University of Bergen
Department of Informatics

Automatic blurring of specific faces in video

Author: Erlend Fonnes
Supervisors: Pekka Parviainen, Hjalti Gislason and Kenneth Cuomo

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

June, 2022

Abstract

With the introduction of the General Data Protection Regulation (GDPR) into European Union law, it became more important than ever before to properly handle personal data. This is an issue for media companies which distribute large amounts of media containing identifiable people, which thus may require the subjects' permission for distribution.

In this Master's thesis, I propose a solution which supports and facilitates compliance with GDPR regarding the distribution of video containing identifiable subjects by automatically blurring a select group of people in the videos. The proposed solution is a pipeline for detecting, identifying and blurring select faces, where the video frames are processed like individual images to detect and recognize faces, and the interrelatedness of adjacent frames in continuous videos is exploited to both to improve their prediction quality and running time. Each part of the pipeline is interchangeable and may be replaced individually, and the deployment of the entire pipeline has been automated. Aspects related to video processing, facial detection and facial recognition were explored for this purpose, and various existing tools and solutions were utilized.

Acknowledgements

I would like to thank Pekka Parviainen, my primary supervisor, who helped me throughout the project. I would also like to thank my family, friends and colleagues for their tremendous support.

Erlend Fonnes
Wednesday 29th June, 2022

Contents

1 Introduction 1

1.1 Introductory background . . . 1

1.2 CuttingRoom . . . 3

1.3 Existing solutions . . . 3

1.4 Proposed solution . . . 4

1.5 Solution requirements . . . 4

1.6 Project scope . . . 5

1.7 Report structure . . . 5

2 Theoretical background 7

2.1 Video . . . 7

2.1.1 FFmpeg . . . 7

2.1.2 Smoothing . . . 8

2.1.3 Interpolation . . . 8

2.1.4 Shot transition detection . . . 8

2.2 Facial recognition pipeline . . . 9

2.3 Facial detection . . . 10

2.3.1 Convolutional neural network . . . 10

2.3.2 Histogram of oriented gradients . . . 11

2.4 Facial recognition . . . 12

2.5 Infrastructure . . . 13

2.5.1 Amazon Web Services . . . 13

2.5.2 Terraform . . . 14

2.5.3 Docker . . . 16

3 Implementation 17

3.1 Video . . . 21

3.1.1 Initial processing . . . 21

3.1.2 Processing every frame . . . 22

3.1.3 Dynamic processing 22
3.1.4 Identifying facial sequences 25
3.1.5 Frame interpolation 27
3.1.6 Detecting shot transition 33
3.1.7 Selecting specific faces 34
3.1.8 Smoothing sequences of bounding boxes 39
3.1.9 Handling false negatives 40
3.1.10 Handling false positives 42
3.1.11 Blurring 44
3.1.12 Sequence models 47
3.2 Facial detection, alignment and representation 51
3.2.1 Retinaface 51
3.2.2 Multi-task Cascaded Convolutional Networks 51
3.2.3 Pigo 52
3.2.4 ArcFace 52
3.2.5 Dlib 53
3.2.6 Deepface 54
3.2.7 AWS Rekognition 56
3.3 Orchestration 57
3.3.1 AWS SageMaker Processing Job 57
3.3.2 AWS SageMaker Batch Transform 58
3.3.3 Automatization 61
3.4 Distribution 62
3.4.1 Example configuration 62
3.4.2 Interchangeability 63

4 Analysis **65**
4.1 Prediction quality 65
4.1.1 Prediction quality results 68
4.2 Cost and running time 71
4.2.1 Running time results 72
4.2.2 Cost 75
4.2.3 Scalability 77
4.3 Use of computational resources 78
4.3.1 Optimizing blurring 79
4.3.2 Optimizing the reading of frames 80
4.4 Removing false negatives 82

4.5 Removing false positives 83
4.6 Selecting specific faces for blurring 85
4.6.1 Selecting known faces 85
4.6.2 Selecting by time and location 86
4.7 Detecting shot transitions 87
4.8 Sequence models 89
4.9 AWS Rekognition 91

5 Conclusion **92**

Glossary **95**

List of Acronyms and Abbreviations **96**

Bibliography **97**

A Thesis description **105**

B Prediction quality video analysis numbers **107**

C List of example videos **110**

D Memory usage overview **112**

E Deepface model licenses **114**

List of Figures

2.1 Illustration of a typical convolutional neural network 11

3.1 Overview of proposed pipeline 18
3.2 Illustrated example mapping faces between frames 26
3.3 Illustration of a piecewise interpolation 28
3.4 Linear interpolation 29
3.5 Polynomial interpolation 29
3.6 False positive without apparent reason 42
3.7 Face detected in photography 42
3.8 Square blurring 44
3.9 Pixelated blurring 45
3.10 Blurring mask 45
3.11 Fully blurred image 46
3.12 Round blurring 46

4.1 Interpolated bounding box when a face is out of view 66
4.2 Face on the edge of the screen not detected 67
4.3 Rapid facial movement 67
4.4 Face from the side not detected by Dlib 69
4.5 Detection of different skin tones in poor lighting conditions 70
4.6 Processing time 74
4.7 Profiling visualization 78
4.8 Profiling visualization after optimizing blurring 79
4.9 Memory usage visualization reading all frames once 81
4.10 Profiling visualization after optimizing the reading of frames 81
4.11 False positive and true positive in adjacent frames incorrectly matched . 85
4.12 Selecting known faces 86
4.13 Missing recognized face 86

List of Tables

4.1 Prediction quality: Deepface (processing every frame) 68
4.2 Prediction quality: Deepface (dynamic processing) 68
4.3 Prediction quality: Dlib (processing every frame) 69
4.4 Prediction quality: Dlib (dynamic processing) 69
4.5 Running time: ml.t3.large (processing every frame) 73
4.6 Running time: ml.c5.xlarge (processing every frame) 73
4.7 Running time: ml.g4dn.xlarge (processing every frame) 73
4.8 Running time: ml.t3.large (dynamic processing) 73
4.9 Running time: ml.c5.xlarge (dynamic processing) 73
4.10 Running time: ml.g4dn.xlarge (dynamic processing) 73
4.11 Cost: Processing every frame 75
4.12 Cost: Dynamic processing 75
4.13 Prediction quality without interpolation 82
4.14 Prediction quality filtering shorter facial sequences 84

List of Listings

1 Terraform resource 14
2 Terraform plan 15
3 Bounding box format 19
4 Facial features format 19
5 Facial sequences format 20
6 FFmpeg split video 21
7 FFmpeg combine video 21
8 Dynamic processing with missing match 24
9 Weighted distance metric for comparing two faces 27
10 SciPy interpolators 31
11 Manually implemented linear interpolator 32
12 Parsing known faces 36
13 Filtering known faces 37
14 Filtering selected faces 38
15 Unweighted sliding-average smoothing 40
16 RNN loss function 48
17 Dlib process 54
18 Batch transform handler using ArcFace and MTCNN 60
19 Package installation 62
20 Simple usage example 63
21 Overview of memory usage 112
22 Overview of memory usage reading every frame once 113

Chapter 1

Introduction

1.1 Introductory background

In recent years, the regulations surrounding the handling of personal data has become a lot stricter. The General Data Protection Regulation (GDPR), one of the strictest and impactful privacy regulations, was put into effect on May 25, 2018 [71]. GDPR was drafted and passed by the European Union (EU), and puts significant restrictions on the processing of personal data belonging to EU citizens and residents. Of particular interest, the restrictions apply to pictures where the subjects are identifiable.

> *Although not a member of the EU, Norway is a member of the European Economic Area (EEA*[1]*). The GDPR was incorporated into the EEA agreement and became applicable in Norway on 20 July 2018. Norway is thus bound by the GDPR in the same manner as EU Member States.* (The Norwegian Data Protection Authority [29])

The Personal Data Act[2] is the main set of laws regulating the handling of personal data in Norway, and it incorporates GDPR into Norwegian law and determines how it relates to other parts of Norwegian law [28]. The Norwegian Data Protection Authority, the agency of the Norwegian Government responsible for managing the Personal Data Act and monitoring adherence to it, describes the practical application of the laws relating to the publishing of media containing identifiable subjects in Norway on their official webpage `https://www.datatilsynet.no/`. They distinguish between two types of media [27]:

[1] `https://www.efta.int/eea`
[2] `https://lovdata.no/dokument/NL/lov/2018-06-15-38`

- **Portrait photography** - Photos where the individual people in the photo are the primary motive, such as a class photo.
- **Situational photography** - Photos where the action or event is the primary motive, such as a football match. These types of images may be distributed publicly *without* the permission of the subjects. However, if the subjects are identifiable, handling the images will still be considered to be *processing of personal data* and falls under the regulations of the Personal Data Act [27].

However, there is often not a clear distinction between the two types:

> *It can sometimes be difficult to decide if something is a situational photo or a portrait photo, or if the situation could be insulting to someone. Therefore, one should as a general rule always ask for permission to share the photo or video.* (The Norwegian Data Protection Authority [27] , translated)

Since the regulations apply to all companies processing personal data belonging to European citizens, this also includes many companies outside of Europe. As a result, the GDPR ends up affecting handling of data not only in Europe, but the entire world.

Failing to comply with the regulations may result in fines of up to €20 million or 4% of global revenue (whichever is higher), in addition to potential compensation for damages to the data subjects [71]. The fine based on percentage of global revenue is especially impactful because it results in a violation of the regulations being a serious issue even for big, international businesses. As of May 2022, businesses had in total been fined over €1.6 billion [35].

This makes it critical for businesses to comply with the regulations, as the alternative is risking a significant fine. However, media companies often distribute a lot of content, which makes manually ensuring adherence to the regulations particularly difficult. Consequently, it would be very useful to have an automated system which facilitates adherence to the regulations by automatically blurring faces in the distributed media, thereby anonymizing the subjects such that their permission to publish the media is no longer required. However, not all faces should be blurred in every video, of course, which increases the complexity of the problem somewhat since the faces have to be selectively blurred.

1.2 CuttingRoom

CuttingRoom is a browser-based video editing and publishing tool [36]. The platform is a cloud-native Software as a Service (SaaS) product built on Amazon Web Services (AWS), which allows the user to ingest footage both from storage and live streams.

The videos processed by CuttingRoom's customers may, of course, include identifiable European citizens. As previously mentioned, this is considered processing of personal data and therefore falls under GDPR. Therefore, it would be beneficial for CuttingRoom's customers if CuttingRoom were to facilitate adherence to applicable privacy regulations, which would also be a potential selling point for CuttingRoom.

1.3 Existing solutions

At the time of writing, there does not appear to be any fully automated tool for selectively blurring faces in videos available neither free-to-use nor commercially. There are many tools which allows the user to blur faces, but most of these seem to rely on the user manually positioning the blur. The user would then have to do this for every frame, use key-frames or motion tracking to track the blurred face throughout the video.

The following is a list of example tools which have been recommended for blurring faces in videos by various online sources[3]:

- Flixier (`https://flixier.com/tools/blur-face-in-video`)
- DaVinci Resolve (`https://www.blackmagicdesign.com/products/davinciresolve/`)
- Lightworks (`https://lwks.com/`)
- Wondershare Filmora (`https://filmora.wondershare.com/video-editor/`)

None of them fully automates blurring a selection of faces in a video, but Wondershare Filmora does offer a "Face-Off" effect, where a face is automatically detected and may be blurred, or replaced by an emoji or other graphic [33]. This does, however, appear to blur faces indiscriminately rather than selectively, and may struggle in non-optimal scenarios. According to Liza Brown, chief editor at Filmora, a high focus on the subject is required, and faces may not be detected in dark conditions [33].

[3]Source articles:
`https://listoffreeware.com/best-free-software-blur-face-in-video-windows/`,
`https://filmora.wondershare.com/video-editor/best-blur-faces-apps.html`,
`https://filmora.wondershare.com/video-editing-tips/blur-face.html`,
`https://flixier.com/tools/blur-face-in-video`

1.4 Proposed solution

This thesis aims to produce a solution which should facilitate adherence to the privacy regulations for CuttingRoom and its customers. For this purpose, the solution should automatically blur a selected group of faces in videos to avoid potential breaches of privacy. By effectively anonymizing a group of people in the video, their permission is no longer needed to publish the media. As a result, whether the video would be considered situational or portrait is no longer of any concern.

In, for instance, a news broadcast at a location, the faces walking by in the background should likely be blurred, while the news reporter's face should not. In this case, all *unrecognized* faces should be blurred. However, in a video where the main focus of the video should remain anonymous, like in many criminal cases, all *recognized* faces should be blurred. The solution should therefore be able to both blur either recognized or unrecognized faces, thereby freely specifying the group of faces to blur.

1.5 Solution requirements

The requirements for the solution include the following:

- **Technical requirements** - The resulting solution should be able to be integrated with CuttingRoom and must be designed in a way which allows it to do so.
- **Regulatory requirements** - The licenses of the software and data being used in the thesis must allow for the solution to be used in a commercial product.
- **Performance requirements** - The videos being edited may be several hours long. If the solution uses too much time, or the quality of the blurring is too low, it will essentially be unusable.

In order to blur faces, the solution needs to detect all faces in each frame of the video, and recognize each face in order to determine which faces should and should not be blurred.

There is no specific time or accuracy requirement, as these are often trade-offs. Larger models may offer greater accuracy than smaller models, but also use more time. However, regardless of model being used, it is unrealistic for the solution presented in the thesis to be 100% accurate. As a result, the processed video may need to be manually reviewed before publication depending on the usage scenario.

1.6 Project scope

Although multiple forms of media may be considered personal data, video is the media form applicable to CuttingRoom's use case. Additionally, still images would, for instance, likely pose a different and more limited problem than video, due to its smaller data sizes (and thus smaller computational requirements) and lack of continuity between images. The thesis will therefore be specific to video.

People can also be identified based on other factors than their face, such as hair, clothes, tattoos or other distinct bodily features. More sensitive videos may therefore also need additional considerations before publication. In some cases, it may be more appropriate to blur or remove the entire persons body from the video. Similarly, voices may also be used to identify someone. However, both voices and other bodily features need to be handled differently than faces. Removing these features may therefore require a more nuanced and specific process than blurring a subset of the faces on screen, and will consequently not be covered by this thesis.

To edit videos, CuttingRoom first splits the videos into individual frames using *FFmpeg* and stores the frames in an AWS S3 bucket (see Section 2.5.1 and 3.1.1). The solution should therefore use this as a starting point and use the individual frames as input, and similarly output the modified frames, thereby making the solution able to be integrated with CuttingRoom's existing framework. The scope of the thesis does *not* include actually integrating the solution with CuttingRoom as this has to be CuttingRoom's own choice and responsibility, and it makes no impact on the rest of the thesis.

1.7 Report structure

After the current introduction, some preliminary background is presented in Chapter 2 to ensure a baseline understanding of important concepts discussed throughout the rest of the report. Afterwards, the main implementation is presented in Chapter 3, which is further analyzed in Chapter 4. Lastly, the report is concluded with Chapter 5.

The full source code is available publicly on GitHub: `https://github.com/ErlendF/face_blur`. Parts of the code may be too large to include in a single listing, or does not contribute substantially to the text, and are therefore not included in the text nor appendices. The source of the listings is included in footnotes wherever applicable.

Throughout the report, various example videos are referenced. Each of these are publicly available as unlisted YouTube videos due to its ease of publication and video hosting. The links to the example videos are provided in footnotes or in the text itself. Please note that a lot of the examples are timestamped to a particular point in the video. A full list of the various example videos is provided in Appendix C.

Chapter 2

Theoretical background

This chapter gives an introduction to various tools and technologies which may be useful to have some preliminary knowledge of when reading the thesis. The explanations do not go in-depth for each technology, but give an overview to ensure the understanding of its use in the implementation discussed in Chapter 3.

2.1 Video

These are some tools, methods and terminology which in the context of this thesis will be used in relation to video and video formats. For instance, neither *smoothing* nor *interpolation* is related to video, but this is the only context for which it is used in this thesis.

2.1.1 FFmpeg

FFmpeg is an open source collection of tools and libraries used to process multimedia content such as video [67]. The tools and libraries allow the user to (among other things) play, manipulate, convert and stream multimedia content. For the purposes of this thesis, it will only be used to split video into individual *frames*. A frame is a single image in the sequence of images which compose a video. Videos are commonly composed of 24, 30 or 60 frames per second.

2.1.2 Smoothing

Data smoothing is used to remove statistical noise and other minor fluctuations from a data set, thereby attempting to capture the important patterns of the data [39, 66]. Many various smoothing algorithms may be used, each with their own advantages and disadvantages, or other minor differences.

2.1.3 Interpolation

In numerical analysis, interpolation is a type of estimation [69]. It is used as a method of constructing (finding) new data points based on existing, known data points. Interpolation may be particularly useful in the case of incomplete data. For the purposes of the thesis, it will specifically be used for estimating the position of bounding boxes for likely false negatives (see Section 3.1.9) and non-sampled frames when dynamically processing videos (see Section 3.1.3).

2.1.4 Shot transition detection

A video shot is a continuous part of a video without any transitions. They are composed of a series of interrelated consecutive frames, and usually represent a continuous action in time and space [68]. In most forms of processed videos, the shot changes throughout the video and there needs to be some sort of transition between the shots. These transitions are usually divided into two main categories:

- **Abrupt transitions** - An abrupt change in the video, often in a single frame and with large differences between the frames before and after the transition, which makes it easier to detect. This usually produces a visual discontinuity in the video. They are often used to change the subject of the video, or show the subject from a different angle.
- **Gradual transitions** - A more gradual change in the video, often with some graphical effect. These are more difficult to detect than the abrupt transitions as the content of the video changes over a greater number of frames which makes it more difficult to distinguish from other changes inside a single shot.

For the purposes of the thesis, detecting the shot transitions automatically may be especially useful in making facial sequences as there is no logical connection between the position of faces in different shots (see Section 3.1.4). Therefore, detecting the shot transitions may help distinguish faces which are in the same part of the screen after each other when there is a shot transition between them. Incorrectly matched faces may cause jarring artifacts of bounding boxes traveling between the two faces seemingly without reason[1].

2.2 Facial recognition pipeline

A modern face recognition pipeline conventionally consists of four stages: detection, alignment, representation and classification [65]:

- Facial detection is the task of detecting faces in an image. This is further discussed in Section 2.3.
- Facial alignment is the task of identifying the geometric structures of faces in an image, and attempt to obtain a canonical alignment of the face based on translation, scale and rotation [55]. In this thesis, this step is in practice combined with facial detection, since a single model will produce both the location of a face and its *facial landmarks* (key points of a human face, e.g. corners of eyes, nose tip, corners of the mouth etc.). The facial landmarks will then be used to align the face. Therefore, it makes sense to discuss them in the same context in this case.
- Facial representation is the task of mapping an image of a face to a target space such that the distances in the target space corresponds to a measure of similarity between faces [57, 34, 65]. Consequently, the representations of the face in the target space may be compared to each other using simple distance metrics, such as Euclidean- or cosine distance, in order to make a classification. Since two images of faces is likely to contain a lot of noise and cannot be viably compared directly, the representation of facial features makes it possible to compare them. Ideally, the facial features vector representations should be as identical as possible for the same person, whereas representations of different people should be as dissimilar as possible, thereby differentiating them.

[1]Example timestamped at 00:09: `https://youtu.be/IuDkF_XdqWc?t=9`

- Facial classification is the task of classifying a face based on its facial feature vector representation. Two vector representations are compared to each other to determine if they are of the same person or not by using some form of classifier. For the purposes of this thesis, this will primarily be a simple distance metric. These will produce a single number score for the distance between the vectors. If the score is below a certain threshold, the faces are classified as the same. If one of the faces compared is a known person, the other face has consequently also been identified. Similarly to the discussion of facial detection and alignment, facial representation and classification will be discussed together under *facial recognition* as both are needed to recognize faces.

Facial detection and alignment is discussed in Section 2.3 and the facial representation and classifications is discussed in Section 2.4.

2.3 Facial detection

Facial detection is used to detect faces in images, and it is one of the most important part of the thesis. Without facial detection, it would be impossible to target any faces for blurring. The following are some common technologies utilized for detecting faces. This is not their only use, but it is the purpose of interest for this thesis. The libraries and methods implemented for facial detection during the thesis will be discussed in Section 3.2.

2.3.1 Convolutional neural network

A Convolutional Neural Network (CNN) is a type of artificial neural network with multiple layers, such as convolutional layers, non-linearity layers, pooling layers and fully-connected layers [2]. They are commonly used in, for instance, computer vision and neural language processing tasks, including facial detection and recognition. For the purposes of the thesis, the composition and internal workings of CNNs are of lesser importance and will therefore not be discussed in further detail.

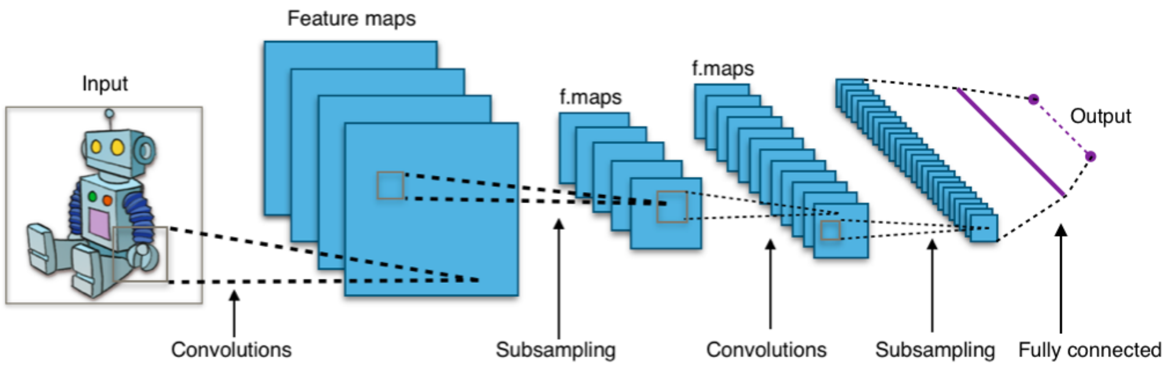

Figure 2.1: Illustration of a typical convolutional neural network, image by Aphex34[1][2]

2.3.2 Histogram of oriented gradients

The histogram of oriented gradients (HOGs) is a feature descriptor used for object detection and recognition in images, including face detection and recognition [54, 40]. When utilizing HOGs, the image is divided into small regions, called cells, and a histogram of edge orientations is computed for each cell. The feature descriptor itself is the concatenation of these histograms.

For the purposes of the thesis, the main advantage of this technique is its low computational cost compared to the alternatives, such as methods based on CNNs. However, it is also often less accurate than the other alternatives. This technique is made available through the dlib package (see Section 3.2.5), where either HOG or a CNN may be used for facial detection. HOG should in this case primarily be used if the computational cost is the main deciding factor as it is less accurate than its counterpart [56].

[1]`https://creativecommons.org/licenses/by-sa/4.0/deed.en`
[2]`https://commons.wikimedia.org/wiki/File:Typical_cnn.png`

2.4 Facial recognition

Facial recognition is used to recognize each face and distinguish the faces of different people from each other. To do so, a vector representation of each face's features is made and used to make a classification, as discussed previously.

In recent years, deep convolutional neural networks have become the method of choice for facial representation [37]. To train these networks to produce facial features representations with the highest possible discriminative power, there are two main lines of research: Methods which train a multi-class classifier to distinguish between the classes of the dataset by using a softmax classifier, and methods which directly learn a representation, such as the triplet-loss.

Softmax-loss based methods train a network to correctly classify each person in a training set, where the person is predicted by feeding the output of the final layer of the network into a softmax unit [65]. The last, or second to last, layer of the network will in this case have a network activation which is as unique as possible to the specific face in order to distinguish it from other faces. These activations may consequently be used as a representation of the face's features.

Triplet-loss based methods use pairs of images where a given face (the anchor) is compared to another image of the same person (a positive) and an image of a different person (a negative) [57]. By basing the loss on minimizing the distance to the positive and maximizing the distance to the negative, the network is directly trained to produce a representation of the faces.

For the purposes of the thesis, being able to recognize faces is of particular importance both when identifying the same face in each frame (see Section 3.1.4) and when selecting which faces in the videos to blur (see Section 3.1.7). The various implemented tools and models for facial recognition are discussed in Section 3.2.

2.5 Infrastructure

Infrastructure is an important aspect of the solution. It is a requirement that the solution needs to be able to be integrated with CuttingRoom, and the infrastructure of the solution therefore needs to allow for this. Additionally, it is important for both its cost and effectiveness. Automating the deployment of the infrastructure makes it far easier both to integrate initially and maintain over time, thereby making it cheaper. The reliability and cost of running the solution itself also depends on the infrastructure.

2.5.1 Amazon Web Services

Amazon Web Services (AWS) is a public cloud platform which offers a myriad of services and APIs in a pay-as-you-go model [10], and is the cloud platform used by CuttingRoom. It is one of the largest cloud platforms and it accounts for the largest market share of cloud infrastructure service providers at 33% [31]. At the time of writing, AWS also generated *all* of Amazon's operating profits [32].

Some of the AWS services most relevant to the thesis are listed below.

- **Amazon Elastic Compute Cloud (EC2)** - EC2 allows users to utilize virtual server instances to run their applications [8]. The user can select the type of machine and software they would like to use.
- **Amazon Simple Storage Service (S3)** - S3 provides simple object storage to the user [15]. All objects are stored in *buckets*, and each bucket can store any number of objects.
- **Amazon SageMaker** - SageMaker is a fully managed machine learning service provided by AWS [19]. It allows the user to build, train and deploy machine-learning models. It offers a number of features such as *Batch Transform* and *Processing* (see Section 3.3).
- **Amazon Elastic Container Registry (ECR)** - ECR is a container image registry service managed by AWS [6]. This allows the user to store container images in a private repository which may be used by other AWS services, such as AWS SageMaker.
- **Amazon Rekognition** - Rekognition is a high level, managed AWS service which offers various features like celebrity recognition, facial analysis and most notably facial detection [25].

2.5.2 Terraform

Terraform is an open-source Infrastructure as Code (IaC) tool made by HashiCorp which integrates well with most major cloud services such as AWS, GCP and Azure [46]. HashiCorp, and the various other providers[2], provide a multitude of Terraform *modules*; self-contained packages of Terraform configurations which simplifies working with various services [47].

Using an IaC tool like Terraform allows for easily setting up a consistent and reproducible set of infrastructure. It uses declarative configuration files written in HashiCorp Configuration Language (HCL) to describe resources. Every part of Terraform exists to facilitate the deployment, updating or deletion of resources. The configuration files define what the infrastructure *should* look like, and Terraform checks the currently deployed infrastructure and updates it to match the description in the configuration files.

```
resource "aws_s3_bucket" "example_bucket" {
    bucket = var.example_bucket_name
    acl    = "private"
}
```

Listing 1: Example of a simple Terraform resource

Using the resource definition in Listing 1, Terraform produces a plan similar to the plan shown in Listing 2.

[2]https://registry.terraform.io/browse/providers

```
> make plan
terraform plan -var-file=variables.tfvars -out=tfplan
(...)


--------------------------------------------------------------------------------------

Terraform used the selected providers to generate the following execution plan.
↪  Resource action are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_s3_bucket.example_bucket will be created
  + resource "aws_s3_bucket" "example_bucket" {
      + acceleration_status         = (known after apply)
      + acl                         = "private"
      + arn                         = (known after apply)
      + bucket                      = "this-is-an-example-name"
      + bucket_domain_name          = (known after apply)
      + bucket_regional_domain_name = (known after apply)
      + force_destroy               = false
      + hosted_zone_id              = (known after apply)
      + id                          = (known after apply)
      + region                      = (known after apply)
      + request_payer               = (known after apply)
      + tags_all                    = {
          + "Team" = "Platform"
          + "User" = "Erlend"
        }
      + website_domain              = (known after apply)
      + website_endpoint            = (known after apply)

      + versioning {
          + enabled    = (known after apply)
          + mfa_delete = (known after apply)
        }
    }

Plan: 1 to add, 0 to change, 0 to destroy.


--------------------------------------------------------------------------------------

Saved the plan to: tfplan

To perform exactly these actions, run the following command to apply:
    terraform apply "tfplan"
```

Listing 2: Example of Terraform plan output

For the purposes of the thesis, the infrastructure will be deployed to AWS primarily because the rest of CuttingRoom's infrastructure is already deployed there. Similarly, Terraform was chosen for the thesis primarily because CuttingRoom already deploys their infrastructure using it. Therefore, adding another Terraform module should require minimal effort.

2.5.3 Docker

Docker is a widely used, open-source containerization platform [41]. It enables developers to package their applications into *containers*, which are standardized executable components. This packages the application with all its dependencies and a operating system to run it. The container may be run in any environment which can run containers, which greatly simplifies distribution. For the purposes of this thesis, Docker has primarily been used for the deployment of the proposed pipeline, as discussed in Section 3.3.

Chapter 3

Implementation

This chapter will discuss the implementation of the proposed solution, provide an explanation for various decisions and designs of the implementation. The solution is meant as an exploration of existing solutions and how best to combine them for the purpose of blurring specific faces in videos. Therefore, it is meant to be as technology agnostic as possible. The proposed solution is a pipeline, where each part of the pipeline should be interchangeable and easily replaceable with new facial detection, facial recognition, processing, interpolation modules etc. Therefore, various options will be discussed wherever there are multiple viable candidates and their replacement may have a significant impact on the result.

Overview

Figure 3.1 shows an overview of the proposed pipeline and all its parts:

1. The input video file is split into individual frames using FFmpeg (Section 3.1.1).
2. The frames are used to detect shot transitions in the video (Section 3.1.6).
3. The frames and shot transitions are used to process the video (Sections 3.1.2 and 3.1.3). This utilizes the facial detection, alignment and representation (Section 3.2).
4. The result of the processing is used to make facial sequences (Section 3.1.4).
5. The missing bounding boxes of the facial sequences are interpolated (Section 3.1.5).
6. The facial sequences of selected faces are removed (Section 3.1.7).
7. The movement of the bounding boxes in each facial sequence is smoothed out (Section 3.1.8).
8. The remaining faces are blurred (Section 3.1.11).
9. The output frames are combined to make the output video file (Section 3.1.1).

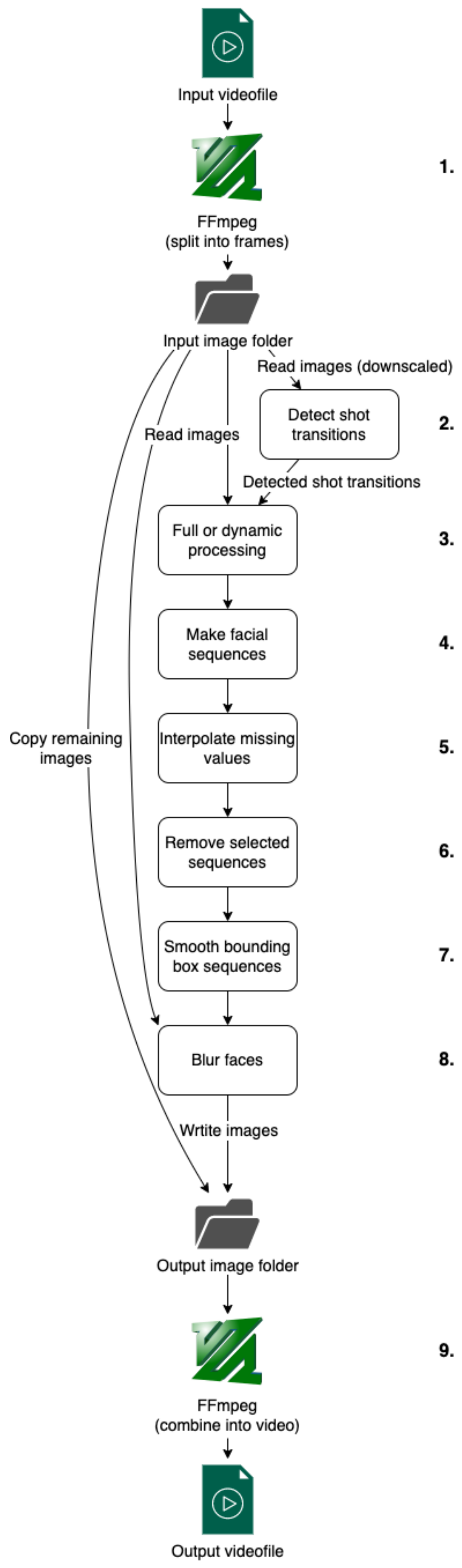

Figure 3.1: Overview of proposed pipeline

Data formats

Throughout this section, the facial detection and recognition are used to make *bounding boxes* and *facial features* for each detected face. The bounding boxes typically consist of four values representing x_1, y_1, x_2 and y_2, which form the corners of a rectangle. For the purposes of the thesis, the bounding box also contains the frame number in order to keep track of which frame is being processed at any given time. These values are combined into a list.

In some cases, bounding boxes may also contain the certainty of the model for the detected bounding box. However, not all models provide the certainty, it did not appear to be of much use when processing the videos. Therefore, it is not included in the bounding boxes used in the thesis.

```
# x1,        y1,        x2,         y2,        frame_nr
[1583.54030, 685.96475, 1770.47392, 962.15031, 373]
```

Listing 3: Bounding box format

The facial features representation is simply a list of floats which can be used to compare two faces, as discussed in Section 2.4.

```
[ 0.016151972115039825, -0.017258254811167717, -0.03813121095299721,
↪   -0.07499738782644272, 0.0544714480638504, -0.007867357693612576,
↪   0.02019760198891163, -0.010829063132405281, -0.0264094490557909,
↪   0.012919466942548752, -0.04304543137550354, -0.04340442642569542,
↪   -0.05539201945066452, -0.0020324839279055595, 0.039676737040281296,
↪   0.05799984186887741, 0.017670873552560806, (...)
]
```

Listing 4: Facial features format

The bounding box and the facial features representation are combined in a map to represent both the identity and location of a face. These maps are then combined in a list to form a sequence of faces representing its movement across each frame of the video. These sequences may then again be combined to contain all the information about every face in the video, its identity and location in every frame. Since not all faces are present at the same time in the video, the sequences of each face may start and end at differing frames. Additionally, frames may not be adjacent. If a face is not detected for a few frames, these frames will not be present in the sequence.

```
[
    [ # Sequence 1
        # Face 1, frame 1
        {"bbox":[32, 41, 25, 57, 1], "feat": [0.01, -0.01, -0.03, (...)]},
        # Face 1, frame 2
        {"bbox":[31, 42, 23, 53, 2], "feat": [0.01, -0.01, -0.03, (...)]},
        (...)
    ],
    [ # Sequence 2
        # Face 2, frame 90
        {"bbox":[98, 47, 77, 44, 90], "feat": [-0.05, 0.02, 0.01, (...)]},
        # Face 2, frame 91
        {"bbox":[96, 45, 79, 46, 91], "feat": [-0.05, 0.02, 0.01, (...)]},
        (...)
    ],
    (...)
]
```

Listing 5: Facial sequences format

In the implementation, facial detection, alignment and facial recognition is combined into a single processing function such that the function returns the format shown in Listing 5. An example of a function combining facial detection, alignment and recognition is shown in Listing 17.

3.1 Video

This section will explore various aspects of automatically blurring a selection of faces in a video. With the exception of the sequence models presented in Section 3.1.12, every section is relevant to the proposed solution.

3.1.1 Initial processing

As mentioned in Section 1.6, CuttingRoom splits their videos into separate frames using *FFmpeg*. FFmpeg will also be used to split the videos into frames for the purposes of the thesis. This is the starting point of the thesis and will not be further explored since it is already open source, easy to use and quite convenient. Additionally, it is already handled by CuttingRoom in their production environment. Consequently, by using the same tool, the thesis has the same starting point as CuttingRoom has in their production environment.

Listing 6 shows an example of how FFmpeg may be used. In this case, a video file named `input.mp4` is split into frames, starting from 15 minutes into the video, and ending at 20 minutes into the video. The resulting images will be written to the current directory, and be named `img0000001.png`, `img0000002.png` etc.

```
ffmpeg -i input.mp4 -ss 00:15:00 -to 00:20:00 img%07d.png
```

Listing 6: Using FFmpeg to split a video file

The processed frames may later be recombined into a video again. Listing 7 shows an example of how this may be done using FFmpeg.

```
ffmpeg -an -i img%07d.png out.webm
```

Listing 7: Using FFmpeg to combine frames into a video file

3.1.2 Processing every frame

Processing every frame is arguably the most intuitive way of processing videos. In this case, every frame is first processed completely independently of each other to detect faces and make a facial features representation for each of them. Processing every frame is, however, very computationally expensive. Additionally, the bounding boxes in the processed videos often appear a bit jittery[1]. The bounding boxes are likely moving around a bit from frame to frame due to small changes in the position or pose of the faces. Without any other form of processing, simply detecting faces and recognizing them would also likely leave a lot of false negatives in cases where a face is not in a conventional pose recognized by the model. Both the jitter and false negatives may be very disruptive to the viewer, causes unnecessary noise, and may expose the identity of the depicted individuals if not handled properly.

The jittering may be helped by combining the detected faces and their facial features representations into facial sequences. By knowing the sequence of faces belonging to one person, the sequence of bounding boxes may be smoothed to remove some of the unnecessary noise. Similarly, the sequence itself may help identify likely false negatives. If there are only a few frames where a bounding box is missing, the person is likely still there, but is not detected. These frames may therefore be interpolated to fill the gaps in the sequence. This is further discussed in Section 3.1.4, 3.1.8 and 3.1.5.

3.1.3 Dynamic processing

In many cases, there may not be a lot of movement in a scene, and the faces may stay fairly still. In these cases, it is possible to estimate the movement of the faces based on a few surrounding frames, and fill in the remaining frames *without* computationally expensive facial detection and recognition of every frame. The frames may then be processed in a given interval (such as every 1th, 15th, 30th, 45th frame etc.), and the bounding boxes in frames that are not directly computed may be interpolated based on the bounding boxes in computed frames. Interpolating the bounding boxes rather than directly processing them is significantly less computationally expensive. Interpolation is discussed further in Section 3.1.5.

[1]Example without further processing: `https://youtu.be/nSvN24R_wRU`

Unlike processing every frame, dynamic processing cannot be used to fully blur all the faces in a video without additional processing. At minimum, the bounding boxes in frames which were not directly computed need to be interpolated. In order to interpolate the frames, the faces in each frame needs to be matched to each other in order to know which bounding box should be interpolated to each bounding box of the next frame. This will be further discussed in Section 3.1.4.

Dynamically processing the video and interpolating the missing frames alone removes a lot of the problems of processing every frame discussed in Section 3.1.2. The resulting interpolated sequences of bounding boxes will naturally be smoother since there is no information gathered from the interpolated frames which may cause noise[2]. Additionally, any false negatives between the processed frames will simply be ignored since the frames are never processed. However, if a false negative occur in a processed frame, the frames before and after it will likely need to be reprocessed since there would be a change in the faces visible in frame.

When there is a shot transition in the video, the frames cannot be interpolated across the shot transition since there is no correlation between the position of the faces in two distinct shots. Consequently, additional frames need to be processed before and after the transition to retrieve all necessary information to properly blur faces in every frame of the video. Identifying shot transitions is therefore essential to interpolating frames correctly. This is discussed further in Section 3.1.6.

In addition to shot transitions, there may also be other changes in the faces visible in the frame, such as when there is a false negative or positive, someone enters the frame, leaves the frame or simply turns around. This will either cause a change in the number of faces in the frame, or make it so the faces cannot be matched to each other. In these cases, it is not possible to know when the change took place without checking some of the frames in question. The first part of Listing 8 shows an example of how frames may be added to a list of frames to be processed if the faces in two frames could not be matched to each other. When they cannot be matched, there is missing information in the frames between the two processed frames. Therefore, the frames between them need to be processed as well. In the listing, every frame between the processed frame is added for processing. However, this could potentially be done using another method, like binary, search to identify the frame(s) where the change(s) occurred. This could reduce the total number of necessary frames to process.

[2]This is an example of a dynamically processed video using the same models as the example in Section 3.1.2: `https://youtu.be/v_49prmpeec`

```python
# Two frames are compared, and faces are matched to each other
matches, matched = compare(frames_by_nr[prev], frames_by_nr[current])

# If the two frames are adjacent, there is no more exploration to do even if they do
↪  not match
if not matched and current != prev+1:
    # Queueing all frames between the non-matched frames
    for frame_nr in range(prev+1, current):
        imgs.append(get_file_name(frame_nr, img_dir, file_ext=file_ext))
        img_nrs.append(frame_nr)

    current = prev+1    # Setting back the current
    # Reducing the interval
    interval = max(int(interval*0.7), min_interval)
    process_consecutively = 0
    break
else:
    if current != prev+1:
        # If enough frames have been successfully processed consecutively, increasing
        ↪  the interval
        if process_consecutively >= proc_count_threshold:
            interval = min(int(interval*1.3), max_interval)
            process_consecutively = 0
        else:
            process_consecutively += 1

    # Storing the matching and updating completed
    matchings[(prev, current)] = matches
    complete = current
```

Listing 8: If the faces in the two frames could not be matched correctly, all frames between the two initial frames are processed[3]

It is very processing intensive to process every frame when encountering a disruption in the faces visible in the video, especially for longer processing intervals. For example with a processing interval of 15, there are 14 additional frames between the initially processed frames. If these frames also needed to be processed because there was a disruption in that interval, this would take 15 times longer than simply processing the initially processed frames. The same computational power could be used to process $14 * 15 = 210$ frames when using an interval of 15 if none of the frames between them needed to be processed independently. The bounding boxes in the remaining frames would then be interpolated instead.

[3]https://github.com/ErlendF/face_blur/blob/b9efa56c27fd69dedb01c378cff631fc2d045ac6/videoface/dynamic_processing.py

It is therefore important to optimize the interval for processing frames. However, as each video is different and the amount of movement and number of shots may vary significantly throughout each video, there is no global optimal processing interval. The interval is therefore dynamically adjusted based on how much can be processed at a time, as shown in Listing 8. If it is frequently needed to process the entire interval in parts of the video, the interval is decreased such that (hopefully) the interval is small enough not to require frequent reprocessing. Additionally, with a smaller interval, the cost of processing the entire interval in the case of a disruption is also lessened. Similarly, if there are very few disruptions, the interval is increased to faster processes the entire video.

3.1.4 Identifying facial sequences

In order to do frame interpolation (see Section 3.1.5), the faces in each frame need to be mapped to the same face in the next processed frame to make a sequence of bounding boxes for each face. This sequence can then be interpolated both to fill in gaps due to false negatives or dynamic processing (see Sections 3.1.9 and 3.1.3). A facial sequence also gives a better chance of correctly identifying a face, as there are multiple facial features representations available for comparison. The facial features detected may vary significantly depending on conditions such as illumination, pose and occlusion of the face. Consequently, it may be difficult to correctly identify moving faces which change pose or where the illumination is changing. For instance, when selecting which faces to blur, this is less of an issue when utilizing sequences of faces as there are multiple chances to identify the face. Facial alignment also helps to lessen the effect of such noise on the facial features representation.

When processing the frames in this thesis, there are two pieces of information which may be used to identify the same face in different frames: its position (bounding box) and facial features representation. Since the frames in each shot are interrelated, the bounding boxes generally do not move significantly between each frame. However, when there are larger gaps, such as in dynamic processing or false negatives, the potential distance moved increases multiplicatively by the number of frames in the gap. Quick camera movement may similarly result in the subjects moving rapidly across the frame[4]. Similarly, the facial features representations may also change, as discussed previously. Facial alignment also helps reduce these differences in order to make the representations

[4]Example timestamped at 2:23: `https://youtu.be/tORTBS7iEGY?t=143`

more consistent, thereby increasing the chances of correctly recognizing the person. The positions of the bounding boxes are also not related across shot transitions, so identifying scene transitions is important for accurately identifying facial sequences. This is further discussed in Section 3.1.6.

Matching each face to the same face in different frames is essentially a *minimum cost bipartite matching* problem. Given a complete, weighted, bipartite graph

$$G = (V, E)$$

where V is composed of two disjoint subsets S and T, the objective is to find the minimum cost matching between S and T of maximum cardinality [26]. S and T are in this case the faces detected in the first and second frame which are being compared, and the comparisons between each face in a frame to the other form the edges of the graph. The calculated distance between the faces, as shown in Listing 9, provides the cost of each edge in the problem. Finding the minimum cost matching with maximum cardinality, maximizes the likelihood of correctly matching each face to itself. Scipys `linear_sum_assignment` [5]function is particularly useful when matching faces for this purpose.

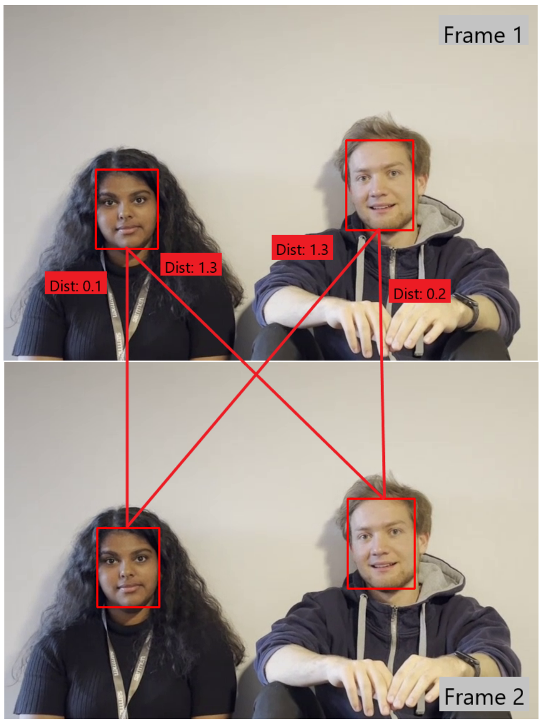

Figure 3.2: Illustrated example of distances mapping faces between frames (the distances are example values)

In Listing 9, a weighted distance metric of the *bounding box distance* and *facial feature distance* is made for comparing two faces `f1` and `f2`. These two distances are multiplied by hyperparameters to be able to adjust the impact of each on the total distance. This combined distance is used to identify the same face in each frame by trying to minimize the total distance, as discussed previously. If the distance of a given face to another is above a set threshold `dist_threshold`, they are assumed to be different faces, regardless of the optimal matching between the two frames.

[5]https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.optimize.linear_sum_assignment.html

The hyperparameters `dist_threshold`, `feat_weight` and `bbox_weight` shown in Listing 9 were somewhat arbitrarily chosen based on manual testing for the purposes of the thesis. These should be properly adjusted for a production setting.

```
dist_threshold = 1.0
feat_weight = 1.0
bbox_weight = 0.001

def bbox_dist(f1, f2):
    return (abs(f1['bbox'][0]-f2['bbox'][0]) + abs(f1['bbox'][1]-f2['bbox'][1]) +
    ↪  abs(f1['bbox'][2]-f2['bbox'][2]) +
    ↪  abs(f1['bbox'][3]-f2['bbox'][3]))*abs(f1['bbox'][0]-f1['bbox'][2])/100


def feat_dist(f1, f2):
    return cosine(f1['feat'], f2['feat'])


def face_dist(f1, f2):
    fd = feat_weight*feat_dist(f1, f2)
    bd = bbox_weight*bbox_dist(f1, f2)
    return fd + bd
```

Listing 9: Weighted distance metric for comparing two faces[6]

When comparing n number of faces in one frame to m number of faces in another frame, this problem scales by $n * m$. In most cases, this will not be an issue, but it may require a greater amount of computational resources when processing footage of large crowds of people. Large crowds of people would, however, most likely be counted as *situational photography*, and not actually require any blurring of faces.

3.1.5 Frame interpolation

Interpolating missing bounding boxes may be necessary both in the case of a false negative, where a face may not be detected for a few frames (see Section 3.1.9), and when dynamically processing a video (see Section 3.1.3). When interpolating the bounding boxes, there are four values to be interpolated for each frame: x_1, y_1, x_2 and y_2, where (x_1, y_1) define one corner of a rectangle and (x_2, y_2) define the opposing corner. The

[6]https://github.com/ErlendF/face_blur/blob/b9efa56c27fd69dedb01c378cff631fc2d045ac6/videoface/dist.py

bounding box is the rectangle formed by these two points. Bounding boxes may alternatively be formed by defining one point of the rectangle in addition to the width and height, but these formats are interchangeable and is not used in the pipeline. Both the Deepface framework and AWS Rekognition discussed in Sections 3.2.6 and 3.2.7 do, however, return this format, which therefore needs conversion. Regardless of the format, each of the four values may be interpolated independently, which in turn composes a complete interpolation of the bounding box in non-processed frames. The following are a few common types of interpolation and their applicability when interpolating bounding boxes.

Piecewise constant interpolation

This is the simplest type of interpolation, where all interpolated points are assigned the same value as their closest valid data point, which is why it is also called *nearest-neighbour interpolation* [69]. In this case, any frame which does not have a valid bounding box would simply use the same bounding box as the nearest valid frame. This would require a lot of data points to fully blur a persons face while they are moving, and it would be very visually disruptive with the blur ”jumping” from position to position if there are a lot of frames without bounding boxes.

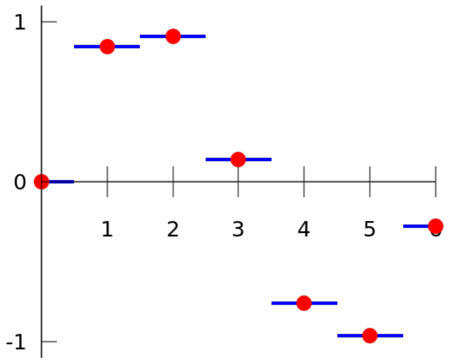

Figure 3.3: Illustration of a piecewise interpolation[7]

Linear interpolation

This is one of the least computationally expensive interpolation methods that will adequately fill in gaps in the sequence of bounding boxes without significant visual disruptions. It simply makes a straight line between each data point and interpolates the value of enclosed points to this line. For a line between (x_a, y_a) and (x_b, y_b), the value y of a point with an x value between x_a and x_b would be [69]:

$$y = y_a + (y_b - y_a)\frac{x - x_a}{x_b - x_a}$$

[7]Source: `https://commons.wikimedia.org/wiki/File:Piecewise_constant.svg`

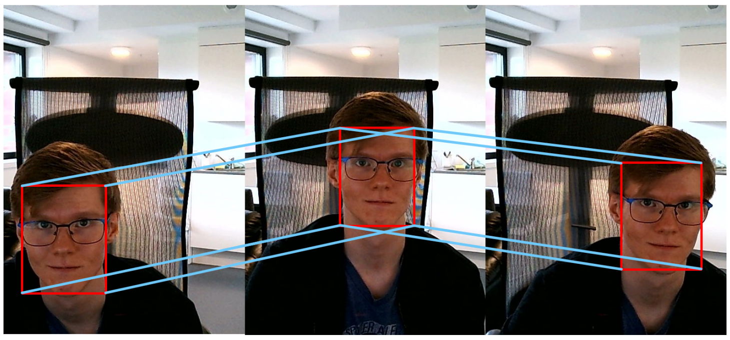

Figure 3.4: Illustration of how a linear interpolation would look for facial detection bounding boxes. The corners of interpolated bounding boxes would fall on the blue lines.

Polynomial interpolation

Instead of a linear function, this method fits a polynomial of a higher degree to match the data points. This method may be more computationally expensive, but it may provide a smoother interpolation than linear. People do not move linearly. Each movement has momentum; it takes time to speed up and slow down. Every movement is not random, and a polynomial interpolation would likely capture this better than a linear interpolation.

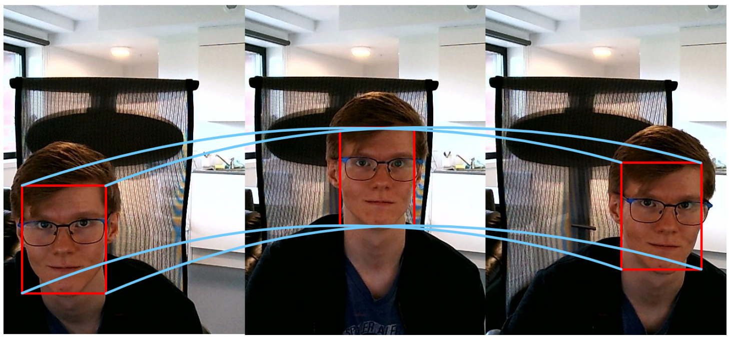

Figure 3.5: Illustration of how a polynomial interpolation would look for facial detection bounding boxes. The corners of interpolated bounding boxes would fall on the blue lines.

Spline interpolation

This method uses low-degree polynomials for a series of intervals, which collectively composes a complete function, called a *spline* [69]. The polynomials are chosen such that they fit smoothly together. An illustration of this for bounding boxes would look similarly to Figure 3.5, and it would likely be visually similarly to a polynomial interpolation.

Scipy interpolators

The SciPy package offers a collection of various easy to use interpolators[8]. In this case, it is the *univariate* interpolators that are of primary interest, as each of the four values composing a bounding box may be interpolated independently. Most of the univariate interpolators have a nearly identical interface, and may therefore be used interchangeably. Listing 10 shows how these interpolators may be used to interpolate the missing bounding boxes in a sequence of bounding boxes. In this example, the default interpolator is the *Piecewise Cubic Hermite Interpolating Polynomial (PCHIP)* interpolator, but it could be replaced by passing another interpolator in the `interpolator` parameter.

[8] `https://docs.scipy.org/doc/scipy/reference/interpolate.html`

```python
def interpolate(seqs, interpolator=pchip_interpolate):
    for i in range(len(seqs)):
        if len(seqs[i]) == 1:
            continue

        x1_observed = [f["bbox"][0] for f in seqs[i]]
        y1_observed = [f["bbox"][1] for f in seqs[i]]
        x2_observed = [f["bbox"][2] for f in seqs[i]]
        y2_observed = [f["bbox"][3] for f in seqs[i]]
        frame_nrs = [f["bbox"][4] for f in seqs[i]]

        # Using the previous valid feature for every frame missing it
        feats = []
        next = 0
        for j in range(seqs[i][-1]["bbox"][4]+1-seqs[i][0]["bbox"][4]):
            feats.append(seqs[i][next]["feat"])

            if seqs[i][next]["bbox"][4] == j+seqs[i][0]["bbox"][4]:
                next += 1

        # +1 to make inclusive
        x_pred = np.arange(seqs[i][0]["bbox"][4], seqs[i][-1]["bbox"][4]+1)

        # Interpolating each value separately
        x1_pred = interpolator(frame_nrs, x1_observed, x_pred)
        y1_pred = interpolator(frame_nrs, y1_observed, x_pred)
        x2_pred = interpolator(frame_nrs, x2_observed, x_pred)
        y2_pred = interpolator(frame_nrs, y2_observed, x_pred)

        # Replacing sequence with fully interpolated sequence
        seqs[i] = [{"bbox": [x1, y1, x2, y2, nr], "feat": feat.copy()} for x1, y1,
        ↪  x2, y2, nr, feat in zip(x1_pred, y1_pred, x2_pred, y2_pred, x_pred,
        ↪  feats)]
    return seqs
```

Listing 10: Interpolating missing frames using SciPy interpolators[9]

Manually implemented linear interpolator

Listing 11 is an example of a manual implementation of a linear interpolator for bounding boxes. The valid frames are identified, and each bounding box in the frames in between is calculated using the difference between the positions of the bounding boxes in the valid frames divided by the number of frames between them. This results in the average amount

[9]https://github.com/ErlendF/face_blur/blob/b9efa56c27fd69dedb01c378cff631fc2d045ac6/videoface/interpolate.py

of movement for a given bounding box per frame, and the direction of the movement, either positive or negative. Lastly, the average amount of movement is multiplied by the difference in frame number to get the value of the bounding box for that given frame.

The manual implementation functions as intended, but is far more complicated and unnecessary than using the scipy interpolators as shown in Listing 10.

```python
def interpolate(seqs):
    to_add = []

    for i in range(len(seqs)):
        prev_frame = -1
        for j, s in enumerate(seqs[i]):
            if prev_frame == -1:    # No previous frames to interpolate
                prev_frame = s[4]
                continue

            if s[4] == prev_frame+1:    # No frames between to interpolate
                prev_frame = s[4]
                continue

            frame_diff = s[4] - prev_frame
            new_faces = []
            for idx, k in enumerate(range(prev_frame, s[4]-1)):
                new = [0, 0, 0, 0, k+1]
                for l in range(4):
                    new[l] = seqs[i][j - 1][l] + ((idx + 1) * ((seqs[i][j][l] -
                    ↪  seqs[i][j - 1][l]) / frame_diff))
                new_faces.append(new)

            to_add.append((i, j, new_faces))
            prev_frame = s[4]

    for i, j, v in reversed(to_add):
        seqs[i] = seqs[i][:j] + v + seqs[i][j:]

    return seqs
```

Listing 11: Manually implemented linear interpolator[10]

[10] https://github.com/ErlendF/face_blur/blob/3cf47446a595e8731b50fff576f73562e83d73ad/videoface/interpolate.py

3.1.6 Detecting shot transition

Automatically detecting shot transitions is an active field of research, and has been so for over two decades [63]. There are some existing standalone solutions with great results for this problem available for commercial use, but most of these solutions seem to require the video file itself rather than the frames from the video. This is the case, for instance, for *PySceneDetect*[11]. This is an issue since the video file may be very large and it may be impractical to load the entire file into memory at once. Additionally, the video file will not be available in CuttingRoom's production environment (see Section 1.6) and can therefore not be used directly in their production environment. It would also be preferable for the solution to be available in a Python library as this would make it significantly easier to integrate with the rest of the solution. It would be possible to call other programs from Python, but this may hurt performance due to out-of-process calls.

TransNetV2 is a deep neural network intended to automatically detect shot transitions [63]. Most notably, it is well suited for integration in this solution because it is implemented in Python, can use frames directly rather than a video file and is available for commercial use under the MIT license. It builds on TransNet made for the same purpose which was also written by, among others, the same authors [64]. Like TransNet, TransNetV2 operates on smaller, resized frames (*47 x 28*) of the video and effectively produces a single number per frame representing the predicted likelihood of that frame being part of a shot transition. According to their paper, the TransNetV2 model had an F1-score ranging from 77.9 to 96.2 on the various datasets they tested [63].

The detected shot transitions may be used both when dynamically processing the video and identifying facial sequences. In both instances the detected shot transitions are very beneficial.

Dynamically processing

When dynamically processing the video, the detected shot transitions make it possible to check the frames before and after a transition to ensure all necessary information is collected to produce an accurate interpolation of the bounding boxes in the scene. Additionally, this reduces the need for reprocessing an interval when the interval contains a shot transition which changes the faces in the shot.

[11]`http://scenedetect.com/en/latest/`

Utilizing the shot transitions when dynamically processing the video is fairly straightforward. The only part affected is determining which frame should be processed next. Normally, this would be the next frame of the interval (1th, 15th, 30th, 45th etc.). However, if there is a shot transition in between any of these, both the frame *before* the transitions starts and the frames included in the transition itself should be processed. Afterwards the processing may continue at the regular interval.

Identifying facial sequences

When identifying facial sequences, using the detected shot transitions prevents incorrect matchings across shot transitions, which may appear very jarring to the viewer[12]. Furthermore, it also removes incorrect matchings to false positives across shot transitions.

Utilizing the detected shot transitions when identifying facial sequences requires a bit more forethought. When initially making the sequences, faces should not be combined into a single sequence across a shot transition. Therefore, if there is a transition between the previously processed frame and the next, all the faces of the newly processed frame should form new sequences themselves. After all the initial facial sequences are made, facial sequences likely belonging to the same person are combined into a single sequence if they are only a few frames apart. When combining the sequences, facial sequences should not be combined across shot transitions since there is no correlation between the faces across a shot transition.

3.1.7 Selecting specific faces

After identifying the facial sequences, as discussed in Section 3.1.4, the sequences may be filtered to remove (or keep) specific faces to select which faces should and should not be blurred. Any sequence which is kept will be blurred, while any removed sequence will not. The sequences themselves are beneficial to correctly identifying the selected faces. Since each of the sequences may contain multiple examples of a face's features representation, it makes it possible to do multiple comparisons between a sequence and an image, or two different sequences, to determine if they are of the same person or not. Various noise, like the pose of a face and its illumination conditions, may significantly affect the facial features representation, and consequently make it difficult to identify a person based

[12]Example timestamped at 00:09: `https://youtu.be/IuDkF_XdqWc?t=9`

on a single image. Since both the facial features representations and position of a face are used to identify facial sequences, they are more reliable than using facial features representations alone. Therefore, the resulting sequences may contain the same face in a multitude of conditions, which makes it easier to identify in other parts of the video, or based on known images.

The faces may be selected in various ways, two of which are demonstrated in this thesis: filtering known faces and filtering faces based on time and location. In both cases, facial features representations are compared using *cosine similarity* (1-*cosine distance*), a measure of similarity between two vectors [72]. Cosine similarity is used instead of cosine distance simply to make it a bit more intuitive to adjust the threshold for determining two faces to belong to the same person; a higher score is a closer match between faces, 1 being a perfect match.

Filtering known faces

If a face is already known (there are existing images of the person available), these images may be processed by the same facial detection, alignment and recognition models as the video such that they have the same form of representation. Consequently, these facial features representations of the known images may be compared to the facial sequences of the video in order to identify them.

Multiple images of each known person may be used to give a better chance of recognizing them. It does not matter whose faces are known, only that they are known, and having multiple images of the same person consequently does not matter either. Either the known faces should be blurred, or the unknown faces. There should not be a need where some known faces should be handled separately from both other known faces *and* unknown faces. This may, however, in such a case also be accomplished by filtering the sequences multiple times.

```python
# Should use the same processing function as when parsing the video
def init_known_faces(known_people_img_dir, processing_func=deep_face_process,
↪   frames=None):
    processed_faces = {}

    if frames is None:
        img_names = sorted(glob(join(known_people_img_dir, "*")))
        img_nrs = np.arange(0, len(img_names))
        processed_faces = processing_func(img_names, img_nrs)
    else:
        img_names = np.arange(0, len(frames))
        img_nrs = np.arange(0, len(frames))
        processed_faces = processing_func(img_names, img_nrs, frames=frames)

    if len(processed_faces) == 0:
        return None

    known_faces = []
    for img in processed_faces.values():
        for f in img:
            # The positions of the known people from the reference photos is of no
            ↪   importance, removing it
            known_faces.append(f["feat"])

    return known_faces
```

Listing 12: Parsing known faces[13]

After parsing the known faces, their facial features representations may be compared to the facial sequences to find the sequences belonging to known people as shown in Listing 13. For this purpose, each of the images containing known people may be compared to several samples from each sequence to give a better chance of correctly identifying the face. Ideally, these samples should be spread across the facial sequence as evenly as possible to give the best chance of showing the face in multiple facial poses under multiple illumination conditions. In Listing 13, a simple way of spreading the samples is shown using the interval `intv`. The known faces identified may then be kept or removed depending on the value of the `remove_known` parameter.

[13]https://github.com/ErlendF/face_blur/blob/b9efa56c27fd69dedb01c378cff631fc2d045ac6/videoface/filter_faces.py

```python
def filter_known_faces(facial_sequences, known_faces, remove_known=False, samples=5,
↪   threshold=0.75):
    comparisons = [0] * len(facial_sequences)

    # Comparing each known face to a sample of each facial sequence
    for i, seq in enumerate(facial_sequences):
        for j, f in enumerate(known_faces):
            if len(seq) < samples:
                for s in seq:
                    sim = 1-cosine(f, s["feat"])
                    if sim > comparisons[i]:
                        comparisons[i] = sim
            else:
                intv = len(seq) // samples
                if intv < 1:
                    intv = 1

                for k in range(samples):
                    sim = 1-cosine(f, seq[k*intv]["feat"])
                    if sim > comparisons[i]:
                        comparisons[i] = sim

    if remove_known:
        return [f for f, s in zip(facial_sequences, comparisons) if s < threshold]
    else:
        return [f for f, s in zip(facial_sequences, comparisons) if s >= threshold]
```

Listing 13: Filtering known faces[14]

Filtering faces of known time and location

In addition to faces with pre-existing images available, faces may also be selected based on their time and location in the video being processed. The time and location may be used to identify the facial sequence containing the selected face, which again may be used to identify other facial sequences belonging to the same face. All instances of the persons face may thereby be blurred, or not blurred depending on the scenario.

In Listing 14, the location is input to the function as the parameters `x` and `y`, and the time is input as the parameter `frame_number`. These are used to identify the closest sequence, which again is used to identify sequences belonging to the same face. All the sequences belonging to the face are then filtered. The function may be used multiple times, and later combined to keep sequences of faces belonging to multiple people.

[14]https://github.com/ErlendF/face_blur/blob/b9efa56c27fd69dedb01c378cff631fc2d045ac6/videoface/filter_faces.py

```python
# Filter a selected face based on location and frame number. Use the sequence to
↪  identify other sequences with the same face. x = y = 0 is the top left corner of
↪  the frame.
def filter_selected_face(sequences, frame_number, x, y, remove_known=True, samples=5,
↪  threshold=0.75):
    closest_seq = -1
    closest_dist = float_info.max

    # Identifying the sequence closest to the selection
    for i, seq in enumerate(sequences):
        if seq[0]["bbox"][4] > frame_number and seq[-1]["bbox"][4] < frame_number:
            continue

        for s in seq:
            if s["bbox"][4] != frame_number:
                continue

            dist = abs(x-s["bbox"][0]) + abs(x-s["bbox"][2]) + \
                abs(y-s["bbox"][1]) + abs(y-s["bbox"][3])
            if dist < closest_dist:
                closest_dist = dist
                closest_seq = i

    (...) # Returning if not found, omitted for brevity, see source
    # Getting samples from the closest identified sequence
    seq_samples = []
    if len(sequences[closest_seq]) < samples:
        seq_samples = sequences[closest_seq]
    else:
        intv = len(sequences[closest_seq]) // samples
        if intv < 1:
            intv = 1

        for k in range(samples):
            seq_samples.append(sequences[closest_seq][intv*k])

    # Comparing the samples to other sequences to identify sequences with the same
    ↪  face
    (...) # Omitted for brevity, see source
    if remove_known:
        return [s for i, s in enumerate(sequences) if i not in identical]
    return [s for i, s in enumerate(sequences) if i in identical]
```

Listing 14: Filtering selected faces[15]

[15]https://github.com/ErlendF/face_blur/blob/b9efa56c27fd69dedb01c378cff631fc2d045ac6/videoface/filter_faces.py

Ideally, a user interface should be used to simplify the selection of the time and location of the face. Potentially, such an interface could allow a user to select a face by simply clicking on it in the video editor. However, it does not make sense to make a new interface separate from CuttingRoom's existing user interface. CuttingRoom's user interface has not been made available for the thesis and an example user interface will therefore not be made.

3.1.8 Smoothing sequences of bounding boxes

The machine learning models used throughout the thesis have nearly always been intended for image recognition, *not* video. As a result, the bounding boxes found by the facial detection may move around a bit arbitrarily from frame to frame, even if the person is standing nearly still[16]. This is of no consequence to the anonymization of the subjects, but may be distracting to the viewer and becomes unnecessary noise in the video.

This effect is significantly reduced by not processing every frame and interpolating the non-processed frames, like discussed in Sections 3.1.3 and 3.1.5. However, this cannot be done for every shot of the video as the shots with changes in the faces shown in the frame need to be processed in more detail0. Therefore, it would be beneficial to reduce the unnecessary movement of each bounding box by smoothing out the minor differences between each frame. The overall movement of each person still needs to be captured to correctly blur their face, though ideally with as little unnecessary movement as possible.

Similarly to when interpolating the bounding boxes as discussed in Section 3.1.5, each of the four values composing the bounding box can be handled separately. Listing 15 shows a rudimentary implementation of the *unweighted moving average smoothing* algorithm, one of the simplest smoothing algorithms. In the algorithm, each value is replaced by the average of a certain number of adjacent points. This effectively lessens the impact of local changes on the overall function.

[16]Example: `https://youtu.be/nSvN24R_wRU`

```python
def avg_smoothing(seqs, smoothing_width=9):
    df = smoothing_width//2

    for i in range(len(seqs)):
        # Getting a copy of the sequence only containing bounding boxes to make it a
        ↪  numpy array
        seq_copy = np.array([s["bbox"].copy() for s in seqs[i]])

        # Smoothing each point of the bounding box individually
        for j in range(len(seqs[i])):
            first = max(0, j-df)
            last = min(len(seqs[i]), j+df)
            seqs[i][j][0] = np.mean(seq_copy[first:last, 0])
            seqs[i][j][1] = np.mean(seq_copy[first:last, 1])
            seqs[i][j][2] = np.mean(seq_copy[first:last, 2])
            seqs[i][j][3] = np.mean(seq_copy[first:last, 3])

    return seqs
```

Listing 15: Example implementation of unweighted sliding-average smoothing[17]

Scikit-fdas smoothing functions[18] could alternatively also be used fairly easily. Unfortunately, at the time of writing there is an issue with one of scikit-fdas dependencies which causes issues when using Python 3.10[19]. The current implementation depends on the `key` parameter added to the `bisect` library in Python version 3.10[20] and can consequently not use an earlier version of Python. The implementation could be changed to not require this parameter, but it seems unnecessary as the scikit-fda library will likely be updated to support Python version 3.10 after some time either way.

3.1.9 Handling false negatives

When someone is moving around, their face often changes pose over the course of a shot. Certain facial poses are often more difficult to detect than others. For example the side of a face is generally more difficult than a head-on shot of a face, as discussed in Section 4.1. Therefore, there are often periods where a model fails to detect faces in videos where

[17]https://github.com/ErlendF/face_blur/blob/b9efa56c27fd69dedb01c378cff631fc2d045ac6/smoothing/average.py

[18]https://fda.readthedocs.io/en/latest/modules/preprocessing/smoothing.html

[19]https://bytemeta.vip/repo/GAA-UAM/scikit-fda/issues/433, https://github.com/jdtuck/fdasrsf_python/issues/20

[20]https://docs.python.org/3/library/bisect.html

the person may have turned to face away from the camera. Their face may, however, still be clearly recognizable from the side and still would be considered personal data. Their face should be detected, but is not, and would therefore be considered a *false negative.*

In these cases, the ideal solution would be to train the model to recognize faces in a greater set of poses, but this may not always be feasible. Some of the models are already very good at detecting faces under an impressive number of conditions, but not all of them. Therefore, the rest of the processing should account for potential false negatives in the data as best as possible. Where false negatives are identified, it would be preferable to interpolate the bounding boxes where the face is not detected (see Section 3.1.5).

After the faces have been detected, the only indication of potential false negatives are gaps in the facial sequences. If a face is detected in a given shot for several seconds, but there are a few intermittent frames where the face is not detected, these are very likely false negatives and the frames should be interpolated. Unless there is a cut, people generally do not disappear for only a few frames at a time.

When dynamically processing the video (see Section 3.1.3), false negatives may be ignored entirely if they fall in the interval of frames that are not processed at all. However, if one of the frames with a false negative *does* get processed, this will cause the entire interval to be processed rather than interpolated, which will slow down the video processing.

Unlike detecting false positives, discussed in Section 3.1.10, the certainty of a detection cannot be used when detecting false negatives. Since the face is not detected, there is no predicted bounding box, and consequently no certainty to use for the detection of false negatives.

The most reliable method seems to set a maximum missing frame number, and interpolating the missing bounding boxes in the frames of a facial sequence. When making facial sequences (see Section 3.1.4), the facial sequences are initially constructed by matching bounding boxes of adjacently processed frames. If there are frames between two detections of the same face where the face is not detected, they will not initially be made into the same facial sequence. However, after making the initial sequences, entire sequences which likely belong to the same person are combined to form larger sequences. When combining the sequences, a maximum number of frames between the combined sequences be defined. It does not make sense to combine two sequences at entirely separate points in the video. Therefore, by setting the threshold sufficiently large; false negatives may fall inside of a single facial sequence and may easily be interpreted, thereby filling in the missing bounding boxes.

3.1.10 Handling false positives

Handling false positives may be very difficult, and may be detrimental to the anonymization of the solution as there is a trade-off between false positives and false negatives. The lower the threshold for detecting faces, the fewer false negatives there will be, but also the more false positives. The opposite is also true. The priority depends on the video. In some of cases, like a football match, the anonymization may just be a precaution. However, in other cases, like a criminal case, it may be far more important that any actual faces get blurred rather than any false positives are removed. False positives may be distracting, but they do otherwise no harm.

Observed cases of false positives

The most prevalent type of false positive seems to be false positives where the detected area looks nothing like a face, such as a hand, an arm, a wall decoration etc. These do not appear to have any meaningful reason for being detected, and is likely a coincidentally good match for the features of a face the facial detector is looking for.

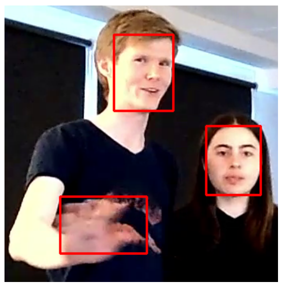
Figure 3.6: False positive without apparent reason[21]

Statues and faces in pictures displayed in videos may also be detected as faces. This is not strictly a false positive, since it is actually correctly detecting a face, but it would be reasonable to assume that not all pictures need to be blurred. Famous paintings or pictures in the public domain likely do not require blurring. However, pictures of protected individuals and their families should obviously be removed. A model could potentially be trained to differentiate between people and pictures, but this seems unnecessary for such a specific scenario and would fall outside the scope of the thesis. Similarly, copyrighted portraits should potentially be blurred, but this is also outside the scope of the thesis.

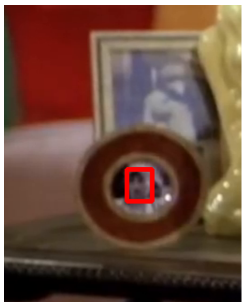
Figure 3.7: Face detected in photography

[21]Example from: `https://youtu.be/73XS75-tdYQ`

Non-human faces may also sometimes be detected. It would be reasonable to assume that non-human faces should not need to be blurred. The Personal Data Act use the terminology *"person"* when discussing subjects whose data should be protected [52], but whether or not anything outside of humans should be considered to be a "person" is more of a philosophical question and is undoubtedly outside the scope of this thesis. Therefore, for the purposes of this thesis, detection of non-human faces will be considered a false positive and handled as such.

Possible solutions

One way of handling false positives would be to use the *certainty* of a prediction to evaluate how likely it is to be an actual face. However, not all models provide certainty for their predictions, so this method is dependent on the model. Unfortunately, even if the model *does* provide a certainty score, it may not be very useful for identifying false positives. From testing the MTCNN detector (see Section 3.2.2), the certainty scores for false positive sequences were indistinguishable from true positives. It would therefore be infeasible to use this score algorithmically or in a machine learning model to remove the false positives.

Another way of handling false positives would be to remove short sequences of faces that are not related to any other sequences in the scene. False positives appear to often only be present for a few frames at a time, and only in short time spans. This appears to be their most distinguishable characteristic, but it may depend on the facial detection model being used. If this is the case, they may be removed by setting a minimum number of frames a face need to be detected for the sequence to be considered correctly identified. However, this would also result in faces that are only visible for a few frames in the video *not* being blurred. Consequently, it is a trade-off between false positives and false negatives. This method of removing false positives is implemented simply by checking the length of facial sequences, and removing sequences shorter than a specified minimum[22]. Alternatively, it could be made to check the amount of frames being detected *in a row* rather than the total frames being detected to identify false positives. For this purpose, an option to set a minimum sequence length was added to the function making the facial sequences[23] (see Section 3.1.4).

[22]`https://github.com/ErlendF/face_blur/blob/15b62ce828e6662d2838fc1d4b45052504822b32/videoface/filter_faces.py#L127`

[23]`https://github.com/ErlendF/face_blur/blob/15b62ce828e6662d2838fc1d4b45052504822b32/videoface/facial_sequences.py`

3.1.11 Blurring

When blurring faces in videos, the faces need to be blurred in each individual frame of the video and the blur needs to be sufficiently strong not to reveal the identity of the individual being blurred. Therefore, it is necessary to read every frame, blur the selected faces, and write the image back to the disk. In these examples, only the face is blurred. However, the entire head may easily be blurred to better protect subjects anonymity by slightly expanding the blurred area, and moving it a bit higher.

Likely the easiest way to blur a face, is to simply blur the specific region of the frame delimited by the bounding box[24]. This produces a square blur, which can hardly be described as appealing. An example of this is shown in Figure 3.8.

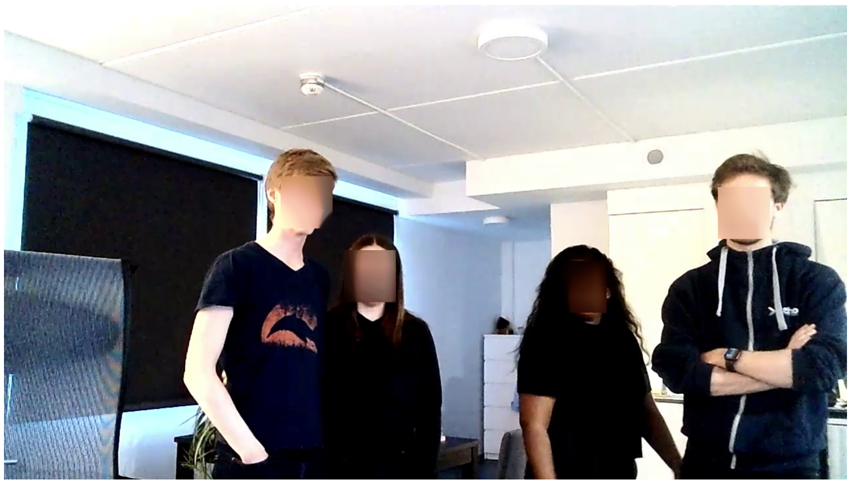

Figure 3.8: Square blurring

By splitting these bounding boxes into smaller regions and taking the mean of each region individually[25], a pixelated blurring effect may be achieved, as shown in Figure 3.9.

[24]See `https://github.com/ErlendF/face_blur/blob/123704872683c16b0a42eeec13cf86ef1e9b5216/blur/blur.py#L38`

[25]See `https://github.com/ErlendF/face_blur/blob/88515a05db7e48381c68e9e6ddbcdbe12c3df6ae/blur/blur.py#L45`

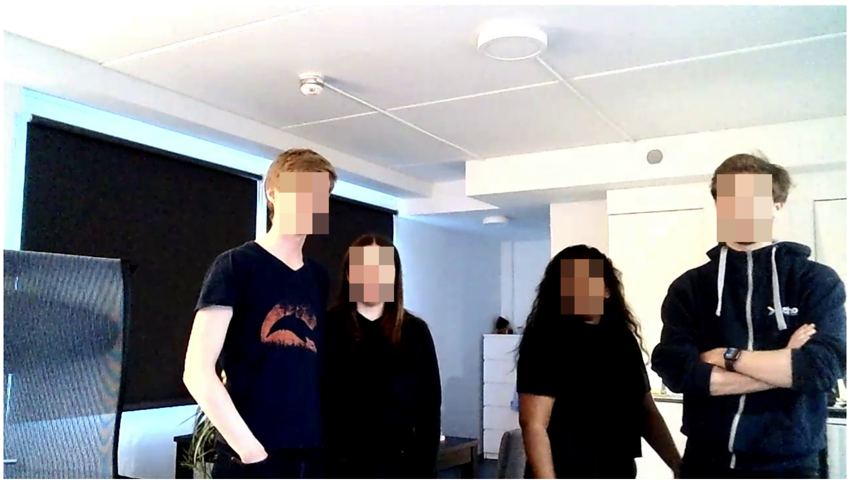
Figure 3.9: Pixelated blurring

Another way to blur a face would be to blur a circle around the bounding box rather than only the area delimited by the bounding box[26]. To make such a round blur, a mask needs to be made marking a circle over the desired locations around the bounding boxes which should be blurred. This mask may then itself be blurred in order to make the transition gradual between blurred and non-blurred regions, as shown in Figure 3.10. Having a gradual transition between the blurred and non-blurred areas of the frame makes the blurring a lot less noticeable compared to the hard-cut edges of the square blur example.

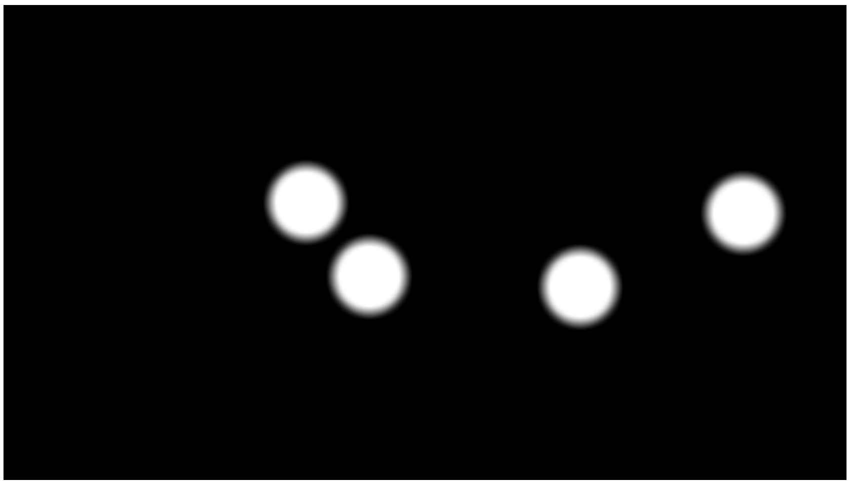
Figure 3.10: Blurring mask, blurred to make transition gradual

[26]See: `https://github.com/ErlendF/face_blur/blob/123704872683c16b0a42eeec13cf86ef1e9b5216/blur/blur.py#L18`

Lastly, a copy of the frame is fully blurred as shown in Figure 3.11, and the blurred and non-blurred versions of the frame are blended using the mask. In each location where the mask is white, the blurred image is used, where it is black, the original image is used. This produces circular blurs around each bounding box with a gradual transition, as shown in Figure 3.12.

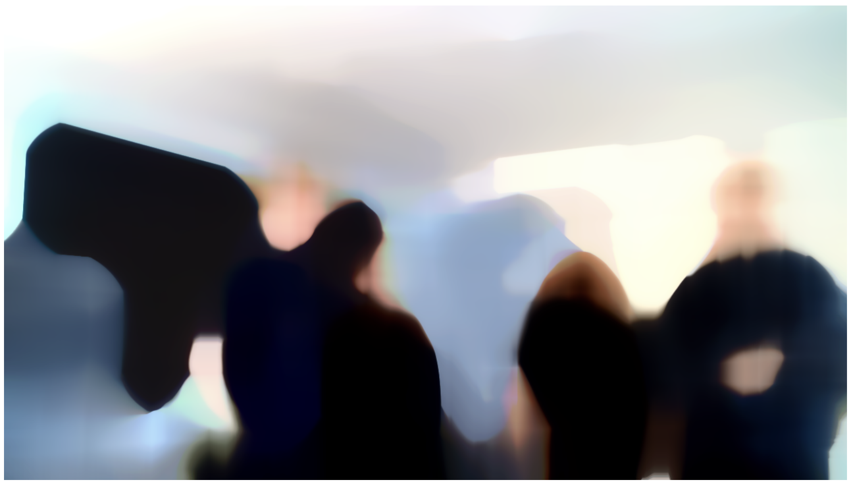

Figure 3.11: A fully blurred version of the image

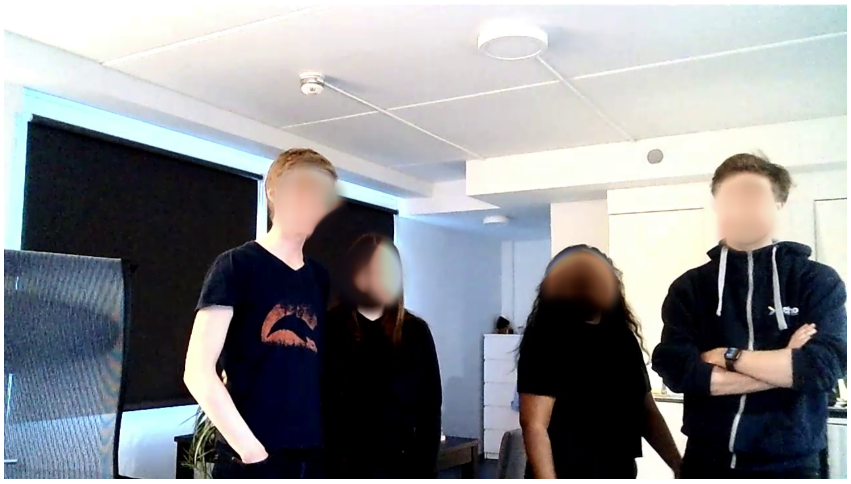

Figure 3.12: Round blurring

3.1.12 Sequence models

Sequence models have great potential for handling video, and could potentially resolve a lot of issues surrounding false positives, false negatives and noisy bounding boxes that would otherwise have to be handled algorithmically (see Sections 3.1.3, 3.1.5 and 3.1.8). For this purpose, a *many to many* model would be most appropriate, since we need to use the entire set of bounding boxes, and all of them should be handled appropriately. RNN, LSTM and GRU Pytorch models were used for this thesis.

Ideally, the sequence model should take the bounding boxes of detected faces as input and output only *true positive* bounding boxes, with missing bounding boxes approximated and less noise in the movement of existing bounding boxes. To do so, the bounding boxes need to be organized into sequences for each face (see Section 3.1.4). The sequence should start at the earliest detected instance of the face and end at the last. The detected bounding boxes should be formatted appropriately, and missing bounding boxes should have an appropriate value to signify them as missing values. The format of the bounding box sequences used in the thesis is *(B x S x d)* where B is the batch size, S is the sequence length and d is the dimension of the bounding box. Typically, d would be of size 4, representing x_1, y_1, x_2 and y_2 which form the corners of a rectangle. It could also include the *certainty* of the model for the detected bounding box. This is the same format presented as presented in Chapter 3, but with the facial features removed. Additionally, all sequences need to be the same length to form a Pytorch tensor. Therefore, the shorter sequences are padded using the `torch.nn.utils.rnn.pad_sequence` function[27].

Handling false negatives and smoothing sequences of bounding boxes

No dataset was made for this thesis. Consequently, there is no true value to compare with the predictions when calculating the loss of a prediction. An appropriate sequence of bounding boxes could be made manually to be used as a true value, but this is work intensive and outside of the scope of the thesis. Therefore, it would be ideal to calculate the loss based on how well the predicted bounding boxes fit together and how closely they relate to the input bounding boxes. Distance between adjacent bounding boxes should be penalized in order to make a smoother movement of the bounding boxes and avoid minor, unnecessary movement. Furthermore, penalizing the distance between adjacent bounding boxes should incentivize an appropriate approximation of missing bounding

[27]https://pytorch.org/docs/stable/generated/torch.nn.utils.rnn.pad_sequence.html

boxes in order to minimize the loss. The distance between input bounding boxes with missing values and predicted bounding boxes is therefore not penalized.

Distance between the predicted and input bounding boxes is required in order to follow the faces movement. If the distance between the predicted and input bounding boxes was not penalized, there would be no incentive for the model to move the bounding boxes around at all. Simply predicting 0 for all values of the bounding box would in this case result in a loss of 0 since the distance between adjacent bounding boxes would also be 0.

An example loss function is shown in Listing 16, where the loss of the batch is calculated as the sum of the difference between the input bounding box and the predicted bounding box plus the difference between the given bounding box and its previous, adjacent bounding box. The squared loss is to allow for minor movement of the bounding boxes. However, large movement should be avoided whenever possible. In the listing, index 4 of the bounding box is used for the certainty. If the certainty is 0.0, it means that there was no bounding box there originally and it should be approximated. Therefore, distance to the original value is not penalized since it was a missing value.

```
def sq_frame_diff(x1, x2):
    return torch.add(torch.add(
        torch.pow(torch.abs(torch.sub(x1[0], x2[0])), 2),
        torch.pow(torch.abs(torch.sub(x1[1], x2[1])), 2)),
    torch.add(
        torch.pow(torch.abs(torch.sub(x1[2], x2[2])), 2),
        torch.pow(torch.abs(torch.sub(x1[3], x2[3])), 2)))

def sep_loss_func(x, pred):
    sum_diff = torch.tensor(0)
    for b in range(x.shape[0]): # batches
        # Remaining predictions after the end of the sequence are not penalized
        seq_end = len(x[b]) - next(i for i, f in enumerate(reversed(x[b])) if
        ↪  torch.round(f[4]) != 0.0)
        for i in range(seq_end):
            if torch.round(x[b, i, 4]) != 0.0:
                sum_diff = sum_diff.add(sq_frame_diff(pred[b, i], x[b, i]))
            if i != 0:
                sum_diff = sum_diff.add(sq_frame_diff(pred[b, i], pred[b, i-1]))

    return sum_diff.sum()
```

Listing 16: RNN loss function

Handling false positives

Similarly to handling false positives without a sequence model, detecting and removing false positives was difficult. In order to train the model to be able to distinguish between true and false positives, the loss needs to be adjusted according to the models performance. Consequently, the loss function needs to be designed reduce the loss when the model correctly classifies true and false positives, and penalize incorrect classifications. Normally, this would be done using a dataset where true and false positives are known, and incorrect classifications are penalized. The model would then learn patterns of the features of true and false positives to distinguish them. However, making such a loss function without knowing the true and false negatives beforehand is less straightforward.

The same methods for handling false positives as discussed in Section 3.1.10 could possibly be utilized when making the loss function. The sequence model could potentially receive the facial detection models certainty score as an additional input and output its own certainty score, or alternatively be trained to otherwise differentiate bounding boxes which should be removed. Unfortunately, as also discussed in Section 3.1.10, there was no discernible difference between the certainty scores of true and false positives. Consequently, it is unclear both how the model may interpret the certainty and how the loss should be calculated. Furthermore, the sequence models performance also seemed to deteriorate when passing the certainty as an additional input.

The model could also be penalized for outputting valid bounding boxes for sequences shorter than a certain threshold. However, the bounding boxes may in this case simply be removed before passing them to the model instead.

The potential of a bidirectional model

A unidirectional model can only base its predictions on preceding (or succeeding) bounding boxes. This may cause issues with rapid movement and approximating missing values. By only using past bounding boxes, there would essentially be no difference between sequences with a few missing frames and sequences which simply end. Furthermore, to interpolate the bounding box of a missing value between two known values is fairly straightforward, as discussed in Section 3.1.5. However, this becomes far more difficult if only one of the two bounding boxes on either side is known. It would then be impossible to know the direction of movement accurately. The movement could potentially be estimated based on the previous bounding boxes, but this cannot be as reliable as when both bounding boxes are known.

Using a bidirectional model could potentially solve both of these issues. Using sufficiently large sequences, the model should be able to distinguish short intervals of missing bounding boxes and approximate these. Additionally, it could utilize both the preceding and succeeding bounding boxes to make a smoother sequence of bounding boxes.

Separating responsibilities

The original intention was to make a single model which would remove false positives, approximate false negatives and make a smoother sequence of bounding boxes. The smoothing of the sequence and approximating the false negatives are quite closely related and it makes sense for them to be done by the same model. However, removing false positives is a different task entirely. It might make more sense for this to be done by another model, or alternatively an algorithm. By separating the tasks, each task may be simplified and possibly make the result more reliable. Each model could in this case be trained independently.

3.2 Facial detection, alignment and representation

The section discusses a selection of tools and models for facial detection, alignment and representation explored. These are discussed together as several of the projects discussed perform more than one of these tasks. Facial classification is based on the facial features representations made, and is discussed in Section 3.1 where applicable, particularly in Sections 3.1.4 and 3.1.7.

3.2.1 Retinaface

Retinaface is a single-stage face detector which can both detect and align faces of various scales by taking advantage of joint extra-supervised and self-supervised multi-task learning [38]. The original implementation is based on MXNet and is the facial detection module of the Insightface project[28]. It is released under the MIT license, but the pre-trained models provided with the library are only available for non-commercial research purposes [45]. However, it has been reimplemented in TensorFlow[29], and later simplified and made available as a pip package[30] which is available for commercial use under the MIT license[31] [58]. This pip package is again used by Serengil/Deepface and implemented in this thesis through it (see Section 3.2.6).

3.2.2 Multi-task Cascaded Convolutional Networks

Multi-task Cascaded Convolutional Networks (MTCNN) is a framework designed to both detect and align faces in images [73]. It uses a cascaded structure with three stages of deep convolutional networks that predict face and landmark locations in a coarse-to-fine manner. In the first stage, candidate windows are produced through a fast *Proposal Network*. Then, the candidates are refined through a *Refinement Network*. Lastly, the final bounding boxes and facial landmark positions are produced by the *Output Network*.

This method works well and is highly accurate. In this thesis, it is used in combination with ArcFace as shown in Section 3.3.2, and is also one of the facial detectors offered by the Serengil/Deepface framework (see Section 3.2.6).

[28]`https://insightface.ai/`
[29]`https://github.com/StanislasBertrand/RetinaFace-tf2`
[30]`https://pypi.org/project/retina-face/`
[31]`https://github.com/serengil/retinaface`

3.2.3 Pigo

Esimov/Pigo is a face detection, pupil/eyes localization and landmark points detection library written by Endre Simo [61]. It is based on the *Pixel Intensity Comparison-based Object detection* paper by Nenad Markuš et al. [51], which describes a method for visual object detection based on an ensemble of optimized decision trees organized in a cascade of rejectors. This method is notable for how fast it is compared to other methods such as HOG and CNNs.

Since this library is written in Go, it is incompatible with the Python implementation of the pipeline presented in this thesis. It may be used to predict the bounding boxes, which may be used with a lot of the implementation when pre-computed, but it cannot feasibly be used with the dynamic processing (see Section 3.1.3). It could potentially be called as a standalone executable, but this is far too slow for regular use, and the detection is not nearly accurate enough to warrant this, as shown by this example: `https://youtu.be/hQ7EhRiJKdo`.

3.2.4 ArcFace

ArcFace is a deep convolutional neural network which uses additive angular margin loss to obtain highly discriminative features for face recognition [37]. It is one of the face recognition modules of the Insightface project[32]. Like the licensing of Retinaface (see Section 3.2.1), the original implementation is provided under the MIT license, but the pre-trained models are only available for non-commercial research purposes only [45]. However, the Open Neural Network Exchange (ONNX) model zoo offers a pre-trained model of ArcFace under the Apache 2.0 license which does permit commercial use. It is also available through the serengil/Deepface framework under the MIT license (see Section 3.2.6). An example SageMaker model was made for serving Batch Transform Jobs (see Section 3.3.2). The example is available here: `https://github.com/ErlendF/face_blur/tree/main/models/arcface-mtcnn-batch-transform`.

[32]`https://insightface.ai/`

3.2.5 Dlib

Dlib is an open-source C++ toolkit offering machine learning algorithms and various tools [49]. Most notably, the machine learning algorithms include facial detection, facial alignment and facial recognition features. For the facial detection, there are two main algorithms of note; convolutional neural network[33] and histogram of oriented gradients[34]. For the facial recognition, a ResNet model is used[35].

Though Dlib is principally a C++ library, a number of its tools is also available for python applications [50]. These tools fortunately includes the facial detection and facial recognition methods. Furthermore, these methods have been used to make a wrapper for the Dlib functions named *ageitgey/face_recognition* to simplify their use [44].

Listing 17 shows how the *face_recognition* python library may be used to detect and locate faces in a batch of images and make facial features representations for each detected face. This example uses the Dlib CNN model rather than the HoG method. The CNN model was in this case chosen because it is more accurate than the HoG method, though a bit slower [56]. The accuracy was here prioritized since there is little point of blurring someone if their identity is revealed anyway. This is further discussed in Chapter 4.

[33]`http://dlib.net/python/index.html#dlib_pybind11.cnn_face_detection_model_v1`
[34]`http://dlib.net/python/index.html#dlib_pybind11.get_frontal_face_detector`
[35]`http://dlib.net/python/index.html#dlib_pybind11.face_recognition_model_v1`

```python
def dlib_process(img_names, img_nrs, frames=None):
    if len(img_names) == 0:
        return {}

    imgs = []
    if frames is None:
        for filename in img_names:
            imgs.append(read_frame(filename))
    else:
        for nr in img_nrs:
            imgs.append(frames[nr])

    locs = batch_face_locations(
        imgs, number_of_times_to_upsample=0, batch_size=len(imgs))

    faces = {}
    for ls, img, inr in zip(locs, imgs, img_nrs):
        img_faces = []
        fs = face_encodings(img, known_face_locations=ls, model="large")
        for l, f in zip(ls, fs):
            img_faces.append(
                {'bbox': [l[3], l[0], l[1], l[2], inr], 'feat': f.tolist()})

        faces[inr] = img_faces

    return faces
```

Listing 17: Using the Face Recognition library built on Dlib to detect and recognize all the faces in the list of images[36]

3.2.6 Deepface

Serengil/Deepface[37] is a face recognition and facial attribute analysis framework wrapping state-of-the-art models like VGG-Face, Google FaceNet, Facebook DeepFace and ArcFace [59, 60]. The framework also provides some functionality to detect and preprocess images of faces before feeding them into the facial recognition models. For facial detection, it offers a choice between Retinaface, MTCNN, OpenCV, SSD, Dlib and MediaPipe. In the thesis, the Facenet512 model[38] is used for facial recognition and Retinaface is used as the facial detection backend (see Section 3.2.1). These may easily be replaced by changing two lines of code. The serengil/Deepface framework itself is available under

[36]https://github.com/ErlendF/face_blur/blob/68b99220241526885deedfde878c55f8d4bfd449/videoface/dlib.py
[37]https://github.com/serengil/deepface
[38]https://github.com/davidsandberg/facenet

the MIT license[39] which notably permits commercial use, copying, modification, publishing and distribution [59]. However, it also wraps various models which have their own licenses. Thankfully, only the VGG-Face model appear to have a license which does not permit commercial use[40]. A full list of the model licenses is included in Appendix E.

The Serengil/Deepface framework does not natively provide the functionality to detect all faces in an image and get their locations and facial representation. However, as it already uses facial detection models which do detect all faces in the processed image, and it does get the facial representation for a single face, the code may be adapted to return all detected faces and representations for each face. The `detect_faces`[41] function used by the higher level functions like `verify`[42] (used to compare two faces) and `represent`[43] (used to make a facial features representation) in the library simply discards every detected face except for the first one returned, and discards the rest. For the Deepface framework to be utilized in the pipeline, this was adapted to keep all the detected faces returned, and make a facial features representation for all of them. The prediction of the representations were batched in order to increase performance.

The framework is mainly powered by TensorFlow and Keras [59], and TensorFlow code and *tf.keras* models[44] can run both on CPUs and on a single GPU without any code changes required [1]. Using a CPU, the Deepface implementation of the facial processing is significantly slower than using the dlib implementation (see Section 3.2.5). However, the Deepface implementation is far more accurate, and when running on a GPU, it is also notably faster. This is further discussed in Chapter 4.

Although multiple projects are named "Deepface"; the serengil/Deepface framework will be referred to simply as "Deepface" for the sake of simplicity from this point onwards.

[39]https://github.com/serengil/deepface/blob/master/LICENSE
[40]https://www.robots.ox.ac.uk/~vgg/software/vgg_face/
[41]https://github.com/serengil/deepface/blob/5d767e2d493b47b914a257b6dbf14dd3d472eddf/deepface/detectors/FaceDetector.py#L35
[42]https://github.com/serengil/deepface/blob/5d767e2d493b47b914a257b6dbf14dd3d472eddf/deepface/DeepFace.py#L70
[43]https://github.com/serengil/deepface/blob/5d767e2d493b47b914a257b6dbf14dd3d472eddf/deepface/DeepFace.py#L721
[44]https://www.tensorflow.org/api_docs/python/tf/keras

3.2.7 AWS Rekognition

AWS Rekognition is a high level, managed AWS service which offers various features such as facial detection and facial recognition in images and video [25]. It is a managed service rather than a machine learning model, but may be used in place of the facial detection and recognition models discussed previously. In order to recognize specific faces, the faces have to be indexed into a server-side container known as a *collection* [13]. Then, the stored media may be searched to look for the indexed faces. AWS Rekognition will then detect faces in the media and indicate which faces were matched to each of the faces in the stored collection. The detected faces will also be given an index which is supposed to be unique to the specific face.

Python scripts are provided in the GitHub repository[45] to make a collection, index faces, start a face search job, collect the results and parse the results: `https://github.com/ErlendF/face_blur/tree/main/scripts/rekognition`. The output results of the AWS Rekognition face search is a list of faces detected in the video with their index, timestamp (in milliseconds) and which faces in the collection they were matched to (if any)[46]. When parsing the results, this information is used to make sequences of faces, which may be handled similarly to the rest of the pipeline. AWS Rekognition only processes one frame every half a second, and the results therefore need to be interpolated to get bounding boxes for every frame, similarly to how the interpolation is needed with dynamic processing (see Section 3.1.3).

The scripts are basic examples made to test the feasibility of using AWS Rekognition to replace the facial detection and recognition parts of the pipeline and should be expanded upon if used in a production setting such that the results will be automatically collected and parsed once the face search job finishes.

[45]The same repository discussed previously: `https://github.com/ErlendF/face_blur`

[46]Full description of response syntax: `https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/rekognition.html#Rekognition.Client.get_face_search`

3.3 Orchestration

The orchestration explored in the thesis focuses on AWS since this is the cloud platform used by CuttingRoom. However, the proposed pipeline may be used in any environment supplying its dependencies, including Google Cloud Platform and Microsoft Azure. The exploration of the AWS services focused primarily on AWS SageMaker, a fully managed machine learning service which offers a variety of tools and services for building, training and deploying machine learning model [19]. Two of the services offered by AWS SageMaker are explored for the purposes of running inferencing using the full pipeline in a cloud environment.

3.3.1 AWS SageMaker Processing Job

Amazon SageMaker Processing Jobs are used for analysing data and evaluating machine learning models [24]. This may include feature engineering, data validation, model evaluation and model interpretation.

When using a processing job, the underlying infrastructure is fully managed by Amazon. The user needs to specify the *ECR image* to use, the *instance count*, *instance type* and *S3 bucket* containing the data which should be processed [11]. This image is used to start the number of instances of the given type specified. Amazon SageMaker Processing starts by copying the data from the specified S3 bucket to the instances. SageMaker then starts the container, which should read the data from the instance's local storage, process it and write the result back to local storage. Consequently, it is fairly simple to adapt a simple example, like the one shown in Listing 20 in Section 3.4.1, to run in a Processing Job as it still reads and writes to local storage. After either all or parts of the processing finishes, Amazon SageMaker copies the result back to an S3 bucket.

Since both the instance number and type are easily customizable, this makes it really easy to scale jobs both vertically and horizontally to speed up processing. For larger jobs, larger instances may be used to speed up the process, thereby scaling vertically. Alternatively, several instances may be used to process the job, and the data is distributed among them, thereby scaling horizontally. When starting the processing job using input from S3, the user is given options to choose between distribution the data to the instances `ShardingByS3Key` or `FullyReplicated` [12]. However, due to the inherent requirement for an uninterrupted sequence of frames when utilizing the interrelatedness of frames, sharding the data by S3 key may not be suitable for processing, depending on how the data is sharded. Though there is no definitive documentation, AWS appears to use hashing of the S3 key to distribute the data evenly across the instances, in which case, the data will not be sequential [30].

When fully replicating the data across instances, it would be beneficial to be able to identify each instance such that each instance could process a separate, sequential part of the data and the job could be manually distributed. Unfortunately, there is seemingly no way for an instance to identify itself. The number of instances could be passed to the processes using for instance environment variables, but this could not identify each instance. Additionally, replicating the entire dataset to each instance would theoretically work fine for a smaller dataset if it was possible to manually distribute the work, but for larger datasets, this would be infeasible. Processing all the frames of an up to eight hour video would require a great deal of space, not to mention processing power, so fully replicating it to several instances would be incredibly costly and wasteful.

When testing the processing jobs, the processing image used needed to be custom made. The Docker container was based on the *nvidia/cuda*[47] docker image to allow for running the models using GPU acceleration. The latest version of the development image based on Ubuntu with Cudnn 8 was used. An Docker container example is available here: `https://github.com/ErlendF/face_blur/tree/main/models/face-blur-processing-job`. It may be used with the scripts provided scripts to automatically start a processing job.

3.3.2 AWS SageMaker Batch Transform

AWS SageMaker Batch transform is used to, for instance, preprocess datasets or perform inferencing on large datasets [20]. When using batch transform, the data which should be processed needs to be stored in S3. Amazon SageMaker sends this data from S3 over HTTP POST requests to the processing container [4]. The container then processes the request and returns the result. Amazon SageMaker receives the result and stores it in a specified location in S3 and names the object the same as the input, but appending the extension `.out`. The returned result needs to be in a JSON format.

[47]`https://hub.docker.com/r/nvidia/cuda`

Because the multiple image files cannot be combined in a single request[48], this is the equivalent of processing every frame like discussed in Section 3.1.2. The output may later be combined and handled like discussed for processing every frame. However, this would require an AWS Lambda, AWS Batch Job[49] or similar to be triggered after the batch transform job has finished.

To use batch transform jobs for preprocessing images to detect and make a representation for the faces in each image, the container needed to be custom made. Adapting a simple example such as the one shown in Listing 20 in Section 3.4.1 therefore requires a bit more work than for an AWS Processing Job. The container needs to listen for HTTP GET requests to the `\ping` endpoint and HTTP POST requests to the `\invocations` endpoint, both on port `8080` [4]. The `\ping` endpoint is used by Amazon SageMaker as a health check for the container, and the actual processing requests are sent to the `\invocations` endpoint. Ideally, the container should also listen to the `\execution-parameters` endpoint, which allows for setting various tuning parameters for a job during runtime.

The simplest way of fulfilling these requirements is to use the *SageMaker Inference Toolkit*. It is built on AWS Labs' Multi Model Server and handles various aspects of serving models, such as the endpoints mentioned [17]. It also starts the maximum number of possible concurrent models automatically, defined by the `MaxConcurrentTransforms` parameter. When using the toolkit, the user needs to implement a *handler* and package entire program in a docker container. An example handler using MTCNN (see Section 3.2.2) for facial detection and ArcFace (see Section 3.2.4) to make facial representation is shown in Listing 18. First, the handler is initialized if it has not been already. Then, when a request is received, the `inference` function is ultimately called. It parses the image from the request and passes it to the MTCNN and ArcFace models to get the locations and facial features representation of each face. Lastly, the result is formatted and returned. The toolkit will JSON encode the response and return it to Amazon SageMaker.

[48]There are batch strategies, but this requires individual files to be split, which is infeasible for images [5]

[49]Not to be confused with an AWS SageMaker Batch Transform Job: `https://aws.amazon.com/batch/`

```
class Handler(object):
    (...) # __init__ function, omitted for brevity, see source
    def initialize(self, context):
        (...) # setting variables, omitted for brevity, see source
        self.detector = MtcnnDetector(
            model_folder=model_path,
            ctx=ctx,
            num_worker=1,
            accurate_landmark=True,
            threshold=det_threshold,
        )
        self.model = get_model(ctx, full_model_path)

    def inference(self, data):
        resp = []
        for input in data:
            body = input.get("body")
            img = Image.open(io.BytesIO(body)).convert("RGB")
            input = get_input(self.detector, np.array(img)[:, :, ::-1].copy())
            if input is None:
                resp.append([])
                continue

            inp, bboxes = input
            feats = get_feature(self.model, inp)
            faces = []
            (...) # Transforming data, omitted for brevity, see source
            resp.append(faces)
        return resp
```

Listing 18: Batch transform handler using ArcFace and MTCNN[50]

The entire program, including all classes and other functions not shown in Listing 18, is then packaged into a docker image with the required dependencies[51]. The docker image is used to make a SageMaker model as discussed in Section 3.3.3.

[50]https://github.com/ErlendF/face_blur/blob/b9efa56c27fd69dedb01c378cff631fc2d045ac6/models/arcface-mtcnn-batch-transform/model/handler.py

[51]See: https://github.com/ErlendF/face_blur/tree/main/models/arcface-mtcnn-batch-transform

3.3.3 Automatization

All parts of the infrastructure deployment has been automated using Terraform[52] and Python scripting[53]. This is done to ensure consistency, reproducibility and ease of use. Several ways of running the project in a cloud environment have been explored, and it is up to CuttingRoom and anyone else who uses the project how they would like to deploy and utilize it to fulfil their needs.

The Terraform code was made to deploy the following resources to simplify most deployments. These resources facilitate multiple ways of deploying the project, and should be adjusted for how the project is actually deployed. This includes the IAM policies which are currently fairly broad to assist development and should be restricted to a least privilege level in a production environment.

- An ECR repository to store the containers.
- A S3 bucket to store the data. Public access to the bucket is blocked.
- A SageMaker Notebook instance for development.
- An IAM role with various permissions to the ECR repository, S3 bucket and various other permissions which may be required for development and inferencing.

Two Python scripts were also made to deploy AWS SageMaker Processing Jobs and Batch Transform Jobs. These scripts require various variables such as the name of the ECR repository and the IAM role to use for the processing. These may be retrieved from the output of the Terraform code, and an example configuration is provided[54]. The scripts then automatically start the job with the given configuration. This makes it easy to run the jobs, and makes it possible to automatically start based on end-user interactions.

Lastly, a group of Python scripts were made to use AWS Rekognition for facial detection and recognition as discussed in Section 3.2.7. The scripts for AWS Rekognition are useful for the same reasons as the rest of the automation, but they were also strictly required since the AWS console cannot be used for the operations required, such as starting the face search. The AWS CLI could be used to perform these operations, but this is a more manual approach which would hurt the consistency and reproducibility of the results.

[52] `https://github.com/ErlendF/face_blur/tree/main/terraform`
[53] `https://github.com/ErlendF/face_blur/tree/main/scripts/sagemaker`
[54] `https://github.com/ErlendF/face_blur/blob/b9efa56c27fd69dedb01c378cff631fc2d045ac6/scripts/sagemaker/ex_variables.py`

3.4 Distribution

The majority of the implementation is made available as a package using *pip*, the most popular package-management system for Python [53], for easy use and installation[55]. It can be installed using the command shown in Listing 19.

```
pip install git+ssh://git@github.com/ErlendF/face_blur.git
```

Listing 19: Package installation

3.4.1 Example configuration

Listing 20 shows a very simple example of how the full pipeline may be used, which follows the outline given in the overview 3.1. First the shot transitions are detected. Then, the video is dynamically processed, and the facial sequences are made, both utilizing the detected shot transitions. Afterwards, the sequences are interpolated to remove gaps and false positives. In this case, any face which is not recognized is then removed from the facial sequences, thereby only blurring known faces. Finally, the faces are blurred and written back to disk.

[55]It is available from the same GitHub repository as discussed previously: `https://github.com/ErlendF/face_blur`

```
from videoface import get_shot_transitions, dynamically_process, make_sequences,
↪  interpolate, filter_short_sequences, init_known_faces, filter_known_faces,
↪  write_faces, copy_remaining_files
from smoothing import avg_smoothing
from os import makedirs

img_dir = "/path/to/your/image/folder"
out_dir = "/path/to/your/output/folder"
known_faces_dir = "/path/to/your/known/faces/folder"
makedirs(out_dir, exist_ok=True)

shot_transitions = get_shot_transitions(img_dir)
frames, matchings = dynamically_process(img_dir, processing_func=deep_face_process,
↪  shot_transitions=shot_transitions)
seqs = make_sequences(frames, matchings, shot_transitions=shot_transitions)
seqs = interpolate(seqs)
seqs = filter_short_sequences(seqs)

known_faces = init_known_faces(known_faces_dir)
seqs = filter_known_faces(seqs, known_faces)

seqs = avg_smoothing(seqs)
write_faces(seqs, img_dir, out_dir)
copy_remaining_files(img_dir, out_dir)
```

Listing 20: Simple usage example

3.4.2 Interchangeability

Each function is designed to be as modular as possible. Any function may be easily replaced, as long as it follows the same interface. The `full_process` and `dynamically_process` functions are interchangeable, and could be replaced by any other function, as long as the inputs and outputs are the same. Except for the two processing functions and the `make_sequences` function, all of the other functions operate using the facial sequences. This makes every function independent and interchangeable. Each function may be used separately, replaced or ignored, and new functions may be added without affecting any other function. Some functions are, however, intended to work together. For instance, it does not make a lot of sense to dynamically process a video without interpolating the missing frames, or otherwise processing them in some other way. Without the interpolation, only a selection of frames would contain any information to blur the faces in the video.

Similarly, both the `full_process` and `dynamically_process` functions take the `processing_func` function (which detects faces in the frames and produces facial features representations) as a parameter, as shown in Listing 20. This makes it easy to replace the facial detection and facial recognition. This is also how the tests in Section 4.1 were performed; by simply replacing the processing function with either the `deep_face_process` or `dlib_process`, depending on the test.

Chapter 4

Analysis

This chapter will discuss and analyse the most important aspects of the implementation, and the performance of the proposed pipeline will be explored primarily in terms of its prediction quality, running time and cost. It is meant to further highlight and justify the various implementation decisions discussed in Chapter 3, and their various benefits and drawbacks.

4.1 Prediction quality

In this section, the prediction quality of four different configurations presented in the thesis will be tested and compared:

- Processing every frame using Deepface
- Processing every frame using Dlib
- Dynamic processing using Deepface
- Dynamic processing using Dlib

In order to measure the prediction quality of each configuration, an example video was made with multiple moving faces in various scales, poses, occlusions, and illumination conditions: `https://youtu.be/_5i5kza5C-M`. The video has several cuts and is intended to be a fairly "normal" video. It does, however, have some situations where the configurations may struggle to correctly detect or recognize the faces in the video. This is intended to highlight the differences in the quality of each configuration's predictions.

The processed videos were made using the example shown in Listing 20, modifying it to test the various configurations. A full list of the videos is available in Appendix C. In the videos, the bounding boxes are also displayed rather than blurring the faces. This is simply to make it clearer what is actually detected correctly and what is not. The videos are 2 minutes and 32 seconds long, and contain a total of 4587 frames each (30 frames per second), and are without sound since it is outside the scope of the thesis. Each frame is divided into one of four categories:

- True positive: All faces in the frame are correctly detected.
- True negative: There are no faces in the frame, and no faces are detected.
- False positive: Something other than a face is detected as a face.
- False negative: A face in the frame is not detected.

The videos were manually reviewed to count the number of frames in each category. There are, however, some frames which are not easily categorizable. The following criteria was used to categorize these. The full overview of the frames and their categorization is available in Appendix B. The frame number has been added at the bottom of each video (except the original) to make it easier to refer to specific sections of the video.

- Instances where a face is briefly out of view, but is still marked by a bounding box, is *not* considered a false positive. This is the intended behaviour, and arguably not particularly disruptive. It does, however, contribute greatly in other aspects by filling in missing values.

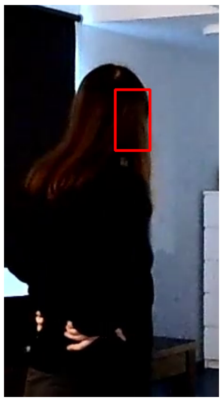
Figure 4.1: The bounding box is interpolated when a face is out of view for a brief period[1]

[1]Example timestamped at 02:01:
`https://youtu.be/v_49prmpeec?t=121`

- False negatives are counted as instances of faces being visible without being detected, where at least one eye is clearly visible. This is typically only a problem with faces viewed from the side. It is difficult to clearly define when a missing detection should be counted as a false negative when someone is entering the frame. Each frame was therefore counted as either a true or false negative for every configuration in order to not differentiate between them. The example shown in Figure 4.2 *is* counted as a false negative.

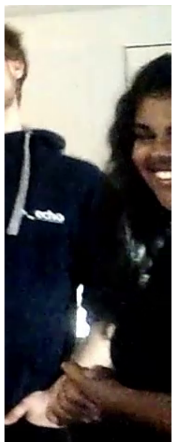

Figure 4.2: Faces on the edge of the frame may be difficult to detect[2]

- In instances with excessive movement, where it is impossible to identify the face even without blurring, like shown in Figure 4.3, the frame is *not* counted as a false negative.

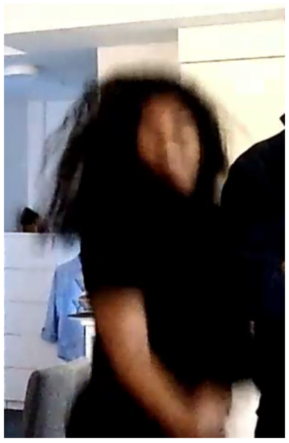

Figure 4.3: Rapid facial movement[3]

- Incorrect matchings where the bounding boxes "travel" across the screen are counted as false positives[4].

- In the example video, there is a face in every frame of the video. Consequently, there are no true negatives since there are no frames where detecting no faces would be correct.

[2]Example timestamped at 01:40: `https://youtu.be/v_49prmpeec?t=100`

[3]Example timestamped at 01:17: `https://youtu.be/v_49prmpeec?t=77`

[4]Example timestamped at 01:17: `https://youtu.be/nnXTJOOwLiQ?t=77`

4.1.1 Prediction quality results

The Deepface configurations outperformed the Dlib configurations in every metric, being both more accurate, precise and with a higher recall. The overall best configuration was the dynamic processing using Deepface, which had slightly fewer false positives than the Deepface configuration processing every frame. Each of the instances of false positives present when processing every frame was still visible when dynamically processing, but they lasted for fewer frames. The dynamic processing likely had fewer false positives because it does not process every frame.

Table 4.1: Prediction quality: Deepface (processing every frame)[5]

	Positive	**Negative**
True	4497 (98.0%)	0 (0.000%)
False	23 (0.501%)	67 (1.461%)
Accuracy	0.980	
Precision	0.995	
Recall	0.985	
F1-score	0.990	

Table 4.2: Prediction quality: Deepface (dynamic processing)[6]

	Positive	**Negative**
True	4514 (98.4%)	0 (0.000%)
False	6 (0.131%)	67 (1.461%)
Accuracy	0.984	
Precision	0.999	
Recall	0.985	
F1-score	0.992	

[5] `https://youtu.be/tORTBS7iEGY`
[6] `https://youtu.be/73XS75-tdYQ`

Table 4.3: Prediction quality: Dlib (processing every frame)[7]

	Positive	**Negative**
True	3783 (82.5%)	0 (0.000%)
False	38 (0.828%)	766 (16.7%)
Accuracy	0.825	
Precision	0.990	
Recall	0.832	
F1-score	0.904	

Table 4.4: Prediction quality: Dlib (dynamic processing)[8]

	Positive	**Negative**
True	3753 (81.8%)	0 (0.000%)
False	33 (0.719%)	801 (17.5%)
Accuracy	0.818	
Precision	0.991	
Recall	0.824	
F1-score	0.900	

All of the configurations struggled with a face hovering at the edge of the screen, particularly the one shown in Figure 4.2. That shot alone is the source of all of the false negatives for the Deepface configurations. Generally, the Deepface configurations seemed to handle faces viewed from the side very well, and it is unclear why this shot in particular was more difficult. It could potentially have struggled because more than half of the face was never visible, and it was at the very edge of the screen. In all other cases, it handled faces from the side without difficulty.

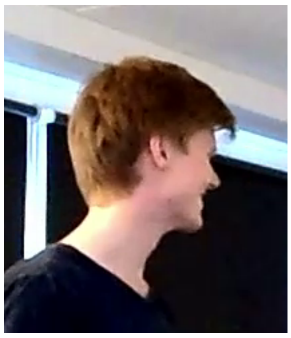
Figure 4.4: Face from the side not detected by Dlib

[7]https://youtu.be/nnXTJOOwLiQ
[8]https://youtu.be/v_49prmpeec

The Dlib configurations did, however, seem to struggle more with faces viewed from the side in general. There were several instances where it was unclear whether or not it should be counted as a false negative or not, since parts of the face were visible, like shown in Figure 4.4, but not enough to be considered a false negative by the definition provided above. It would be hard to identify someone based on such an image, but it does reveal some of their facial features. The Deepface configurations had far fewer of these instances and it mostly detected the faces before the faces became clear enough to properly identify them.

The Dlib configurations also struggled in poor lighting conditions, especially with darker skin tones, as shown in Figure 4.5. There was otherwise no noticeable difference between skin tones in the scenes with better lighting conditions in the video. However, in the example of using Dlib to process every frame without interpolation shown in Section 4.4, there were a greater number of instances of false negatives for darker skin tones, though this is not nearly a large enough sample size to definitively conclude there is such a bias.

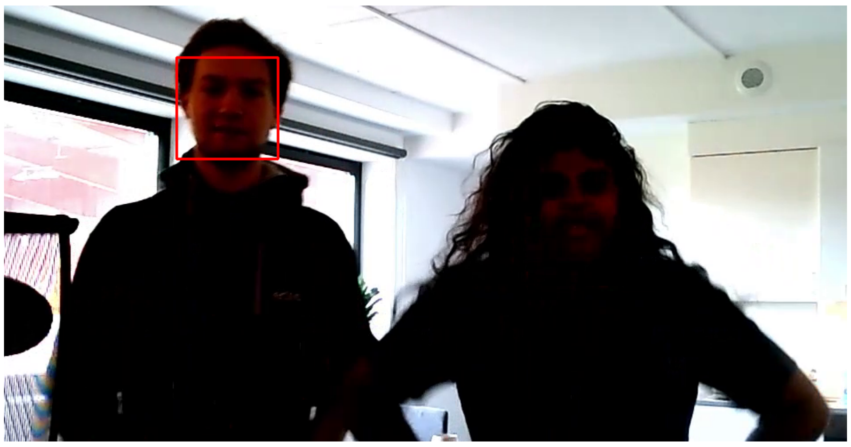
Figure 4.5: Difference between detection of two different skin tones in poor lighting conditions

There were no significant differences between the dynamic processing and processing every frame, neither for the Deepface nor Dlib configurations. As mentioned, in the case of the Deepface configurations, the only difference between the two was that the dynamic processing had slightly fewer false positives, likely *because* it did not process every frame. In the case of the Dlib configurations, processing every frame was a bit more accurate due to its fewer false negatives, but also very slightly less precise due to a minor increase in the number of false positives.

Regardless of the performance metrics, the Deepface configurations bounding boxes moved a lot smoother throughout the video. The Dlib bounding boxes were noticeably more jittery, especially when processing every frame. This is not as disruptive to the viewer when blurring the images rather than displaying the bounding boxes, but is still very noticeable. When dynamically processing the frames, the jitter of the bounding boxes was less noticeable (as discussed in Section 3.1.3), but it was still clearly present and far more noticeable than with the Deepface configurations.

4.2 Cost and running time

Due to the results of Section 4.1.1, only the implementations using Deepface were tested in this section. The following is a collection of tests of three different AWS instances running AWS SageMaker Processing Jobs (see Section 3.3.1) using three various videos as input. AWS SageMaker Processing Jobs were used since they handle the underlying transfer of data and utilize standardised AWS instances, thereby making reproducible results. Each of the test videos is 1 minute long, 30 frames per second at 1080p quality, and each tests a different scenario. The variety of videos was chosen to test the dependence of the running time on the video being processed, and highlight any differences between the impact on running time on the various configurations.

- **Multiple faces -** A video containing multiple moving faces, entering and exiting the view[9].
- **Single face -** A video containing a single, still face visible for the entire duration of the video[10].
- **No face -** A video without any faces at all[11].

[9]`https://youtu.be/HZz4862_lII`
[10]`https://youtu.be/VXteG6A2ME0`
[11]`https://youtu.be/Mku0Um84Iew`

The three different AWS instances were chosen to present a variety in options of instance types for the processing. Each of the instances are at a different price point per hour of usage, and each serve different purposes [18][12]. The prices provided are the prices of the Stockholm region (AWS region "eu-north-1") at the time of writing — May 2022.

- **ml.t3.large -** A standard instance. Fairly cheap ($0.104 per hour), with 2 vCPUs and 8GiB of memory.
- **ml.c5.2xlarge -** A compute optimized instance. A bit more expensive ($0.437 per hour), with 8 vCPUs and 16GiB of memory.
- **ml.g4dn.xlarge -** A GPU accelerated instance, utilizing an NVIDIA T4 GPU [23]. Even more expensive ($0.781 per hour), with 4 vCPUs and 16GiB of memory.

Each of the tests measured three separate values in order to detect differences in how each configuration affected the running time of each part of the application:

- **The container time -** The total processing time as reported by AWS SageMaker.
- **The full running time -** The total time spent by the application, as recorded from the start to the end of the main function.
- **The processing time -** The time spent processing the frames themselves, using either dynamic processing or processing every frame.

These tests were performed after the optimizations presented in Section 4.3.

4.2.1 Running time results

As shown by the test result tables and Figure 4.6, the dynamic processing is significantly faster than processing every frame, especially when there is little facial movement in the video. With multiple moving faces, the dynamic processing was between 3 and 4 times faster than processing every frame, depending on the AWS instance type being used. In contrast, with both a single, still face and no faces at all, the dynamic processing was between 6 and 18 times faster, again depending on the instance type. This effect is lessened somewhat when comparing the total runtime of the container, as there is some constant overhead when starting both the container and application.

[12]A full list of instances and their cost is available here: `https://aws.amazon.com/sagemaker/pricing/`

Table 4.5: Running time: ml.t3.large (processing every frame)

Measurement \Test	Multiple faces	Single face	No face
Container time	10h, 5m, 15s	9h, 45m, 39s	9h, 34m, 35s
Full running time	9h, 56m, 2s	9h, 36m, 27s	9h, 25m, 24s
Processing time	9h, 43m, 32s	9h, 23m, 1s	9h, 23m, 46s

Table 4.6: Running time: ml.c5.xlarge (processing every frame)

Measurement \Test	Multiple faces	Single face	No face
Container time	48m, 41s	47m, 24s	45m, 2s
Full running time	46m, 34s	45m, 59s	43m, 5s
Processing time	41m, 53s	41m, 14s	42m, 36s

Table 4.7: Running time: ml.g4dn.xlarge (processing every frame)

Measurement \Test	Multiple faces	Single face	No face
Container time	15m, 27s	14m, 49s	9m, 57s
Full running time	14m, 9s	13m, 46s	8m, 12s
Processing time	8m, 23s	8m, 4s	8m, 2s

Table 4.8: Running time: ml.t3.large (dynamic processing)

Measurement \Test	Multiple faces	Single face	No face
Container time	2h, 41m, 43s	55m, 6s	42m, 22s
Full running time	2h, 32m, 46s	45m, 59s	33m, 19s
Processing time	2h, 20m, 24s	32m, 11s	31m, 42s

Table 4.9: Running time: ml.c5.xlarge (dynamic processing)

Measurement \Test	Multiple faces	Single face	No face
Container time	17m, 38s	9m, 22s	5m, 37s
Full running time	15m, 50s	7m, 57s	3m, 41s
Processing time	10m, 45s	3m, 0s	3m, 0s

Table 4.10: Running time: ml.g4dn.xlarge (dynamic processing)

Measurement \Test	Multiple faces	Single face	No face
Container time	9m, 58s	8m, 11s	2m, 52s
Full running time	8m, 47s	7m, 7s	1m, 34s
Processing time	2m, 53s	1m, 24s	1m, 23s

The difference in speedup when using dynamic processing for the multiple faces example compared to the single- and no face examples, as visualized in Figure 4.6, is likely primarily due to certain sections of the video needing to be reprocessed. In the single- and no face examples, there is no instance of any face entering nor leaving the frame, and it is therefore very easy to interpolate without needing to process additional frames (as discussed in Section 3.1.3). Therefore, the interval used for the dynamic processing may be increased, such that a greater number of frames are interpolated and do not need to be processed. Additionally, a facial features representation needs to be made for every face detected, thereby further increasing the running time by the number of faces. However, this increase is comparatively small, as demonstrated by the little to no difference between the single- and no face examples.

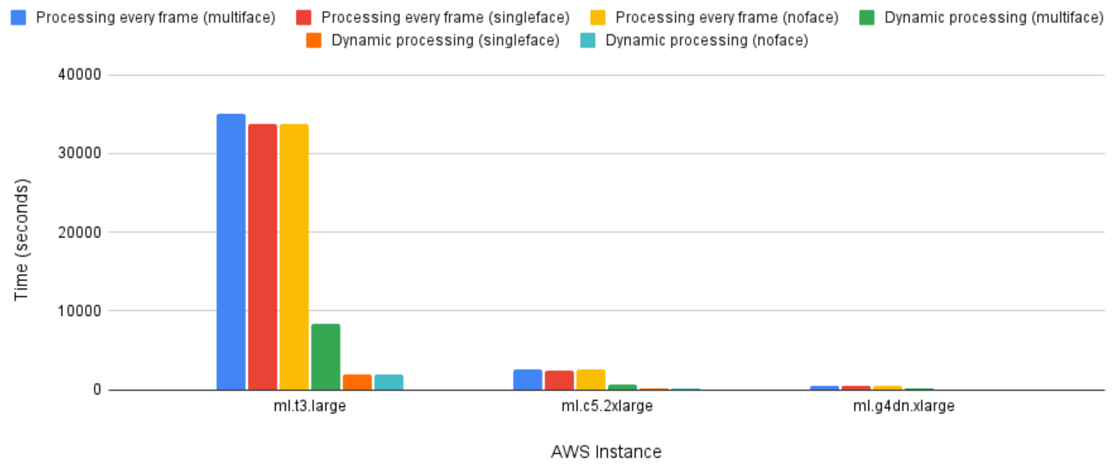

Figure 4.6: Processing time

Additionally, the tests performed on the smaller AWS instances had a much larger speedup when using dynamic processing compared to processing every frame. This is likely because as the processing time decreases, the overhead which is constant regardless of the instance, such as the time spent reading images from the disk, becomes a greater part of the total processing time. In the case of the ml.g4dn.xlarge instance utilizing GPU acceleration for the facial detection and facial recognition, there is also some overhead each time the work is handed off to the GPU [70], though it was still the fastest AWS instance by a wide margin. Using larger batch sizes for the processing would likely alleviate the extra overhead partly, as the GPU processing would be started fewer times, but with a greater workload each time. The batch size can be passed as a parameter both when dynamically processing and processing every frame.

The tests also showed the significant performance impact of utilizing GPU acceleration. The ml.g4dn.xlarge instance, which only has 4 vCPUs, but also an NVIDIA T4 GPU, ran each test significantly faster than the ml.c5.2xlarge instance, which has 8 vCPUs [23, 18]. The processing was completed about 5 times faster on the ml.g4dn.xlarge instance compared to the ml.c5.2xlarge instance when processing every frame, and between 2 and 4 times faster when using dynamic processing. The difference between speedups when using dynamic processing and processing every frame is likely for the same reason as the other instances discussed previously.

4.2.2 Cost

Unfortunately, the costs of the AWS bills are aggregated and can therefore not be used for direct price comparisons. The prices for the AWS instances are listed per hour, but they are billed per second of usage [9]. Therefore, the costs shown in Tables 4.11 and 4.12 are calculated by multiplying the cost of the AWS instance type by the container running time, rounded off to the nearest cent. They show the cost of the same tests as in Section 4.2.1 on each AWS instance type, both when processing every frame and using dynamic processing.

Table 4.11: Cost: Processing every frame

AWS Instance \Test	**Multiple faces**	**Single face**	**No face**
ml.t3.large	$1.05	$1.02	$1.00
ml.c5.2xlarge	$0.35	$0.35	$0.33
ml.g4dn.xlarge	$0.20	$0.19	$0.13

Table 4.12: Cost: Dynamic processing

AWS Instance \Test	**Multiple faces**	**Single face**	**No face**
ml.t3.large	$0.28	$0.10	$0.07
ml.c5.2xlarge	$0.13	$0.07	$0.04
ml.g4dn.xlarge	$0.13	$0.11	$0.04

Since processing every frame takes more time than dynamic processing, as discussed in Section 4.2.1, it is consequently also more expensive. In most cases, processing every frame is between twice and ten times more expensive than dynamic processing. Considering the small differences in prediction quality, as discussed in Section 4.1, it makes sense to use dynamic processing in nearly every case. In scenes with multiple faces moving in and out of the frame, the differences between the two processing methods are reduced, but still clearly noticeable. On a larger scale, these differences in cost will be much more apparent.

There is no one cheapest instance among the AWS instances tested. When processing every frame, the ml.g4dn.xlarge was the cheapest in every case. However, when using dynamic processing, the ml.g4dn.xlarge and ml.c5.2xlarge instances were equal with the exception of the single face test, where the ml.c5.2xlarge instance was slightly cheaper. Consequently, the cheapest instance may differ based on the dataset. Considering the minor differences, and the lower running time of the ml.g4dn.xlarge instance, it is likely preferable to use it for most applications. Additionally, the ml.g4dn.xlarge instance will likely be cheaper in comparison to instances without GPU acceleration for larger data sets. Since processing the frames is the most computationally intensive part of the application, a larger portion of the workload can be offloaded to the GPU with larger data sets. Using larger batch sizes may also affect performance, as previously discussed in Section 4.2.

Additional costs

There are also some additional costs related to storing both the data in S3, and the container image in ECR. For the S3 bucket, it costs \$0.023 per GB per month [16], and for ECR it costs \$0.10 per GB per month [7]. There is also an additional fee when retrieving data from S3 of \$0.0004 per 1000 requests for a standard S3 instance, and a fee for outbound data transfers from ECR private repositories at \$0.09 per GB. This would for instance be when you retrieve the image for use. The image could also be moved to a public repository, which has no fee associated, given certain constraints[13]. These costs are, however, likely of little importance in a production environment, and most of them will be required regardless of using the proposed pipeline at all. The videos still need to be stored regardless of whether or not any faces are blurred in them.

[13]See `https://aws.amazon.com/ecr/pricing/`

Cheaper alternatives

Using AWS SageMaker Processing Jobs is comparatively expensive compared to using the underlying EC2 instances directly. For instance the ml.g4dn.xlarge instance costs \$0.558 in EC2 on demand [21]. Using spot instances (unused EC2 capacity[14]), this cost is further reduced to \$0.1982, nearly a fourth of the Processing Job cost [22]. However, the spot price fluctuates frequently, and spot instances may be terminated by AWS at any time when there is not enough unused EC2 capacity. Additionally, when using EC2 directly, data transfers and spinning up and down instances need to be handled independently. Using AWS Batch (not to be confused with AWS SageMaker Batch Transform) may be a good alternative since it can scale up and down automatically, and has no additional cost compared to the underlying EC2 instances [3].

4.2.3 Scalability

Larger videos may potentially be split and processed in parallel with nearly a linear speedup since sequential parts of the video should be able to be processed entirely independently. Consequently, the only overhead would be splitting and recombining the data set. Given that the data is stored in S3 and needs to be retrieved and written back regardless of the parallelization, this should be practically indistinguishable from processing the data on a single node. Thus, the only overhead should be deciding how to split the video. This is assuming that the video is processed by already running instances, thereby ignoring startup times. In the examples given, this would however not be the case, and the instances would have to be started specifically to process the video. Starting the instances takes time and the cost consequently scales by the number of instances started. The time overhead of starting the instances should be constant, but as the data set is split across a greater number of instances, it will become a greater portion of the total running time.

When processing the video in parallel, it could be split evenly across the instances, but this may cause artifacts at the points in the video where it was divided since there will be no information shared before and after the divide. This could be alleviated by having some overlap between adjacent sections of the video, and averaging the results in the overlap. Alternatively, the video could be split on detected shot transitions. This may, however, be problematic if there are no shot transitions, and it would limit the possible parallelization of the processing. This may or may not be an issue depending on the video and the time requirement.

[14]`https://aws.amazon.com/ec2/spot/`

4.3 Use of computational resources

When discussing the use of computational resources, it is important to measure what is using the most resources. For this purpose, profiling is a well suited tool. In this case, the python profiler[15] was used. The profiler generates a *profile*, which is simply a list of statistics which describes the time spent on various parts of the program, and how many times each function was called [43]. This profile may then be visualized using Graphviz[16], but it first needs to be converted into a dot graph, which Graphviz can visualize [42]. In order to do so, *gprof2dot*[17] was used.

Profiling the running time of the dynamically processed Deepface implementation produced the graph shown in Figure 4.7. The figure only shows a part of the graph since the whole graph would not fit into a single page[18]. Please note that the profiling was done on a personal computer using an SSD and GPU acceleration, which greatly decreases running time as discussed in Section 4.2.

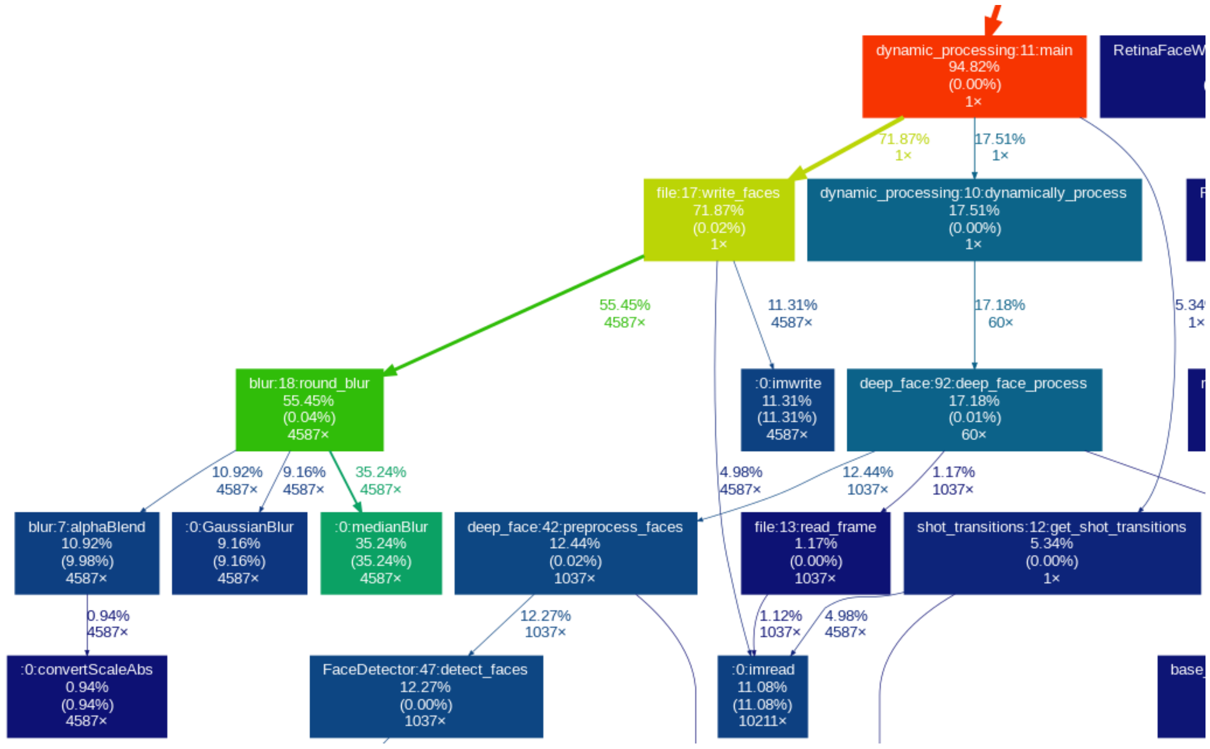
dynamic_processing:11:main 94.82% (0.00%) 1×
RetinaFaceW
71.87% 1×
17.51% 1×
file:17:write_faces 71.87% (0.02%) 1×
dynamic_processing:10:dynamically_process 17.51% (0.00%) 1×
55.45% 4587×
11.31% 4587×
17.18% 60×
5.34 1×
blur:18:round_blur 55.45% (0.04%) 4587×
:0:imwrite 11.31% (11.31%) 4587×
deep_face:92:deep_face_process 17.18% (0.01%) 60×
10.92% 4587×
9.16% 4587×
35.24% 4587×
4.98% 4587×
12.44% 1037×
1.17% 1037×
blur:7:alphaBlend 10.92% (9.98%) 4587×
:0:GaussianBlur 9.16% (9.16%) 4587×
:0:medianBlur 35.24% (35.24%) 4587×
deep_face:42:preprocess_faces 12.44% (0.02%) 1037×
file:13:read_frame 1.17% (0.00%) 1037×
shot_transitions:12:get_shot_transitions 5.34% (0.00%) 1×
0.94% 4587×
12.27% 1037×
1.12% 1037×
4.98% 4587×
:0:convertScaleAbs 0.94% (0.94%) 4587×
FaceDetector:47:detect_faces 12.27% (0.00%) 1037×
:0:imread 11.08% (11.08%) 10211×
base

Figure 4.7: Profiling visualization

[15] https://docs.python.org/3/library/profile.html#module-profile
[16] https://graphviz.org/
[17] https://github.com/jrfonseca/gprof2dot
[18] Full graph available here: https://github.com/ErlendF/face_blur/blob/7346af7216f5bef12d645b2d5e544134fe3e6cec/docs/Profiling_before_blur_update.png

Each node in the graph shown in Figures 4.7, 4.8 and 4.10 represents a function and contains four pieces of information, in the following order [43]:

- The name of the function (`filename:lineno(function)`).
- The percentage of total running time of the function and all sub-functions. This is the cumulative time of the function, measured from invocation till exit.
- The percentage of the total running time of the function alone, excluding sub-functions.
- The number of times the function was called.

4.3.1 Optimizing blurring

As shown in Figure 4.7, the `write_faces` function (which blurs and writes the frame to disk) makes up the majority of the running time of the program at 72% while the dynamic processing itself only used 18%. Most notably, over half of the total running time was spent by the `round_blur` function alone, which simply blurs the selected faces as described in Section 3.1.11. This seemed a bit excessive and should be reduced. By replacing both the `medianBlur` and `GaussianBlur` OpenCV functions with the `blur` OpenCV function, and slightly simplifying the `alphaBlend` function, the total time spent blurring faces was reduced from 55% to 21%, more than halving its impact, as shown in Figure 4.8. The optimization did slightly change how the blurring looked, though not very noticeably.

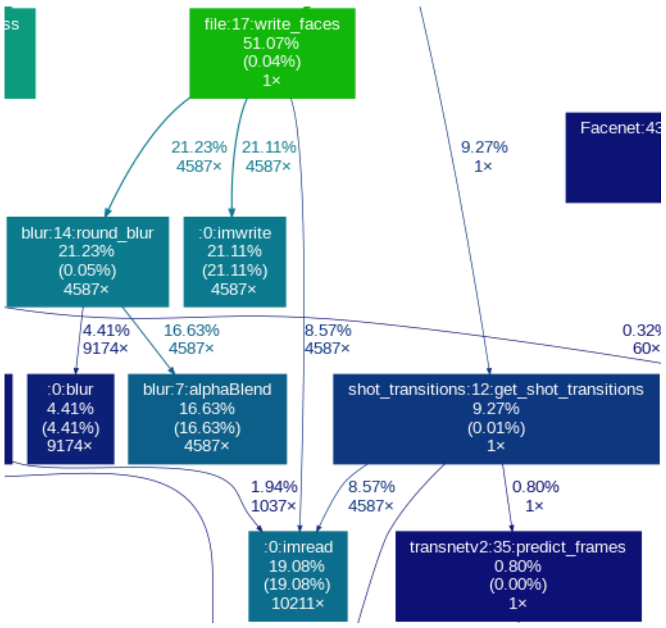

Figure 4.8: Profiling visualization after optimizing blurring[19]

[19]Full graph available here: `https://github.com/ErlendF/face_blur/blob/7346af7216f5bef12d645b2d5e544134fe3e6cec/docs/Profiling_after_blur_update.png`

4.3.2 Optimizing the reading of frames

As shown in Figure 4.8, simply reading the frames from disk makes up 19% of the running time (the `imread` function from the OpenCV python library[20]) after optimizing the blurring. This could be reduced by using a faster disk, but this is not always feasible. The comparatively long running time is partly because the current implementation reads each frame 2-3 times depending on the configuration, as shown in Figure 3.1. A frame may be read in the following occasions:

1. To detect scene transitions. These frames are only stored as a significantly down scaled version ($48x27$).
2. When processing the frames. Processing every frame will, of course, read every frame. Dynamic processing will only read a subset of the frames.
3. When blurring the faces, the frame needs to be read, the faces blurred, and the frame is written back to the disk.
4. Any frame which did not have any faces to blur is copied to the output folder.

Each frame may be read at four different occasions, but since only frames which did not contain any faces to blur are copied to the output folder, a single frame will never be read to both blur the faces in it and copy it to the output folder. Consequently, a single frame can be read a maximum of three times.

As shown in Section 4.2, when using dynamic processing with GPU acceleration, more time was spent reading and writing the frames to disk than actually processing the video itself. This could be significantly reduced by only reading each frame once, rather than 2-3 times. This implementation is mostly due to memory capacity restrictions. Storing every frame of larger videos is infeasible with limited memory capacity, though it would be possible by increasing the memory available, or by splitting the video into smaller sections in order to reduce the total number of frames each instance needs to process.

[20]`https://pypi.org/project/opencv-python/`
[21]`https://pypi.org/project/memory-profiler/`

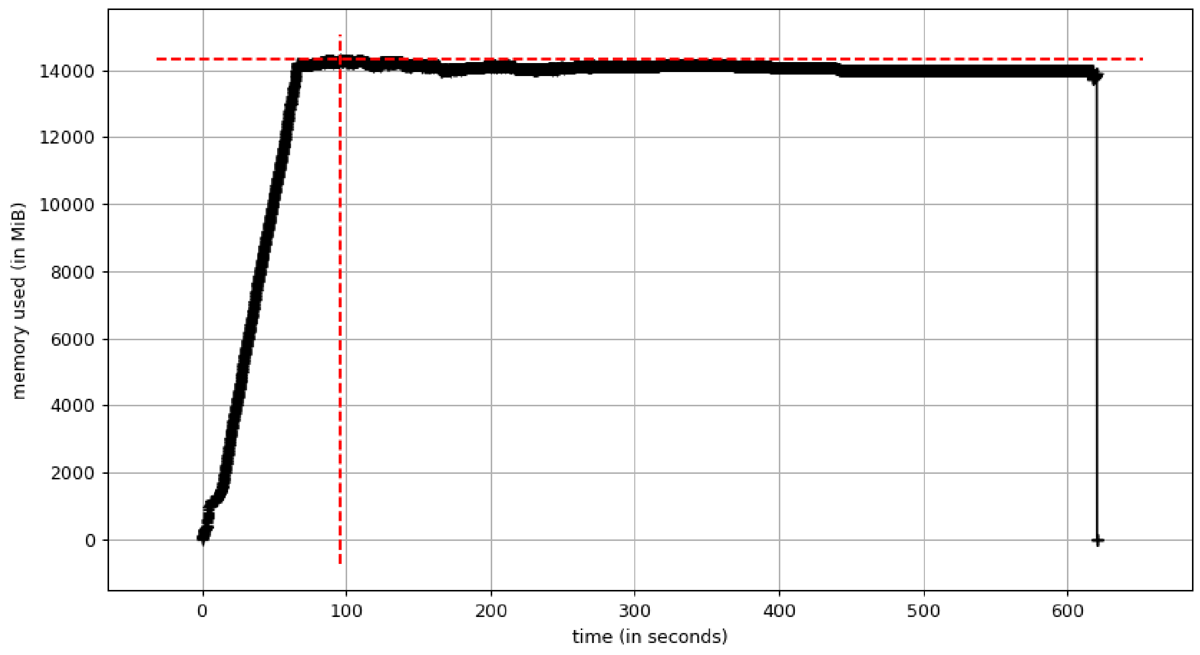

Figure 4.9: Memory usage visualization reading all frames once, made using memory-profiler library[21]

In the case of the 720p example video, it is just small enough to be fully loaded using 16GB of memory, as shown in Figure 4.9 (a full overview is available in Appendix D). For this purpose, the implementation was adapted to both accept pre-loaded frames, and reading them manually like the previous implementation for when the video is too large. By doing so, the frames could be read a single time rather than 2-3 times and passed to each required section of the pipeline. This reduced the percentage of time spent reading the frames from 19% to 8% as shown in Figure 4.10, more than halving its impact.

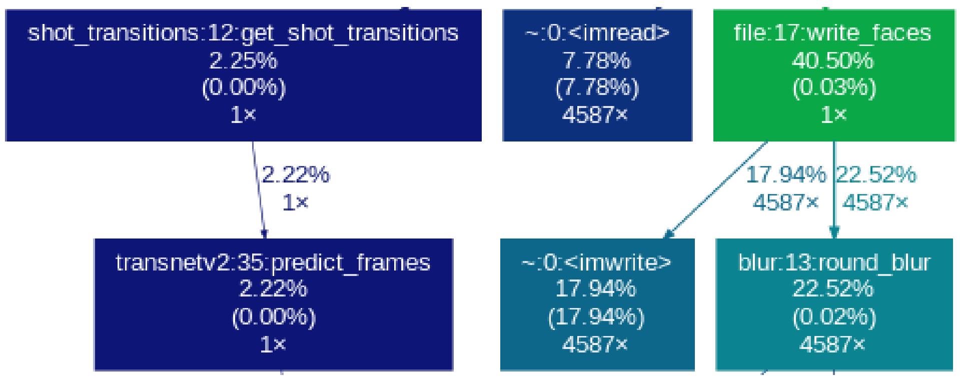

Figure 4.10: Profiling visualization after optimizing the reading of frames[22]

[22]Full graph available here: `https://github.com/ErlendF/face_blur/blob/605eef859e6097351acbc8bc5f91d2de0072f278/docs/Profiling_reading_frames_once.png`

Even in the cases where there is not sufficient memory to read all frames at the same time, some of the frames may still be kept in memory automatically by *read caching*. When reading data from I/O, a copy may be retained in order to accelerate future requests for the data [62]. This is managed by a cache replacement algorithm [48], and there is no guarantee of the data still being in cache the next time it is read. It would therefore still be beneficial to reduce the number of times the frames are read.

4.4 Removing false negatives

Removing false negatives was mostly effortless using the facial sequence and interpolation mechanisms when the face was intermittently detected correctly. Table 4.13 shows the prediction quality results of an example where Dlib was used to process every frame *without* interpolating the result. Comparing this to Table 4.3, where the results were interpolated, shows that there are a bit fewer false positives, but a lot more false negatives, resulting in a slightly lower accuracy, recall and F1-score, but a slightly increased precision. This is because there were far more instances of false negatives, where the bounding boxes were missing for frames at a time. In the example with interpolated sequences, there were 7 separate instances of false negatives, each being several frames long (see Appendix B for a full overview). However, in this case where the sequences where not interpolated, there were 34 instances, nearly five times as many, each often being just a few frames long.

Table 4.13: Prediction quality without interpolation: Dlib (processing every frame)[23]

	Positive	**Negative**
True	3672 (80.1%)	0 (0.000%)
False	6 (0.131%)	909 (19.8%)
Accuracy	0.801	
Precision	0.998	
Recall	0.802	
F1-score	0.889	

[23]`https://youtu.be/nSvN24R_wRU`

As shown, the interpolation and facial sequences successfully removed most instances of false negatives. However, when a face is not detected at all, like in the example shown in Figure 4.2, there is no way of detecting the face based on the output information from the facial detection model. In these cases, the only options available are to either replace or further train the model since any changes to other parts of the pipeline cannot affect this. Fortunately, the Retinaface model performed very well and had only one instance of a face not being detected at all, as discussed in Section 4.1.1.

The downside of using interpolation with inaccurate models is the increase in false positives. There were fewer false positives when not interpolating since the interpolation often also interpolated between the false positives, thereby extending them. Additionally, when the facial sequences were incorrectly matched, and the bounding boxes were interpolated between different faces, which was also counted as false positives. With the Dlib model, this happened a few times throughout the example video. In contrast, there was not a single instance of false positives due to interpolation in the Deepface examples. Consequently, it greatly depends on the model. Regardless, it is likely beneficial to have fewer false negatives at the cost of slightly more false positives in most scenarios, since the false negatives may reveal the faces of the people in the video.

4.5 Removing false positives

In both of the configurations using Deepface to process the video, there were three instances of false positives, each of which lasted for two frames (see Appendix B). By filtering short facial sequences (as discussed in Section 3.1.10), these could potentially be removed without negatively impacting the true positive facial detections. In this case, the false positives and their length are known, and we could therefore remove them by filtering any sequence shorter than three frames. However, in most cases, this will not be known beforehand. Therefore, it would be a more representative test to filter sequences shorter than 10 frames (1/3 of a second at 30fps) instead. A face would then have to be visible for 10 frames or more in order to be detected. The following test was therefore performed filtering sequences shorter than 10 frames using dynamic processing with Deepface.

Table 4.14: Prediction quality filtering shorter facial sequences: Deepface (dynamic processing)[24]

	Positive	**Negative**
True	4518 (98.5%)	0 (0.000%)
False	2 (0.0436%)	67 (1.461%)
Accuracy	0.985	
Precision	1.00	
Recall	0.985	
F1-score	0.992	

This worked well for two of the three instances of false positives. Unfortunately, the last instance has been incorrectly matched to a facial sequence, which makes the total facial sequence longer than the filtering length. Consequently, it was not removed when filtering the short sequences. The false positive was incorrectly matched to the facial sequence for two reasons:

1. The false positive was coincidentally in the same area of the screen (a bounding box distance of 0.47), and had a surprisingly low facial feature distance of 0.27. Combining the bounding box and facial feature distances results in a score lower than the threshold of 1, which allowed them to form a single facial sequence.

2. The false positive ended on the *exact* frame before the actual facial sequence started. This allowed the sequences to be matched since there was no actual detected instance of the face to be correctly matched. If there had been an actual instance of the face, it would most likely have had a lower distance to the rest of the sequence, thus being matched to the sequence instead of the false positive. The false positive would then have formed a separate facial sequence two frames long, which would have been removed.

[24]`https://youtu.be/yaW0Vu0yGPk`

This could be prevented by lowering the threshold for considering two faces to belong to the same person. However, this would also increase the number of faces which are not correctly matched in cases where a face is moving, or for various reasons (illumination, pose, occlusion etc.) may produce a facial features representation with substantial changes between frames.

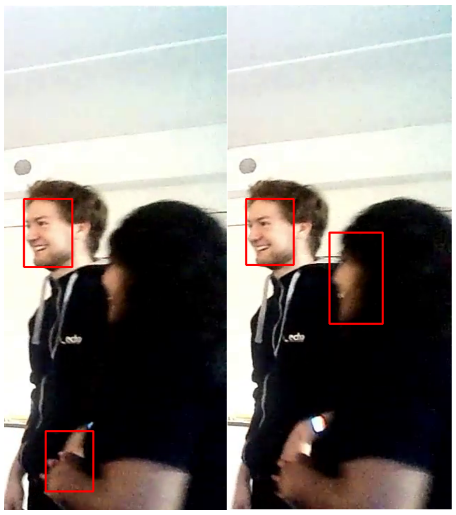
Figure 4.11: False positive and true positive in adjacent frames which were incorrectly matched

4.6 Selecting specific faces for blurring

A selected group of the detected faces in the video may be blurred, or not blurred, depending on the desired result. Similarly to the previous tests, the bounding boxes are displayed rather than blurring the faces in order to clearly show what has been correctly detected and what has not. For the purposes of selecting specific faces, the selected faces are coloured blue, and all non-selected faces are still coloured red.

4.6.1 Selecting known faces

When selecting already known faces, images of the face or faces which should be selected need to be provided. For this test, the images shown in Figure 4.12 were provided. Five images were provided, giving five different opportunities to correctly recognize the face in the video. Furthermore, the photos are taken from various angles, thereby giving a variety of perspectives which may make it easier to correctly recognize. Obtaining such photos, for instance, for any show host or news reporter which should not be blurred in a video would be very easy given the likely high number of recordings of them available.

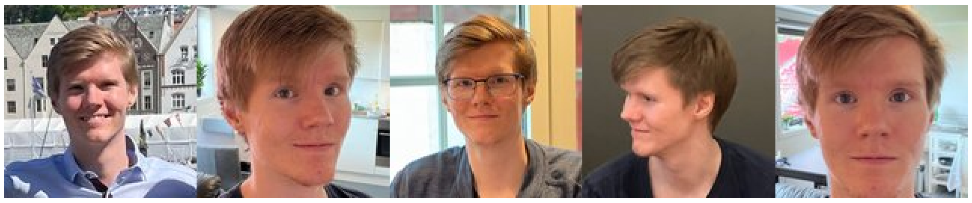

Figure 4.12: Selecting known faces

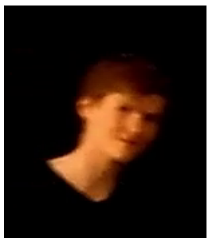

Figure 4.13: Missing recognized face

This worked fairly well, and all instances of the selected face were correctly identified except for one: `https://youtu.be/Wf_yMDSXteo?t=27`. In order to fix this, the threshold for recognition could be lowered, which would likely include the missing sequence. However, this would also increase the likelihood of other faces incorrectly being identified. Alternatively, additional images could be added, which would provide additional opportunities to correctly match the face. The easiest images to use in order to ensure detection, are images taken directly from the video itself, as shown in Figure 4.13. This image is easily retrievable after processing the video, and could potentially be selected using some form of user interface. All instances of the face were correctly identified after adding the image shown in Figure 4.13: `https://youtu.be/AJNC7OxWvrs`.

4.6.2 Selecting by time and location

When selecting a face based on time and location, a frame number and its coordinates in the frame need to be provided. For this test, the frame number 1500, and the coordinates $(640, 300)$ were used (coordinates are the number of pixels counted from the top left corner of the frame). The resulting video[25] nearly perfectly selected the correct face in each of the shots, except for the very first shot, which was incorrectly not included in the selection. It was likely not included due to the poor illumination in the first shot, which may have caused the facial features to be too different from the other sequences to be recognized.

In order to include it in the selection, a second selection may be made in order to include the missed face in the first shot. In this case, where only a single facial sequence needed to be selected, the threshold for identifying another sequence as belonging to the same face can be set to 1. With a threshold of 1, the other sequence needs to contain a *perfect* match of the facial features in the primary sequence. This will in practice

[25]`https://youtu.be/fkTlFBNTb94`

never happen, unless the video is artificially constructed to accomplish this. A second test was made to validate this (selecting frame number 200, location $(500, 360)$)[26], which selects the facial sequence incorrectly not included in the original video. As expected, all instances of the face in each shot was correctly selected in the second video.

In a real world scenario, this could be done by going through the output video and using some sort of user interface to select any faces which was not correctly selected, or selecting any faces which were incorrectly included in the selection. Thereby, the selection could be easily changed to only include the correct selection of faces.

4.7 Detecting shot transitions

Shot transition detection using TransNetV2 has worked remarkably well, and has been used to make all of the other examples shown. Without detecting the shot transitions, facial sequences are sometimes made across shot transitions when dynamically processing, which may look very peculiar[27]. This was not a major issue for the example video, likely due to the changing number of faces between frames which caused these sections to require reprocessing. When all the frames are processed in the section containing a shot transition, it is less likely that a facial sequence will be made across the shot transition due to the additional information gathered.

The TransNetV2 model detected all the shot transitions in the original example video perfectly. This is an example where all the shot transitions have been marked by a red circle in the lower right corner to display detections: `https://youtu.be/sTvsOq-1orQ`. Note that the mark is only visible for the exact frame of before the transition. The original example video, however, only contained cuts, which are easier to detect than gradual transitions [68]. Therefore, a second example video was made with a variety of gradual transitions and cuts to test how well the model handled it: `https://youtu.be/ZBVLBcMtlIE`. This also worked well, and nearly all the shot transitions were still detected, with the exception of two "push-left" transitions. However, the model is still more than good enough for the purposes of this thesis, and likely the most of video types that will likely be processed.

[26] `https://youtu.be/cvGPJO6klzc`

[27] Example timestamped at 00:08: `https://youtu.be/IuDkF_XdqWc?t=8`

When using the shot detection, the frames need to be handled separately from the rest of the pipeline since the TransNetV2 model uses scaled down versions of the frames. For this reason, the frames either need to be read from disk an additional time, or all the frames need to be read before processing, like discussed in Section 4.3.2, and a downscaled copy of each frame needs to be stored. In either case, the memory usage when detecting shot transitions is small, as shown by Listings 21 and 22 in Appendix D, at an approximate 700MiB for the example video.

Detecting shot transitions also uses some additional time. Although the time spent detecting the transitions themselves is negligible at 1-2% of the total running time, as shown in Figures 4.8 and 4.10 in Section 4.3, it takes significantly longer if the images need to be loaded an additional time. When reading the frames and resizing them independently from the rest of the pipeline when detecting shot transitions, detecting shot transitions was responsible for about 9% of the total running time, as shown in Figure 4.10, a significant increase in time spent. The detection of shot transitions is in other words limited by I/O performance when it has to read the images independently, and thus benefits greatly from not having to read the images directly, like implemented in the performance optimalizations in Section 4.3.2.

Though detecting shot transitions takes some additional time, it also sped up the processing itself when using dynamic processing. Because the shot transitions are known beforehand, the frames before and after the transitions could be processed. Consequently, there were no instances where an entire interval needed to be processed due to a shot transition. Whether or not this speedup will make up for the time spent detecting the shot transitions will depend entirely on the video and the number of transitions in it. When reading the frames only once, the impact of detecting the shot transitions will be smaller, thus making it easier to be compensated for when using dynamic processing.

4.8 Sequence models

When first starting to experiment with sequence models for smoothing and removal of false positives and negatives, the intention was to use a bidirectional Pytorch RNN[28], GRU[29] or LSTM[30] models, followed by two fully connected layers separated by ReLU activation functions. These models were trained using the output of processing every frame of the example video using the Dlib processing and the dynamic processing using Deepface. Afterwards, the models were used to predict the bounding boxes for the same video to see that it could fit the training data properly. If it could not properly fit the training data, it was very unlikely to fit any other data. The input coordinates of the bounding boxes were normalized by dividing them by the width and height of the image, thereby making the input range from 0 to 1.

Primarily, the RNN was used for testing, although all three models behaved similarly. Unfortunately, with the original loss function used (see Section 3.1.12), it was not able to properly follow the bounding boxes, and very poorly covered the faces in the video. The predicted bounding boxes were also mostly the same size, regardless of the size of the input bounding boxes and consequently the size of the faces in the video. Therefore, one of the fully connected layers were removed to simplify the model, and the loss function was simplified to only penalize the distance between adjacent bounding boxes if the input bounding box was missing. Otherwise, only the distance to the input bounding boxes would be penalized. This worked better, but the missing bounding boxes, and the beginning and end of sequences, were still not correctly matching the faces. Therefore, the loss function was yet again changed to use the absolute distance between each point of the bounding box rather than the square of the distance. This worked better, and the bounding boxes which were not missing fit the input data better, but the missing bounding boxes and the start and end of each sequence did not.

Therefore, the default value of the missing bounding boxes was changed from 0 to the value of the previous bounding box which was not missing, and the penalty was increased for the first and last few frames of each sequence. It could alternatively be set to the mean of the sequence, but the previous bounding box is more likely to be closer to the desired prediction value. This helped somewhat for the missing values, but not really for the ends of the sequences. The loss function was therefore changed yet again to penalize

[28]`https://pytorch.org/docs/stable/generated/torch.nn.RNN.html`
[29]`https://pytorch.org/docs/stable/generated/torch.nn.GRU.html`
[30]`https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html`

deviation from the input bounding boxes even more in the first and last frames of each sequence This did unfortunately not particularly improve the result.

Ultimately, the algorithmic alternatives worked very well, and exploring the sequence model consequently seemed less than fruitful[31]. The sequence models still have a great potential and would ideally be explored further. However, due to the limited time and resources for this project, other parts of the thesis had to be prioritized. Thus, the sequence models were never in a state where they could properly be used as a replacement nor addition to any of the other functions of the pipeline in a production environment.

The sequence models may have had issues due to a lack of training data, too little training, configuration issues or a combination of these. However, there were more fundamental issues with the approach. By penalizing the loss between bounding boxes of adjacent frames for missing values, the prediction with the lowest possible loss would have been to place the bounding box directly between the two adjacent frames. Therefore, it would have been indistinguishable from a linear interpolation given that the model had predicted the bounding box perfectly. In other words, there would not have been any reason to use the sequence models in place of interpolation. Lastly, using the sequence model for smoothing may have worked well. However, the algorithmic smoothing function is interchangeable and may easily be replaced by another, and since there is a myriad of smoothing algorithms available[32], one of them would likely have functioned similarly anyway.

[31]This is an example of the results (the predicted bounding boxes are coloured blue, and the input bounding boxes are coloured red): `https://youtu.be/5PALkWWpv44`

[32]Examples: `https://opendatascience.com/a-short-summary-of-smoothing-algorithms/`, `https://en.wikipedia.org/w/index.php?title=Smoothing&oldid=1084668720`

4.9 AWS Rekognition

The Python scripts provided in the GitHub repository were used to make an example video testing using AWS Rekognition for facial detection and recognition: `https://youtu.be/6mmBYOYsQVU`. Searching for faces stored in a collection may be used equivalently to filtering by known faces, and in the example, the same images as in Section 4.6.1, shown in Figure 4.12, were indexed into the collection which was used when searching for faces. The faces in the video which were matched to the collection are coloured blue, like in Section 4.6.1.

As shown in the video, faces are frequently not detected correctly[33], and the faces in the collection are not correctly identified from the side[34]. Some of the faces which were not detected is likely due to AWS Rekognition only parsing one frame every half second. Thus, anyone entering the frame will be visible for up to half a second before their face is detected and may be blurred. The half-second processing interval also causes some faster movement to be completely missed[35].

Some of these issues may be fixed by additional processing of the output of the AWS Rekognition face search job. However, the job unfortunately does not return any facial features representations, which somewhat limits the possible improvements after the initial facial detection and recognition. AWS Rekognition does make a vector representation of facial features for each face indexed into a collection in order to recognize the faces, but unfortunately only stores them in a backend database without any way of retrieving them directly [14]. The facial features representations could be used in combination with the position of the bounding boxes to fix some of the issues discussed, such as some of the instances of the faces in the collection not being correctly identified. The position of the bounding box could also be used to improve this alone, but this was not implemented due to the clear issues with utilizing AWS Rekognition for the thesis.

Due to the issues discussed, AWS Rekognition is not suitable for the purpose of automatically blurring specific faces in videos in its current state. The facial detection and recognition is simply not accurate enough, and the half-second interval is not suitable for the application. This is, however, likely not the services primary intended use case. Additionally, it is regularly updated and may be more suitable in the future[36].

[33]Examples at timestamps 0:01, 0:30, 0:33 etc.
[34]Example at timestamp 0:36, 0:39, 0:49 etc.
[35]Example timestamped at 1:17: `https://youtu.be/6mmBYOYsQVU?t=77`
[36]See `https://docs.aws.amazon.com/rekognition/latest/dg/document-history.html`

Chapter 5

Conclusion

Overall, the proposed solution to the project turned out very well, as demonstrated by the example videos[1]. A wide range of tools, technologies and areas of research were explored and evaluated, which was really interesting. It was particularly enjoyable to work with a real-world problem, which may directly be of use to others. The work includes a large variety of subjects, such as machine learning, video and media, and cloud technologies, and it was a great learning experience to be able to utilize and combine these subjects in a single project.

The proposed pipeline has a high quality of predictions and a reasonable running time, which was further improved utilizing GPU accelerated hardware. It is by no means perfect, as discussed in Chapter 4, and there should be manual reviews in scenarios where maintaining the anonymity of the subjects is critical. However, for other scenarios, it is likely more than good enough without extensive reviews. Consequently, the proposed solution ended up fulfilling all of the solution requirements to an arguably acceptable degree:

- **Technical requirements -** The solution pipeline is made as technology agnostic as possible, with examples for running it both locally and deployed in AWS. The infrastructure deployment has been automated using Terraform, the same tool used by CuttingRoom, in addition to Python scripts. Consequently, it should be possible to integrate with CuttingRoom's existing platform without too much effort. However, as discussed in Section 3.1.7; although fully functional without it, would be beneficial to make a user interface for selecting which faces should and should

[1]Complete example blurring selected faces: `https://youtu.be/2mkA9qIHXHc`

not be blurred. Blurring known faces works well without any custom user interface by simply providing a list of images, but blurring by time and location would be more difficult without it.

- **Regulatory requirements -** All parts of the proposed pipeline use tooling and software available for commercial use, and should therefore be able to be used without regulatory restrictions in CuttingRoom's production environment[2]. Some of the software offers options for using models which are *not* available for commercial use, but the user would have to actively choose to use it. This has been documented in Section 3.2.6 and Appendix E.
- **Performance requirements -** The performance requirement was intentionally never specifically defined; both the prediction quality and running time requirements need to be determined by CuttingRoom depending on the service they would like to offer their customers. The proposed pipeline has been optimized as best as possible given the various constraints and considerations discussed in the thesis. In a production environment, the total running time of such a job will entirely depend on the input, models used and orchestration of the pipeline. As discussed in Section 4.2, a single input video may be processed in parallel to significantly decrease the running time. The prediction quality is dependent on the models being used for facial detection, alignment, representation and classification, and they may be replaced at any time.

It would have been nice to spend more time to explore a greater number of tools, and further explore the use of sequence models for facial detecting, alignment, representation and classification in video. This is likely a field with great potential, but it requires more time and resources than available for this thesis. Given more time, it would be interesting to try using video motion tracking combined with facial detection and facial recognition to further increase the prediction quality of the pipeline. Additionally, it may be possible to use the differences between frames, rather than the whole frames themselves, to track the movement of faces in the video which could also possibly improve performance. In the case of these areas being explored further, or simply the ever improving facial detection and recognition technology, each part may be replaced independently due to the interchangeable nature of the pipeline components.

[2]The source code, dependencies and references are all listed. This is no guarantee of correctness, and this claim should be validated independently before use.

An alternative, non-technical solution would simply be to ask the video subjects for permission to publish the media in question. In cases where there are few participants, this should not be an issue. For sporting events or other larger events, permission could be a requirement for attending the event. This would, however, be difficult to enforce for past, large scale events, although for large scale events, the majority of the media would very likely be considered situational photography, and therefore not require the permission of the subjects.

Glossary

Bounding box For the purposes of this thesis, the bounding box is a rectangle that surrounds a face, specifying its possition, and potentially the confidence of it being correctly identified. In other cases, it may also define the type of object, but this is not applicable in this case as all objects of interest are faces.

Data processing *"Any action performed on data, whether automated or manual."* [71] This includes collecting, recording, organizing, structuring, storing, using, erasing etc.

Data subject *"The person whose data is processed. These are your customers or site visitors."* [71]

Frame A frame is a single image in the sequence of images which compose a video. Videos are commonly composed of 24, 30 or 60 frames per second.

Interpolation Estimating unknown intermediate values of a function based on known values [69].

Personal data *"Personal data is any information that relates to an individual who can be directly or indirectly identified. Names and email addresses are obviously personal data. Location information, ethnicity, gender, biometric data, religious beliefs, web cookies, and political opinions can also be personal data."*[71] Similarly, pictures and videos where the subjects are clearly identifiable can also be personal data.

Shot *"A video shot is composed of a series of interrelated consecutive frames. It usually represents a continuous action in time and space. These frames in a shot are related in contents."* [68]

List of Acronyms and Abbreviations

API - Application Programming Interface
AWS - Amazon Web Services
CLI - Command-Line Interface
CNN - Convolutional Neural Network
CPU - Central Processing Unit
GDPR - General Data Protection Regulation
GiB - Gibibyte
GPU - Graphics Processing Unit
GRU - Gated Recurrent Unit
HoG - Histogram of Oriented Gradients
HTTP - Hypertext Transfer Protocol
IaC - Infrastructure as Code
JSON - JavaScript Object Notation
LSTM - Long Short-Term Memory
MTCNN - Multi-Task Cascaded Convolutional Network
ONNX - Open Neural Network Exchange
RNN - Recurrent Neural Network
SSD - Single Shot Detector

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Use a gpu, 2022. **URL:** `https://www.tensorflow.org/guide/gpu`. [Online; accessed 10-May-2022].

[2] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017. doi: 10.1109/ICEngTechnol.2017.8308186.

[3] Amazon. Aws batch pricing, 2022. **URL:** `https://aws.amazon.com/batch/pricing/`. [Online; accessed 09-June-2022].

[4] Amazon. Use your own inference code with batch transform, 2022. **URL:** `https://docs.aws.amazon.com/sagemaker/latest/dg/your-algorithms-batch-code.html`. [Online; accessed 13-May-2022].

[5] Amazon. Transforminput, 2022. **URL:** `https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_TransformInput.html`. [Online; accessed 13-May-2022].

[6] Amazon. What is amazon elastic container registry?, 2022. **URL:** `https://docs.aws.amazon.com/AmazonECR/latest/userguide/what-is-ecr.html`. [Online; accessed 11-May-2022].

[7] Amazon. Amazon elastic container registry pricing, 2022. **URL:** `https://aws.amazon.com/ecr/pricing/`. [Online; accessed 25-May-2022].

[8] Amazon. What is amazon ec2?, 2022.
URL: `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html`. [Online; accessed 25-June-2022].

[9] Amazon. New – per-second billing for ec2 instances and ebs volumes, 2022.
URL: `https://aws.amazon.com/blogs/aws/new-per-second-billing-for-ec2-instances-and-ebs-volumes/`. [Online; accessed 06-June-2022].

[10] Amazon. Aws pricing, 2022.
URL: `https://aws.amazon.com/pricing/`. [Online; accessed 25-June-2022].

[11] Amazon. Build your own processing container (advanced scenario), 2022.
URL: `https://docs.aws.amazon.com/sagemaker/latest/dg/build-your-own-processing-container.html`. [Online; accessed 11-May-2022].

[12] Amazon. Processings3input, 2022.
URL: `https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_ProcessingS3Input.html`. [Online; accessed 13-May-2022].

[13] Amazon. Searching faces in a collection, 2022.
URL: `https://docs.aws.amazon.com/rekognition/latest/dg/collections.html`. [Online; accessed 20-April-2022].

[14] Amazon. Indexfaces, 2022.
URL: `https://docs.aws.amazon.com/rekognition/latest/APIReference/API_IndexFaces.html`. [Online; accessed 18-June-2022].

[15] Amazon. What is amazon s3?, 2022.
URL: `https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html`. [Online; accessed 25-June-2022].

[16] Amazon. Amazon s3 pricing, 2022.
URL: `https://aws.amazon.com/s3/pricing/`. [Online; accessed 25-May-2022].

[17] Amazon. Sagemaker inference toolkit, 2022.
URL: `https://github.com/aws/sagemaker-inference-toolkit/blob/52cd814baccd64d611427a6e8a23e4b8169b42b3/README.md`. [Online; accessed 13-May-2022].

[18] Amazon. Amazon sagemaker pricing, 2022.
URL: `https://aws.amazon.com/sagemaker/pricing/`. [Online; accessed 25-May-2022].

[19] Amazon. What is amazon sagemaker?, 2022.
URL: `https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html`. [Online; accessed 11-May-2022].

[20] Amazon. Use batch transform, 2022.
URL: `https://docs.aws.amazon.com/sagemaker/latest/dg/batch-transform.html`. [Online; accessed 13-May-2022].

[21] Amazon. Amazon ec2 on-demand pricing, 2022.
URL: `https://aws.amazon.com/ec2/pricing/on-demand/`. [Online; accessed 09-June-2022].

[22] Amazon. Amazon ec2 spot instances pricing, 2022.
URL: `https://aws.amazon.com/ec2/spot/pricing/`. [Online; accessed 09-June-2022].

[23] Amazon. Amazon ec2 g4 instances, 2022.
URL: `https://aws.amazon.com/ec2/instance-types/g4/`. [Online; accessed 06-June-2022].

[24] Amazon. Process data, 2022.
URL: `https://docs.aws.amazon.com/sagemaker/latest/dg/processing-job.html`. [Online; accessed 07-December-2021].

[25] Amazon. What is amazon rekognition?, 2022.
URL: `https://docs.aws.amazon.com/rekognition/latest/dg/what-is.html`. [Online; accessed 20-April-2022].

[26] Giovanni Righini at Universita degli Studi di Milano. Minimum cost bipartite matching (complements of operations research), 2018.
URL: `https://homes.di.unimi.it/righini/Didattica/OttimizzazioneCombinatoria/MaterialeOC/11%20-%20Min%20cost%20bipartite%20matching.pdf`. [Online; accessed 26-May-2022].

[27] The Norwegian Data Protection Authority. Deling av bilder, 2019.
URL: `https://www.datatilsynet.no/personvern-pa-ulike-omrader/internett-og-apper/bilder-pa-nett/`. [Online; accessed 12-March-2022].

[28] The Norwegian Data Protection Authority. Om personopplysningsloven med forordning og når den gjelder, 2021.
URL: `https://www.datatilsynet.no/regelverk-og-verktoy/lover-og-regler/om-personopplysningsloven-og-nar-den-gjelder/`. [Online; accessed 12-March-2022].

[29] The Norwegian Data Protection Authority. Regulations, 2022.
URL: `https://www.datatilsynet.no/en/regulations-and-tools/regulations/`. [Online; accessed 29-March-2022].

[30] Rahul Awati and James Denman. Sharding, 2022.
URL: `https://www.techtarget.com/searchoracle/definition/sharding`. [Online; accessed 13-May-2022].

[31] Justinas Baltrusaitis. Amazon aws accounts for 33service market, 2022.
URL: `https://finbold.com/amazon-aws-statistics/`. [Online; accessed 19-April-2022].

[32] Justinas Baltrusaitis. How amazon makes money, 2022.
URL: `https://www.investopedia.com/how-amazon-makes-money-4587523`. [Online; accessed 19-April-2022].

[33] Liza Brown. Ep. 30 how to use face-off effect in wondershare filmora9, 2022.
URL: `https://filmora.wondershare.com/get-creative/face-replacement.html`. [Online; accessed 09-May-2022].

[34] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546 vol. 1, 2005. doi: 10.1109/CVPR.2005.202. [Accessed 15-May-2022].

[35] CMS. Statistics: Fines imposed over time, 2022.
URL: `https://www.enforcementtracker.com/?insights`. [Online; accessed 09-May-2022].

[36] CuttingRoom. The next-generation video editing, live clipping and publishing platform, 2022.
URL: `https://www.cuttingroom.com/`. [Online; accessed 10-May-2022].

[37] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4685–4694, 2019. doi: 10.1109/CVPR.2019.00482. [Accessed 10-May-2022].

[38] Jiankang Deng, Jia Guo, Evangelos Ververas, Irene Kotsia, and Stefanos Zafeiriou. Retinaface: Single-shot multi-level face localisation in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. [Accessed 10-May-2022].

[39] Rajeev Dhir. Data smoothing definition, 2021.
URL: https://www.investopedia.com/terms/d/data-smoothing.asp. [Online; accessed 25-June-2022].

[40] O. Déniz, G. Bueno, J. Salido, and F. De la Torre. Face recognition using histograms of oriented gradients. *Pattern Recognition Letters*, 32(12):1598–1603, 2011. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2011.01.004.
URL: https://www.sciencedirect.com/science/article/pii/S0167865511000122. [Online; accessed 25-June-2022].

[41] IBM Cloud Education. Docker, 2021.
URL: https://www.ibm.com/in-en/cloud/learn/docker. [Online; accessed 26-April-2022].

[42] José Fonseca. About gprof2dot, 2022.
URL: https://github.com/jrfonseca/gprof2dot/blob/2245ac568b4bb0b97762eca71061075a709d7d86/README.md. [Online; accessed 27-May-2022].

[43] Python Software Foundation. The python profilers, 2022.
URL: https://docs.python.org/3/library/profile.html. [Online; accessed 27-May-2022].

[44] Adam Geitgey. Face recognition, 2020.
URL: https://github.com/ageitgey/face_recognition/blob/87a8449a359fbc0598e95b820e920ce285b8a9d9/README.md. [Online; accessed 10-May-2022].

[45] Jia Guo, Jiankang Deng, Xiang An, and Jack Yu. Insightface: 2d and 3d face analysis project, 2022.
URL: https://github.com/deepinsight/insightface/blob/92a0bb6b0f0ff4266c9a1d285d3fbbd191ac2e96/README.md. [Online; accessed 14-May-2022].

[46] HashiCorp. Terraform, 2022.
URL: https://www.terraform.io/. [Online; accessed 18-April-2022].

[47] HashiCorp. Modules, 2022.
URL: https://registry.terraform.io/browse/modules. [Online; accessed 28-May-2022].

[48] Amir Keshavarz. Cache replacement algorithms: How to efficiently manage the cache storage, 2021.

URL: `https://dev.to/satrobit/cache-replacement-algorithms-how-to-efficiently-manage-the-cache-storage-2ne1`. [Online; accessed 28-May-2022].

[49] Davis E. King and Dlib Authors. Dlib c++ library, 2022.
URL: `http://dlib.net/`. [Online; accessed 10-May-2022].

[50] Davis E. King and Dlib Authors. Dlib c++ library, 2022.
URL: `http://dlib.net/python/index.html`. [Online; accessed 10-May-2022].

[51] Nenad Markuš, Miroslav Frljak, Igor S. Pandžić, Jörgen Ahlberg, and Robert Forchheimer. Object detection with pixel intensity comparisons organized in decision trees, 2013.
URL: `https://arxiv.org/abs/1305.4537`. [Online; accessed 29-October-2021].

[52] Justis og beredskapsdepartementet. Lov om behandling av personopplysninger (personopplysningsloven), 2018.
URL: `https://lovdata.no/dokument/NL/lov/2018-06-15-38`. [Online; accessed 26-October-2021].

[53] Python Packagin Authority (PyPa). Project summaries, 2022.
URL: `https://packaging.python.org/en/latest/key_projects/`. [Online; accessed 18-March-2022].

[54] Lourdes Ramirez Cerna, G. Cámara-Chávez, and D. Menotti. Face detection: Histogram of oriented gradients and bag of feature method, 2013. [Accessed 25-June-2022].

[55] Meta AI Research. Face alignment, 2021.
URL: `https://paperswithcode.com/task/face-alignment`. [Online; accessed 14-May-2022].

[56] Adrian Rosebrock. Face detection with dlib (hog and cnn), 2021.
URL: `https://pyimagesearch.com/2021/04/19/face-detection-with-dlib-hog-and-cnn/`. [Online; accessed 18-April-2022].

[57] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823, 2015. doi: 10.1109/CVPR.2015.7298682.

[58] Sefik Ilkin Serengil. Retinaface, 2022.
URL: `https://github.com/serengil/retinaface/blob/34b1ec11a4a0beee2ebd2c095742b3d070e23fb5/README.md`. [Online; accessed 10-May-2022].

[59] Sefik Ilkin Serengil and Alper Ozpinar. Lightface: A hybrid deep face recognition framework. In *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*, pages 23–27. IEEE, 2020. doi: 10.1109/ASYU50717.2020.9259802.
URL: `https://doi.org/10.1109/ASYU50717.2020.9259802`.

[60] Sefik Ilkin Serengil and Alper Ozpinar. Hyperextended lightface: A facial attribute analysis framework. In *2021 International Conference on Engineering and Emerging Technologies (ICEET)*, pages 1–4. IEEE, 2021. doi: 10.1109/ICEET53442.2021.9659697.
URL: `https://doi.org/10.1109/ICEET53442.2021.9659697`. [Online; accessed 25-June-2022].

[61] Endre Simo. esimov/pigo, 2021.
URL: `https://github.com/esimov/pigo/blob/131dc573e45a067006e318d7072e49070420e98d/README.md`. [Online; accessed 09-October-2021].

[62] Carol Sliwa. Read cache, 2013.
URL: `https://www.techtarget.com/searchstorage/definition/read-cache`. [Online; accessed 28-May-2022].

[63] Tomáš Souček and Jakub Lokoč. Transnet v2: An effective deep network architecture for fast shot transition detection, 2020.
URL: `https://arxiv.org/abs/2008.04838`. [Online; accessed 03-May-2022].

[64] Tomáš Souček, Jaroslav Moravec, and Jakub Lokoč. Transnet: A deep network for fast detection of common shot transitions, 2019.
URL: `https://arxiv.org/abs/1906.03363`. [Online; accessed 03-May-2022].

[65] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification, 2014.
URL: `https://research.facebook.com/publications/deepface-closing-the-gap-to-human-level-performance-in-face-verification/`. [Online; accessed 10-May-2022].

[66] Corporate Finance Institute Team. Data smoothing, 2021.
URL: `https://corporatefinanceinstitute.com/resources/knowledge/other/data-smoothing/`. [Online; accessed 25-June-2022].

[67] FFmpeg team. Ffmpeg readme, 2021.
URL: `https://github.com/FFmpeg/FFmpeg/blob/52a14b8505923116ed6acc5e691c0c7c44e6f708/README.md`. [Online; accessed 8-May-2022].

[68] Shaohua Teng, Wenwei Tan, and Wei Zhang. *Cooperative Shot Boundary Detection for Video*, page 99–110. Springer-Verlag, Berlin, Heidelberg, 2008. ISBN 9783540927181.
URL: `https://doi.org/10.1007/978-3-540-92719-8_10`. [Online; accessed 03-May-2022].

[69] Wikipedia contributors. Interpolation — Wikipedia, the free encyclopedia, 2022.
URL: `https://en.wikipedia.org/w/index.php?title=Interpolation&oldid=1068675581`. [Online; accessed 13-March-2022].

[70] Holly Wilper, Robert Knight, and Jason Cohen. Understanding the visualization of overhead and latency in nvidia nsight systems, 2020.
URL: `https://developer.nvidia.com/blog/understanding-the-visualization-of-overhead-and-latency-in-nsight-systems/`. [Online; accessed 06-June-2022].

[71] Ben Wolford. What id gdpr, the eu's new data protection law?, 2022.
URL: `https://gdpr.eu/what-is-gdpr`. [Online; accessed 04-March-2022].

[72] Peipei Xia, Li Zhang, and Fanzhang Li. Learning similarity with cosine similarity ensemble. *Information Sciences*, 307:39–52, 2015. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2015.02.024.
URL: `https://www.sciencedirect.com/science/article/pii/S0020025515001243`. [Online; accessed 25-June-2022].

[73] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, oct 2016. doi: 10.1109/lsp.2016.2603342.
URL: `https://doi.org/10.1109%2Flsp.2016.2603342`. [Online; accessed 14-May-2022].

Appendix A

Thesis description

Automatic blurring of selected faces in video

This Master's thesis will explore solutions to automate the process of blurring a selected group of faces, or alternatively all faces *except* for a selected group of faces. The blurring of faces is mainly intended to anonymize people present in the background of videos, thus assisting in following various regulations regarding the distribution of media containing identifiable people, such as the Personal Data Act in Norway[1].

The solution is intended to be integrable with Vimond IO such that it may be used in IO if it is considered to be beneficial to the product. The thesis will therefore take into account the requirements of IO. This doesn't necessarily mean the solution will be used after completion of the thesis. Nor is the solution critical to the operation of IO in any way, rather it would be a "nice to have" feature.

The requirements to be considered include the following:

- Technical requirements - The resulting solution should be integrable with Vimond IO and must be designed in a way which allows it to do so.
- Regulatory requirements - The licences of the software and data being used in the thesis must allow for the solution to be used in a commercial product.
- Performance requirements - The videos being edited may be several hours long. If the solution takes too long to run, or is not accurate enough, it will essentially be unusable.

In order to blur faces, the solution needs to both detect and localize all faces in each frame of the video. It also needs to recognize the faces in order to determine which faces should and should not be blurred. Additionally, segmentation may also be beneficial in order to accurately distinguish a face from its surroundings and restrict the blur to the face specifically, thereby preserving the rest of the frame. However, this will probably be more difficult to find both existing solutions and datasets for segmentation compared to localization.

The thesis will explore the use various machine learning models, such as RNNs and SVMs, in addition to building upon pre-existing solutions, such as AWS Rekognition[2]. It may also be useful to make use of pre-trained models for facial recognition, thus lessening the time spent training models and the need for labeled data. No labeled data will be provided for the thesis, although some unlabeled data may be received from TV2 news.

After completion of the thesis, the code produced will be published along with the thesis, although certain parts may be excluded if necessary.

[1] https://www.datatilsynet.no/personvern-pa-ulike-omrader/internett-og-apper/bilder-pa-nett/
[2] https://aws.amazon.com/rekognition

Appendix B

Prediction quality video analysis numbers

The following is a list of each frame which was considered to be false positives, false negatives, true negatives and true positives for each video calculating performance numbers. The frame numbers start from 0 and end at 4586. The ranges are inclusive.

Full processing Deepface(RetinaFace and FaceNet512)

- **False positive (23 total):** 2505-2506, 2833-2838, 2977-2978, 2995-3007
- **False negative (67 total):** 3013-3061, 3100-3117
- **True negatives (0 total):** N/A
- **True positives (4497 total):** 0-2504, 2507-2832, 2839-2976, 2979-2994, 3008-3012, 3062-3099, 3118-4586

Dynamic processing Deepface (RetinaFace and FaceNet512)

- **False positive (6 total):** 2505-2506, 2833-2834, 2977-2978
- **False negative (67 total):** 3013-3061, 3100-3117
- **True negatives (0 total):** N/A
- **True positives (4514 total):** 0-2504, 2507-2832, 2835-2976, 2979-3012, 3062-3099, 3118-4586

Full processing Dlib

- **False positive (38 total):** 1878-1891, 1905-1906, 1910-1911, 2345, 2349-2354, 2850-2853, 3865-3869, 4556-4559
- **False negative (766 total):** 0-298, 545-578, 834-1185, 2305-2307, 3013-3061, 3100-3117, 3630-3640
- **True negative (0 total):** N/A
- **True positive (3783 total):** 299-544, 579-833, 1186-1877, 1892-1904, 1907-1909, 1912-2304, 2308-2344, 2346-2348, 2355-2849, 2854-3012, 3062-3099, 3118-3629, 3641-3864, 3870-4555, 4560-4586

Dynamic processing Dlib

- **False positive (33 total):** 1887-1889, 1905-1906, 2349-2354, 2831-2833, 2850-2853, 3865-3869, 4550-4559
- **False negative (801 total):** 0-298, 545-607, 834-1185, 2305-2313, 3013-3061, 3100-3117, 3630-3640
- **True negative (0 total):** N/A
- **True positive (3753 total):** 299-544, 608-833, 1186-1886, 1890-1904, 1907-2304, 2314-2348, 2355-2830, 2834-2849, 2854-3012, 3062-3099, 3118-3629, 3641-3864, 3870-4549, 4560-4586

Dynamic processing Deepface (removing short sequences)

- **False positive (2 total):** 2977-2978
- **False negative (67 total):** 3013-3061, 3100-3117
- **True negatives (0 total):** N/A
- **True positives (4518 total):** 0-2976, 2979-3012, 3062-3099, 3118-4586

Full processing Dlib (without interpolation)

- **False positive (6 total):** 1887-1889, 1910-1911, 3869,
- **False negative (909 total):** 0-298, 545-578, 580, 582-583, 590-598, 602-607, 651-652, 655-663, 668-670, 834-1185, 1872-1873, 1876-1886, 1890-1891, 1896-1902, 1905-1906, 2305-2307, 2310-2313, 2570-2571, 2849-2854, 2925-2931, 2954-2955, 3013-3061, 3100-3117, 3603-3604, 3630-3640, 3810-3826, 3881-3883, 3886-3887, 4032-4049, 4054-4059, 4441, 4457-4464, 4529-4535, 4584-4585
- **True negatives (3672 total):** N/A
- **True positives (3672 total):** 299-544, 579, 581, 584-589, 599-601, 608-650, 653-654, 664-667, 671-833, 1186-1871, 1874-1875, 1892-1895, 1903-1904, 1907-1909, 1912-2304, 2308-2309, 2314-2569, 2572-2848, 2855-2924, 2932-2953, 2956-3012, 3062-3099, 3118-3602, 3605-3629, 3641-3809, 3827-3868, 3870-3880, 3884-3885, 3888-4031, 4050-4053, 4060-4440, 4442-4456, 4465-4528, 4536-4583, 4586

Appendix C

List of example videos

The following is a list of the various example videos linked throughout the thesis.

- Original example video: `https://youtu.be/_5i5kza5C-M`
- Complete example, dynamically processed with Deepface, selectively blurring faces based on time and location: `https://youtu.be/2mkA9qIHXHc`
- Processing every frame (Deepface): `https://youtu.be/tORTBS7iEGY`
- Processing every frame (Dlib): `https://youtu.be/nnXTJOOwLiQ`
- Dynamically processed (Deepface): `https://youtu.be/73XS75-tdYQ`
- Dynamically processed (Dlib): `https://youtu.be/v_49prmpeec`
- Processing every frame without interpolation (Dlib): `https://youtu.be/nSvN24R_wRU`
- Filtering short sequences: `https://youtu.be/yaW0VuOyGPk`
- Selecting faces by time & location: `https://youtu.be/fkTlFBNTb94`
  - Additional selection: `https://youtu.be/cvGPJ06klzc`
- Selecting known faces: `https://youtu.be/Wf_yMDSXteo`
  - Adding additional face: `https://youtu.be/AJNC7OxWvrs`
- Without detected shot transitions: `https://youtu.be/IuDkF_XdqWc`
- Marked shot transitions (cuts): `https://youtu.be/sTvsOq-1orQ`
- Marked shot transitions (gradual transitions): `https://youtu.be/ZBVLBcMtlIE`
- Example LSTM predictions: `https://youtu.be/5PALkWWpv44`
- Processed using AWS Rekognition: `https://youtu.be/6mmBYOYsQVU`
- Processed using Pigo: `https://youtu.be/hQ7EhRiJKdo`

Example videos used for testing running times:

- Multiple faces: `https://youtu.be/HZz4862_lII`
- Single face: `https://youtu.be/VXteG6A2ME0`
- No face: `https://youtu.be/Mku0Um84Iew`

Appendix D

Memory usage overview

```
        Line #    Mem usage    Increment  Occurrences   Line Contents
        =============================================================
            11   1554.0 MiB   1554.0 MiB           1   @profile
            12                                         def main():
            13   1554.0 MiB      0.0 MiB           1       img_dir =
            ↪  "/mnt/sdb3/erlend/multiface_cut"
            14   1554.0 MiB      0.0 MiB           1       out_dir =
            ↪  "/mnt/sdb3/erlend/test"
            15
            16   2256.9 MiB    702.9 MiB           1       shot_transitions =
            ↪  get_shot_transitions(img_dir)
            17   2879.6 MiB    622.7 MiB           1       proc_frames, matchings =
            ↪  dynamically_process(img_dir, shot_transitions=shot_transitions)
            18   2879.6 MiB      0.0 MiB           1       seqs =
            ↪  make_sequences(proc_frames,
            ↪  matchings,shot_transitions=shot_transitions)
            19   2885.7 MiB      6.1 MiB           1       seqs = interpolate(seqs)
            20   2278.4 MiB   -607.2 MiB           1       write_faces(seqs, img_dir,
            ↪  out_dir)
            21   2277.3 MiB     -1.1 MiB           1
            ↪  copy_remaining_files(img_dir, out_dir)
```

Listing 21: Overview of memory usage by line of source code when reading frames during processing

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    15   1577.8 MiB   1577.8 MiB           1   @profile
    16                                         def main():
    17   1577.8 MiB      0.0 MiB           1       img_dir =
    ↪   "/mnt/sdb3/erlend/multiface_cut"
    18   1577.8 MiB      0.0 MiB           1       out_dir = "/mnt/sdb3/erlend/test"
    19
    20   1577.8 MiB      0.0 MiB           1       frames = []
    21  13452.5 MiB  -1039.6 MiB        4588       for filepath in
    ↪   sorted(glob(join(img_dir, "*.png"))):
    22  13452.5 MiB  10849.3 MiB        4587
    ↪   frames.append(imread(filepath))
    23
    24  13452.5 MiB      0.0 MiB           1       print(len(frames))
    25
    26  14168.6 MiB    716.1 MiB           1       shot_transitions =
    ↪   get_shot_transitions(img_dir, frames=frames)
    27  14168.6 MiB    -42.8 MiB           2       proc_frames, matchings =
    ↪   dynamically_process(
    28  14168.6 MiB      0.0 MiB           1           img_dir,
    ↪   shot_transitions=shot_transitions, frames=frames)
    29  14125.8 MiB    -42.8 MiB           2       seqs = make_sequences(proc_frames,
    ↪   matchings,
    30  14125.8 MiB      0.0 MiB           1
    ↪   shot_transitions=shot_transitions)
    31  14132.2 MiB      6.4 MiB           1       seqs = interpolate(seqs)
    32  13988.7 MiB   -143.4 MiB           1       write_faces(seqs, img_dir,
    ↪   out_dir, frames=frames)
    33  13990.1 MiB      1.4 MiB           1       copy_remaining_files(img_dir,
    ↪   out_dir)
```

Listing 22: Overview of memory usage by line of when reading every frame only once before processing

Appendix E

Deepface model licenses

The following is a list of their licenses.

- DeepFace model: MIT [1]
- VGG-Face: Non-commercial Creative Commons Attribution [2]
- Facenet: MIT [3]
- OpenFace: Apache 2.0 [4]
- DeepID: GNU 3 [5]
- ArcFace: MIT [6]

[1] `https://github.com/swghosh/DeepFace/blob/master/LICENSE`

[2] `https://www.robots.ox.ac.uk/~vgg/software/vgg_face/`

[3] `https://github.com/davidsandberg/facenet/blob/master/LICENSE.md`

[4] `https://github.com/iwantooxxoox/Keras-OpenFace/blob/master/LICENSE`

[5] `https://github.com/Ruoyiran/DeepID/blob/master/LICENSE.md`

[6] `https://github.com/leondgarse/Keras_insightface/blob/master/LICENSE`