amount)
75
76      print("Run for video '" + str(video_id) + "' took " +
        ↪  str(round(time.time()-model_start,2)) + " seconds.")
77
78  # Create an empty dataframe for storing all time dfs later
79  total_time = pd.DataFrame()
80
81  loop_start = time.time()
82  # Run all models on all videos
83  for video in video_ids:
84
85      # Run all models on one video
86      loop(video)
87
88      # Extract all model times from video and save to time_df
89      time_df = pd.DataFrame({"Video ID":time_VID, "IC":time_IC,
        ↪  "IS":time_IS, "OD":time_OD, "MC":time_MC, "SR":time_SR,
        ↪  "NR":time_NR, "TR":time_TR, "Frames":time_amount, "Unique
        ↪  Frames":unique_amount})
90
91      # Concatenate total_time df and time_df of new video
92      total_time = pd.concat([total_time,time_df])
93
94      # Save Runtime dataframe as csv
95      time_df.to_csv("Runtime.csv",index=False)
96
97  print("Loop through all videos took " +
    ↪  str(round(time.time()-loop_start,2)) + " seconds.")
```

F.3 Word Count Delimitation Tests

```
1  # General
2  import pandas as pd
3  import os
4  import time
5  import numpy as np
6  import matplotlib.pyplot as plt
7
8  # Evaluation
9  from sentence_transformers import SentenceTransformer,util
```

```
sentence_transformer = SentenceTransformer('sentence-transformers/all-mpnet-base-v2', device="cpu")

def evaluate_predictions(dataframe):

    """
    This function takes one dataframe and a video id as input and adds quality scores (of the predictions to the ground truth) to the output dataframe. Also, it saves the resulting dataframe as csv in the following format: "{video_id}_Results.csv"
    """

    print("Evaluating Predictions...")
    # Start timer
    start_timer = time.time()

    # Create list and dictionary with empty lists as values of relevant columns
    list_of_models = [column for column in dataframe.columns if column[:2] in ["IC","IS","OD","MC","SR","NR","TR"]]
    dict_of_models = {column:[] for column in dataframe.columns if column[:2] in ["IC","IS","OD","MC","SR","NR","TR"]}

    for scene in range(len(dataframe)):

        # Compute the Embedding for the Ground Truth
        ground_truth_embedding = sentence_transformer.encode(dataframe["Ground Truth Description"][scene])

        # Compute Embeddings for all Predictions
        predictions_embedding = sentence_transformer.encode([dataframe[column][scene] for column in list_of_models])

        # Compute the Dot Score of every Prediction to the Ground Truth
        scores = util.dot_score(ground_truth_embedding, predictions_embedding)[0].tolist()

        # Append Scores to Model List
        for model_name in range(len(list_of_models)):
            # Avoid unexpected high evaluation scores in case of no prediction
            if dataframe[list_of_models[model_name]][scene] == "":
                dict_of_models[list_of_models[model_name]].append(0)
            else:
```

```
44                  # Only add absolute values as negative equal to
                    ↪ positive ones but distort mean calculation
45                  dict_of_models[list_of_models[model_name]]
                    ↪ .append(abs(scores[model_name]))
46
47      # Append every Model Score List as new Column to the given
        ↪ dataframe
48      for item in dict_of_models:
49          dataframe["".join([item.split(":")[0],"_Score"])] =
            ↪ dict_of_models[item]
50
51      # Print duration of summarization
52      print("Evaluation of " + str(len(dataframe)) + " Predictions from
        ↪ " + str(len(list_of_models)) + " different Models took " +
        ↪ str(round(time.time()-start_timer,2)) + " seconds. (" +
        ↪ str(round(round(time.time()-start_timer,2) /
        ↪ len(dataframe),2)) + "s/scene)")
53
54      return dataframe.copy()
55
56  def concat_results(folder_path):
57
58      """
59      This function imports all csv files from a specified folder and
    ↪   concatenates them to one total dataframe.
60      """
61
62      # Create an empty dataframe
63      dataframe = pd.DataFrame()
64
65      # Loop through all results and concatenate them all together to
        ↪ one "results" dataframe
66      for summarization in [item for item in os.listdir(folder_path) if
        ↪ ".csv" in item]:
67          result = pd.read_csv(folder_path+"/"+summarization)
68          dataframe = pd.concat([dataframe,result])
69
70      # Remove scenes where we had no videos
71      to_be_deleted = pd.read_csv("To_Be_Deleted.csv")
72      dataframe = dataframe[~dataframe["Scene
        ↪ ID"].isin(to_be_deleted["Scene ID"])]
73      dataframe.fillna("", inplace=True)
74
75      return dataframe.reset_index(drop=True)
76
77  def plot_optimal_word_count(csv_path):
78
79      """
```

```
80      This function takes a file path to a csv file for word count
    ↪   tests as input and plots their performance.
81      """
82
83      word_count_eval = pd.read_csv(csv_path)
84
85      model_names = [name.split("_")[1] for name in
        ↪  word_count_eval.columns[-20:-10]]
86      model_perfs = [np.mean(word_count_eval[name]) for name in
        ↪  word_count_eval.columns[-10:]]
87
88      plt.plot(model_names,model_perfs)
89
90      # Add a title and axis labels
91      #plt.title('MC Word Count Delimitation')
92      plt.xlabel('Word Count')
93      plt.ylabel('Performance')
94
95      # Add a vertical dashed line at the maximum value of model_perfs
96      max_index = model_perfs.index(max(model_perfs))
97      plt.axvline(x=max_index, linestyle='--', color='red')
98
99      # Show the plot
100     plt.show()
101
102 # Read in all model summarizations
103 IC_summarizations = concat_results("D:/Thesis/Master Thesis
    ↪   Scripts/04_Pipeline/Pipeline/IC/Summarizations")
104 IS_summarizations = concat_results("D:/Thesis/Master Thesis
    ↪   Scripts/04_Pipeline/Pipeline/IS/Summarizations")
105 OD_summarizations = concat_results("D:/Thesis/Master Thesis
    ↪   Scripts/04_Pipeline/Pipeline/OD/Summarizations")
106 MC_summarizations = concat_results("D:/Thesis/Master Thesis
    ↪   Scripts/04_Pipeline/Pipeline/MC/Summarizations")
107 TR_summarizations = concat_results("D:/Thesis/Master Thesis
    ↪   Scripts/04_Pipeline/Pipeline/TR/Summarizations")
108
109 # Merge summarizations to one file and fill NAs
110 summarizations = pd.merge(IC_summarizations,
    ↪   IS_summarizations.drop('Ground Truth Description', axis=1),
    ↪   on=["Video ID", "Scene ID", "Start-Time",
    ↪   "End-Time"]).merge(OD_summarizations.drop('Ground Truth
    ↪   Description', axis=1), on=["Video ID", "Scene ID", "Start-Time",
    ↪   "End-Time"]).merge(MC_summarizations.drop('Ground Truth
    ↪   Description', axis=1), on=["Video ID", "Scene ID", "Start-Time",
    ↪   "End-Time"]).merge(TR_summarizations.drop('Ground Truth
    ↪   Description', axis=1), on=["Video ID", "Scene ID", "Start-Time",
    ↪   "End-Time"])
```

```
summarizations.fillna("", inplace=True)

# Create predictions with word amounts from 1 to 10
models = {}
# For every (frame description) model column ...
for column in summarizations.columns[5:]:

    models[column] =
    ↪ summarizations[summarizations.columns[:5]].copy()

    # and for a word amount ranging from 1 to 10 ...
    for word_count in range(1,11):

        # add a new column to the dataset with the specified amount
        ↪ of words as max
        models[column][column+"_"+str(word_count)] =
        ↪ summarizations[column].apply(lambda x: "
        ↪ ".join(str.split(x,"
        ↪ ")[:min(word_count,len(str.split(x,", ")))]))

# Evaluate predictions with each word count and save each model's
↪ results as csv
for col in models.keys():
    models[col] = evaluate_predictions(models[col])
    models[col].to_csv(col+"_word_count_evaluation.csv")

# Plot all model word count test results
plot_optimal_word_count("IC_word_count_evaluation.csv")
plot_optimal_word_count("IS_word_count_evaluation.csv")
plot_optimal_word_count("OD_word_count_evaluation.csv")
plot_optimal_word_count("OD_word_count_evaluation.csv")
plot_optimal_word_count("TR_word_count_evaluation.csv")

# To check Text Recognition for a word count higher than 10, do the
↪ same again with word counts until 20:

# Read in TR summarizations
TR_summarizations = concat_results("D:/Thesis/Master Thesis
↪ Scripts/04_Pipeline/Pipeline/TR/Summarizations")

summarizations = TR_summarizations.copy()

# Create predictions with word amounts from 1 to 20
models = {}
# For every (frame description) model column ...
for column in summarizations.columns[5:]:

```

```
150     models[column] =
        ↪ summarizations[summarizations.columns[:5]].copy()
151
152     # and for a word amount ranging from 1 to 20 ...
153     for word_count in range(1,21):
154
155         # add a new column to the dataset with the specified amount
            ↪ of words as max
156         models[column][column+"_"+str(word_count)] =
            ↪ summarizations[column].apply(lambda x: ",
            ↪ ".join(str.split(x,",
            ↪ ")[:min(word_count,len(str.split(x,", ")))]))
157
158 # Evaluate predictions and save results as csv
159 models["TR"] = evaluate_predictions(models["TR"])
160 models["TR"].to_csv("TR20_word_count_evaluation.csv")
161
162 # Plot word count test results for TR with up to 20 words
163 plot_optimal_word_count("TR20_word_count_evaluation.csv")
```

F.4 Ensemble Method

```
1  # General
2  import os
3  import time
4  from time import strftime
5  import pandas as pd
6  import numpy as np
7  import math
8
9  # Mixing
10 import itertools
```

F.4.1 Predictions

```
1  def concat_results(folder_path):
2
3      """
4      This function imports all csv files from a specified folder and
       ↪ concatenates them to one total dataframe.
5      """
6
7      # Create an empty dataframe
8      dataframe = pd.DataFrame()
9
10     # Loop through all results and concatenate them all together to
       ↪ one "results" dataframe
```

```
    for summarization in [item for item in os.listdir(folder_path) if
    ↪ ".csv" in item]:
        result = pd.read_csv(folder_path+"/"+summarization)
        dataframe = pd.concat([dataframe,result])

    # Remove scenes where we had no videos
    to_be_deleted = pd.read_csv("To_Be_Deleted.csv")
    dataframe = dataframe[~dataframe["Scene
    ↪ ID"].isin(to_be_deleted["Scene ID"])]
    dataframe.fillna("", inplace=True)

    return dataframe.reset_index(drop=True)

# Apply the Ensemble Method to all MSE Variations
for MSE in [0,250,500,750,1000]:

    # Read in and concatenate all summarizations of all videos and
    ↪ one specific MSE
    summarizations = concat_results("D:/Thesis/Master Thesis
    ↪ Scripts/04_Pipeline/Pipeline/Results_" + str(MSE) +
    ↪ "/Summarizations")

    # Reorder the model columns
    summarizations = summarizations[["Video ID","Scene
    ↪ ID","Start-Time","End-Time","Ground Truth
    ↪ Description","SR","NR","IC","IS","OD","MC","TR"]]

    # Create the ensembles ...
    models = summarizations.columns[5:]

    # ... for ensembles of size 2 to 7
    for model_amount in range(2,8):

        # Determine all model combinations if always picking
        ↪ #model_amount models
        combinations = list(itertools.combinations(models,
        ↪ model_amount))

        for combination in combinations:
            # Combine model results with comma (only if model also
            ↪ has made a prediction)
            summarizations["_".join(combination)] =
            ↪ summarizations[list(combination)].apply(lambda x: ',
            ↪ '.join(filter(lambda v: v and v.strip(), x)), axis=1)

    # Add the MSE value to all columns (for later recognition)
    summarizations.rename(columns={col:col+"_"+str(MSE) for col in
    ↪ summarizations.columns[5:]}, inplace=True)
```

```
46
47      # Save all ensembles for one MSE value as csv file
48      summarizations.to_csv("Ensembles_"+str(MSE)+".csv",index=False)
```

F.4.2 Runtimes

```
1   runtimes = pd.read_csv("Results_0/Runtime_0.csv")
2
3   runtimes = runtimes[["Video
    ↪  ID","SR","NR","IC","IS","OD","MC","TR","Frames", "Unique
    ↪  Frames"]]
4
5   # Apply the Ensemble Method to all MSE Variations
6   for MSE in [0,250,500,750,1000]:
7
8       # Read in runtime df of a specific MSE value
9       runtimes =
        ↪  pd.read_csv("Results_"+str(MSE)+"/Runtime_"+str(MSE)+".csv")
10
11      # Reorder the model columns
12      runtimes = runtimes[["Video
        ↪  ID","SR","NR","IC","IS","OD","MC","TR","Frames","Unique
        ↪  Frames"]]
13
14      # Read out model names
15      models = list(runtimes.columns[1:8])
16
17      # Read out average runtimes of each model
18      runtime = [round(sum(runtimes[model])/sum(runtimes["Frames"]),3)
        ↪  for model in models]
19
20      # Create two ensemble lists
21      ensemble_name = models.copy()
22      ensemble_runtime = runtime.copy()
23
24      # ... for ensembles of size 2 to 7
25      for model_amount in range(2,8):
26
27          # Determine all model combinations if always picking
            ↪  #model_amount models
28          combinations = list(itertools.combinations(models,
            ↪  model_amount))
29
30          for combination in combinations:
31
32              # Append model combination (ensemble) names to ensemble
                ↪  name list
33              ensemble_name.append("_".join(combination))
```

```
34
35              # Append aggregated ensemble runtimes to ensemble runtime
                ↪ list
36              ensemble_runtime.append(round(sum(
                ↪ [runtime[models.index(model)] for model in
                ↪ combination]),4))
37
38      # Append MSE value to ensemble names for later recognizability
39      ensemble_name = [name+"_"+str(MSE) for name in ensemble_name]
40
41      # Convert model/ensemble names and runtimes to dataframe
42      runtime_df =
        ↪ pd.DataFrame({"Method/Ensemble":ensemble_name,"Runtime per
        ↪ Frame":ensemble_runtime})
43
44      # Save model/ensemble runtime df with specific MSE value
45      runtime_df.to_csv("Results_" + str(MSE) +
        ↪ "/Ensembles_Runtime_"+str(MSE)+".csv")
```

F.5 Results Merge & Evaluation

```
1  # General
2  import os
3  import time
4  from time import strftime
5  import pandas as pd
6  import numpy as np
7  from tqdm import tqdm
8  from sentence_transformers import SentenceTransformer,util
9  sentence_transformer =
   ↪ SentenceTransformer('sentence-transformers/all-mpnet-base-v2',
   ↪ device="cpu")
```

F.5.1 Runtime Concatenation

```
1  # Read in all runtime dataframes
2  runtimes_0 = pd.read_csv("Results_0/Ensembles_Runtime_0.csv")
3  runtimes_250 = pd.read_csv("Results_250/Ensembles_Runtime_250.csv")
4  runtimes_500 = pd.read_csv("Results_500/Ensembles_Runtime_500.csv")
5  runtimes_750 = pd.read_csv("Results_750/Ensembles_Runtime_750.csv")
6  runtimes_1000 =
   ↪ pd.read_csv("Results_1000/Ensembles_Runtime_1000.csv")
7
8  # Concatenate runtimes of all models/ensembles and MSE values
9  runtimes =
   ↪ pd.concat([runtimes_0,runtimes_250,runtimes_500,runtimes_750,
   ↪ runtimes_1000]).reset_index(drop=True)
10
```

```
# Save concatenated runtimes
runtimes.to_csv("Runtime_Evaluation.csv")
```

F.5.2 Prediction Evaluation

```
def evaluate_predictions(dataframe):

    """
    This function takes one dataframe as input and adds quality scores
    (of the predictions to the ground truth) to the output dataframe.

    """

    print("Evaluating Predictions...")
    # Start timer
    start_timer = time.time()

    # Create list and dictionary with empty lists as values of relevant columns
    list_of_models = [column for column in dataframe.columns if column[:2] in ["IC","IS","OD","MC","SR","NR","TR"]]
    dict_of_models = {column:[] for column in dataframe.columns if column[:2] in ["IC","IS","OD","MC","SR","NR","TR"]}

    for scene in tqdm(range(len(dataframe))):

        # Compute the Embedding for the Ground Truth
        ground_truth_embedding = sentence_transformer.encode(dataframe["Ground Truth Description"][scene])

        # Compute Embeddings for all Predictions
        predictions_embedding = sentence_transformer.encode([dataframe[column][scene] for column in list_of_models])

        # Compute the Dot Score of every Prediction to the Ground Truth
        scores = util.dot_score(ground_truth_embedding, predictions_embedding)[0].tolist()

        # Append Scores to Model List
        for model_name in range(len(list_of_models)):
            # Avoid unexpected high evaluation scores in case of no prediction
            if dataframe[list_of_models[model_name]][scene] == "":
```

```
                dict_of_models[list_of_models[model_name]].append(0)
            else:
                # Only add absolute values as negative equal to positive ones but distort mean calculation
                dict_of_models[list_of_models[model_name]].append(abs(scores[model_name]))

    # Append every Model Score List as new Column to the given dataframe
    for item in dict_of_models:
        dataframe["".join([item.split(":")[0],"_Score"])] = dict_of_models[item]

    # Print duration of summarization
    print("Evaluation of " + str(len(dataframe)) + " Predictions from " + str(len(list_of_models)) + " different Models took " + str(round(time.time()-start_timer,2)) + " seconds. (" + str(round(round(time.time()-start_timer,2) / len(dataframe),2)) + "s/scene)")

    return dataframe.copy()

# Read in all prediction dataframes and fill empty predictions (NAs) with empty strings
results_0 = pd.read_csv("Results_0/Ensembles_0.csv")
results_0.fillna("", inplace=True)
results_250 = pd.read_csv("Results_250/Ensembles_250.csv")
results_250.fillna("", inplace=True)
results_500 = pd.read_csv("Results_500/Ensembles_500.csv")
results_500.fillna("", inplace=True)
results_750 = pd.read_csv("Results_750/Ensembles_750.csv")
results_750.fillna("", inplace=True)
results_1000 = pd.read_csv("Results_1000/Ensembles_1000.csv")
results_1000.fillna("", inplace=True)

# Merge all predictions of all models/ensembles (for all MSE values)
results = pd.merge(results_0, results_250.drop('Ground Truth Description', axis=1), on=["Video ID", "Scene ID", "Start-Time", "End-Time"]).merge(results_500.drop('Ground Truth Description', axis=1), on=["Video ID", "Scene ID", "Start-Time", "End-Time"]).merge(results_750.drop('Ground Truth Description', axis=1), on=["Video ID", "Scene ID", "Start-Time", "End-Time"]).merge(results_1000.drop('Ground Truth Description', axis=1), on=["Video ID", "Scene ID", "Start-Time", "End-Time"])

# Save merged predictions dataframe
results.to_csv("All_Results.csv")
```

```
# Evaluate all predictions
evaluation = evaluate_predictions(results)

# Extract all model/ensemble names and average scores from evaluation dataframe
methods = [method [:-6] for method in list(results.columns[640:])]
scores = [np.mean(results[method]) for method in results.columns[640:]]

# Convert ensemble names and scores to dataframe
scores_df = pd.DataFrame({"Method/Ensemble":methods,"Scores":scores})

# Save dataframe with all evaluated scores
scores_df.to_csv("Scores_Evaluation.csv",index=False)
```

F.5.3 Runtime & Score Merge

```
# Merge Runtime and Scores dataframes to one final dataframe
total_df = pd.merge(scores_df,runtime_df,on="Method/Ensemble")

# Save final dataframe as csv
total_df.to_csv("Total_Evaluation.csv",index=False)
```

F.6 Correlation Test

```
# General
import os
import time
from time import strftime
import pandas as pd
import seaborn as sns
import numpy as np
import datetime
from collections import Counter

# Read in all evaluations
evaluation = pd.read_csv("All_Results_Evaluated.csv")

# Filter for onlys score columns
new_scores = evaluation[["Scene ID","Start-Time","End-Time","Ground Truth Description"]+[name for name in evaluation.columns if "Score" in name]].copy()

# Calculate scene lengths
```

```
scene_ranges = [int((datetime.datetime.strptime(
    new_scores.iloc[row]["End-Time"], "%H:%M:%S") -
    datetime.datetime(1900,1,1)).total_seconds()+1) -
    int((datetime.datetime.strptime(
    new_scores.iloc[row]["Start-Time"], "%H:%M:%S") -
    datetime.datetime(1900,1,1)).total_seconds()) for row in
    range(len(new_scores))]

# Calculate word amounts
word_amounts = [len(sentence.split(" ")) for sentence in
    new_scores["Ground Truth Description"]]

# Insert both into the existing dataframe
new_scores.insert(1, "Word Amount",word_amounts)
new_scores.insert(1, "Scene Length",scene_ranges)

# Subset the dataframe for relevant columns
subset = pd.DataFrame({"Scene Length":scene_ranges,"Word
    Amount":word_amounts,"Average Score":mean_scores})

# Calculate correlation matrix
corr_matrix = subset.corr()

# Plot correlation matrix
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
```