UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE

DIPARTIMENTO DI INFORMATICA
GIOVANNI DEGLI ANTONI

Corso di Laurea magistrale in
Informatica

DE - IDENTIFICATION METHODS FOR VISUAL PRIVACY PROTECTION

Relatore: Prof. Raffaella Lanzarotti

Tesi di Laurea di:
Giorgio Borghetto
Matr. Nr. 920764

ANNO ACCADEMICO 2019-2020

Thanks

Alla mia preziosa famiglia.

Contents

Chapter 1

Introduction

Recent advances in both camera technology as well as supporting computing hardware have made it significantly easier to deal with large amounts of visual data. This enables a wide range of new usage scenarios involving the acquisition, processing and sharing of images. Examples include the development of intelligent monitoring systems for video surveillance and ambient-assisted living. By using this technology, these systems are able to automatically interpret visual data from the environment and perform tasks that would have been unthinkable years ago. These achievements represent a radical improvement but they also suppose a new threat to individual's privacy.

Unlike images and videos enacted by consented subjects, for those collected from real world, the law requires that the privacy of the people inadvertently captured by camera need to be protected before such data can be used. The General Data Protection Regulation (GDPR) came to effect as of 25th of May, 2018, affecting all processing of personal data across Europe. GDPR requires regular consent from the individual for any use of their personal data. However, if the data does not allow to identify an individual, companies are free to use the data without consent.

Considering the new capabilities of such systems that give them the ability to collect and index a huge amount of private information about each individual, and the fact that the face is the most identifiable part of a human, visual anonymity can be achieved by changing the faces, a problem commonly known as face De-Identification.

Every face De-Id (face De-Identification) process starts with a face image (or simple 'face' or 'image'), which is a 3D matrix I of m columns, n rows, and c channels. c is usually 3 in common color spaces (e.g. RGB and YUV). Each cell in I stores a color coding for a pixel, ranging from 0 to 255 inclusively and a face image contains a normalized image of only one person's face. Considering a face set H of M face images, $H = \{I_i, ..., I_M\}$, H is said to be person-specific if and only

if each $I_i \in H$ only relates to one person and $I_i \neq I_j$, for any $i \neq j$. Afterwards, a specific function called Face De-Id Function attempts to obfuscate the identity of the original face image, transforming the Previous H set into a De-Id one, H^d.

For this task, the dilemma is that on the one hand, we want the De-Identified image to look as different as possible from the original image to ensure the removal of identity; on the other hand, we expect the De-Identified image to retain as much structural information in the original image as possible so that the image utility remains.

This, applied to video, is a challenging computational task: the video needs to be modified in a seamless way, without causing flickering or other visual artifacts and distortions, such that the identity is changed, while all other factors remain identical. These factors include pose and expression, occlusion, illumination and shadow, and their dynamics. Many early works on face De-Id are based on deteriorating the image by blurring, pixelization, image segmentation, downsampling, deletion of part of the face, or cartoonizing. While these methods can effectively remove identity-related information from images, they also get rid of such useful facial attributes which are loosely related to personal identity, making difficult to adopt in further applications.

More sophisticated face De-Id methods focus on changing faces rather than removing features related to their identity. Early works generate De-Id faces by removing high frequency details, but they usually lead to faces with blurred appearances. The developments of image synthesis methods based on deep neural networks, in particular, generative adversary networks (GANs), inspire a new vein of face De-Id methods, which use synthesized faces to replace the originals. Furthermore, another deep generative model used for this task is the Encoder-Decoder architecture which converts the input face to a representative feature (the "code"), and then reverses the process to synthesize a face from the code. As mentioned before, a set of face images of the same subject is called a face set. Different face sets share the same Encoder, but each has a dedicated Decoder. This specific structure is to ensure the encoder to capture the identity-independent attributes common to all face sets, while the individual decoders can preserve identity-dependent attributes of each subject and map such attributes onto the synthesized faces.

Another approach is to use face swapping methods for Face De-Id. Face swap is commonly related to the term deepfake (stemming from "deep learning" and "fake"), i.e. a technique that can superimpose face images of a target subject to a video of a source subject to create a video of the target subject doing or saying things the source subject does. Deep learning models such as the aforementioned ones have been applied widely in deepfake algorithms to examine facial expressions and movements of a person and synthesize facial images of another person making analogous expressions and movements. Face swap algorithms normally require a

large amount of image and video data to train models to create photo-realistic images and videos. As public figures such as celebrities and politicians may have a large number of videos and images available online, they were initial targets of face swapping. Despite deepfakes have being created only and exclusively for recreational and entertainment purposes, current literature on De-Identification has shown a great deal of robustness to the source image as well as for the properties of the image, from which the target face is taken. While these classical face swapping methods work in the pixel space and copy the expression of the target image, recent deep-learning based work swap the identity, while maintaining the other aspects of the source image.

This work is focused on the last approach and seeks to clarify how privacy can be protected in video data through several selected face swap tools, so as a main contribution a comprehensive classification of the protection methods used in the face swapping technique for visual privacy as well as an up-to-date review of them are provided. The selected tools are configured at the best of their capability, in order to ensuring the best possible quality for the final swapping of a specific dataset composed by 24 actors which are characterized by differences between sex, ethnicity and physical aspect. Furthermore, several metrics are used and described in order to evaluate qualitative and quantitative measurement of the degree of privacy applied by each tool, considering how well a tool conceals the identity of the original subject and retains useful facial information, providing a complete comparison between them.

Chapter 2

Related work

The vast majority of early algorithms for face De-Identification fall into one of two groups: ad-hoc distortion/suppression methods and the k-Same family of algorithms implementing the k-anonymity protection model. More recently, deep neural networks as Generative Adversary Networks (GANs) and Encoder-Decoder architectures are considered the state-of-the-art for De-Identification tasks.

These categories of algorithms along with their weaknesses are described in the following sections.

2.1 Naïve De-Identification Methods

Following similar practices in traditional print and broadcasting media, image distortion approaches to face De-Identification alter the region of the image occupied by a person using data suppression or simple image filtering.

These ad-hoc methods have been discussed many times in the literature, often in the context of computer supported cooperative work (CSCW) and home media spaces where explicit user control is desired to balance between privacy and data utility. Image filtering approaches use simple obfuscation methods such as blurring (smoothing the image with e.g. a Gaussian filter with large variance) or pixelation (image sub-sampling). [1][2][3]

While these algorithms are applicable to all images, they lack a formal privacy model. Therefore no guarantees can be made that the privacy of people visible in the images is actually protected. As a consequence naïve De-Identification methods neither preserve privacy nor data utility as results presented in the next section show.[4][5][6]

Figure 1: Comparison of some of the canonical image processing methods for face obfuscation. From left to right: original image, blurred image, pixelated image and masked image.

2.1.1 Weaknesses of Naïve De-Identification Methods

While naïve De-Identification algorithms such as blurring and pixelation have been shown to successfully thwart human recognition, they lack an explicit privacy model and are therefore vulnerable to comparatively simple attacks.

A very effective approach to defeat naïve De-Identification algorithms was proposed by [7] as (manual) parrot recognition. Instead of comparing De-Identified images to the original images (as humans implicitly or explicitly do), parrot recognition applies the same distortion to the gallery images as contained in the probe images prior to performing recognition. As a result, recognition rates drastically improve, in effect reducing the privacy protection offered by the naïve De-Identification algorithms. [8]

Authors in [4] demonstrated this empirically using frontal images from 228 subjects from the CMU Multi-PIE database displaying neutral, smile, and disgust expressions. [9] Images were shape normalized using manually established Active Appearance Model labels.

Figure 2: Examples of the CMU Multi-PIE database. From left to right: neutral, smile and disgust.

Subsequently, [4] built Principle Component Analysis (PCA) bases on a small subset of the data (68 subjects representing 30% of the data) and encode the remainder of the data using these basis vectors. With the neutral face images as gallery and smile and disgust images of varying blur and pixelation levels as probes, [4] computed recognition rates using a whitened cosine distance, which has been shown to perform well in face PCA spaces.

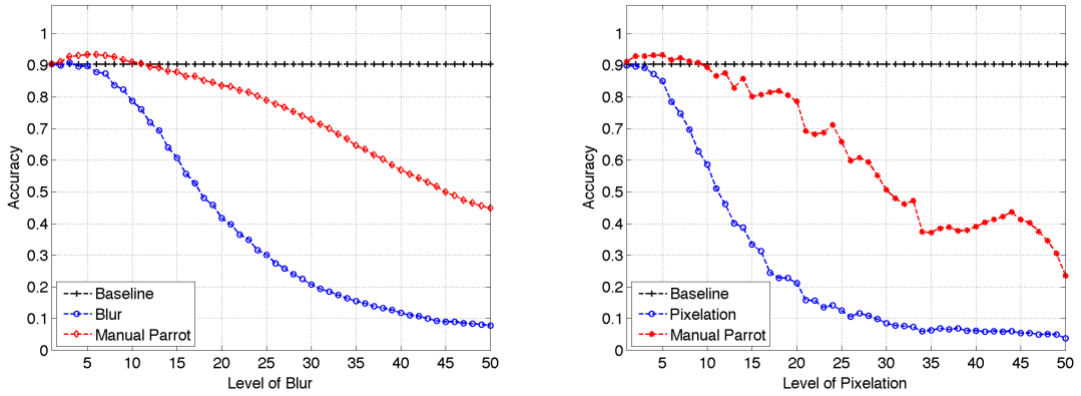


Figure 3: Manual parrot recognition (in red) for both blurred and pixelated images.

In Figure 3 the accuracies of De-Identified images are compared to parrot recognition rates. For both blurring and pixelation, parrot recognition rates are significantly higher than the original De-Identification rates. For low parameter settings of either algorithm, parrot recognition performs even better than using original, unaltered images in both gallery and probe. This is likely due to a reduction in image noise in De-Identified images.

The knowledge of the amount of blurring or pixelation present in the probe images was used to De-Identify the gallery images with the same amount. This information, however, can be extracted directly from the De-Identified probe images for an automatic parrot attack. In the case of pixelation we simply determine the size of blocks of equal (or approximately equal) pixel intensities in the image. A

similar procedure can be applied in the case of blurring by analyzing the frequency spectrum of the De-Identified images.

2.2 k-Same family of algorithms

The k-Same family of algorithms implements the k-anonymity protection model for face images. [10] [4] [7]

Given a person-specific set of images $H = \{\mathbf{I}_1, ..., \mathbf{I}_M\}$, where each subject is represented by no more than one image, k-Same computes a De-Identified set of images $H^d = \{\mathbf{I}_1^d, ..., \mathbf{I}_M^d\}$ in which each $\mathbf{I}_i^d$ indiscriminately relates to at least k elements of H.

It can then be shown that the best possible success rate for a face recognition algorithm linking an element of H^d to the correct face in H is $\frac{1}{k}$. k-Same achieves this k-anonymity protection by averaging the k closest faces for each element of H and adding k copies of the resulting average to H^d.

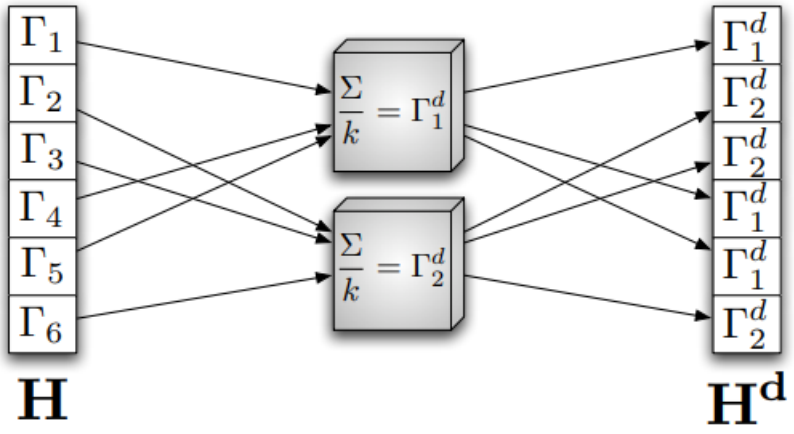


Figure 4: Overview of the k-Same algorithm. Images are De-Identified by computing averages over the closest neighbors of a given face in H and adding k copies of the resulting average to H^d. [4]

While k-Same provides provable privacy guarantees, the resulting De-Identified images often contain undesirable artifacts. Since the algorithm directly averages pixel intensity values, even small alignment errors of the underlying faces lead to "ghosting" artifacts.

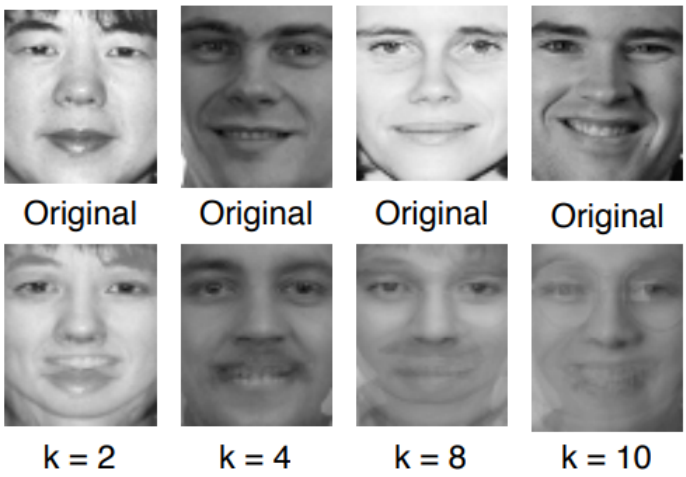


Figure 5: Faces shown in the image above were De-Identified using the appearance-based version of k-Same. Due to misalignments in the face set, ghosting artifacts appear.

2.2.1 k-Same-Select and k-Same-M

To overcome this problem, two extensions to k-Same, referred to as k-Same-M and k-Same-Select, have been developed.

On the one hand, k-Same-Select extends the k-Same algorithm ensuring the utility of the data. This algorithm divides the face image set into mutually exclusive subsets using a data utility function defined as

$$\Phi = \mathbb{R}^m \rightarrow \mathbb{R}$$

which assignes a utility score to an m-dimensional image vector, then k-Same-Select uses the k-Same algorithm on the different subsets. Examples for data utility functions for face image data include gender and facial expression classification accuracies as well as raw image distances.

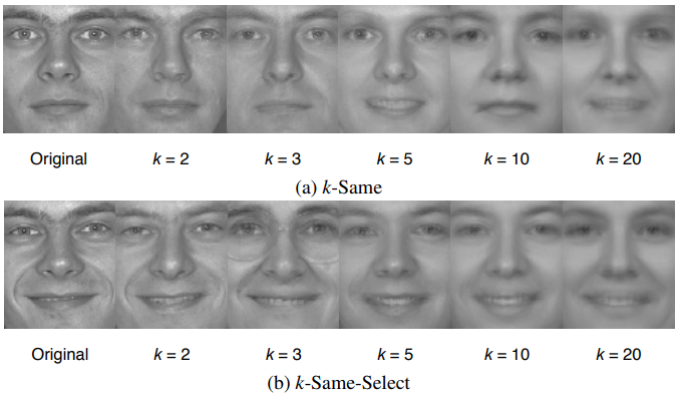


Figure 6: Examples of applying k-Same and k-Same-Select to expression variant faces. Since k-SameSelect factors facial expression labels into the image selection process, facial expressions are preserved better (notice the changing expression in the first row). Both algorithm provide k-anonymity privacy protection.

On the other hand, the k-Same-M algorithm was introduced to fix a shortcoming of the k-Same-Select algorithm. Appearance-based algorithms, like k-Same-Select, work directly in the pixel level producing sometimes alignment mismatch that leads to undesirable artefacts in the De-Identified face images.

In order to overcome this issue k-Same-M relies on Active Appearance Model (AAM), a statistical and photo-realistic model of the shape and texture of faces. The resulting De-Identified images have much higher quality than images produced by k-Same while the same privacy guarantees can still be made.

Figure 7: Faces De-Identified using k-Same-M, the model-based extension of k-Same. In comparison with the k-Same, the images produced by k-Same-M are of much higher quality.

2.2.2 Weaknesses of k-Same family of algorithms

The aforementioned methods represent the majority of approaches used for De-Identification. However, these methods have notable limitations:

- k-Same assumes that each subject is only represented once in the dataset, but this may be violated in practice. The presence of multiple images from same subject or images share similar biometric characteristics can lead to lower levels of privacy protection.
- k-Same operates on a closed set and produces a corresponding De-Identified set, which is not applicable in situations that involve processing individual images or sequences of images.
- Generated images by AAM do not yet look natural enough.

Referring to the first point, [4] reported results of experiments on the Multi-PIE database. Each face in the dataset is represented using the appearance coefficients of an AAM. Recognition is performed by computing the nearest neighbors in the appearance coefficient space.

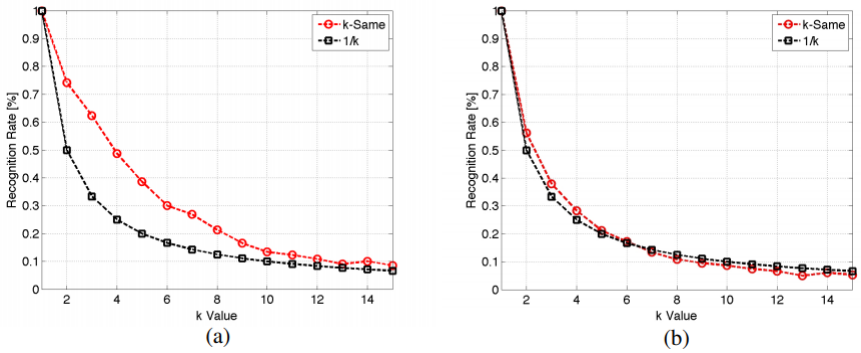


Figure 8: Recognition performance of k-Same on image sets containing multiple faces per subject. (a) shows recognition accuracies after applying k-Same to a subset of the CMU Multi-PIE database containing multiple expressions (neutral, surprise, and squint) of each subject. (b) shows recognition accuracies after applying k-Same on an illumination-variant subset of Multi-PIE. For both datasets recognition accuracies exceed $\frac{1}{k}$, indicating lower levels of privacy protection. [11]

In the first experiment [4] employed images of 203 subjects in frontal pose and frontal illumination, displaying neutral, surprise, and squint expressions. In the second experiment [4] used images of 249 subjects recorded in frontal pose,

displaying neutral expressions. Images of five illumination conditions per subject were included in the dataset.

In either case, k-Same failed to provide adequate privacy protection. For the expression-variant dataset, accuracies stayed well above the $\frac{1}{k}$ rate guaranteed by k-Same for datasets with single examples per class. The same observation holds for the illumination-variant dataset for low k values. We obtain similar results even when class information is factored into the k-Same De-Identification process. [4] concluded that k-Same does not provide sufficient privacy protection if multiple images per subject are included in the dataset.

2.3 Deep generative models

A deep generative model is a powerful way of learning any kind of data distribution using unsupervised learning. All types of generative models aim at learning the true data distribution of the training set so as to generate new data points with some variations. A Generative Adversarial Network is a highly successful training architecture to model a natural image distribution. GANs enable us to generate new images, often indistinguishable from the real data distribution. They have a broad diversity of application areas, from general image generation to text-to-photo generation, style transfer and much more. With the numerous contributions since their conception, they have gone from a theoretical idea to a tool that a user can apply for practical use cases. GANs are an efficient tool to remove privacy-sensitive information without destroying the original image quality.

A GAN has two parts in it: the generator that generates images and the discriminator that classifies real and fake images.

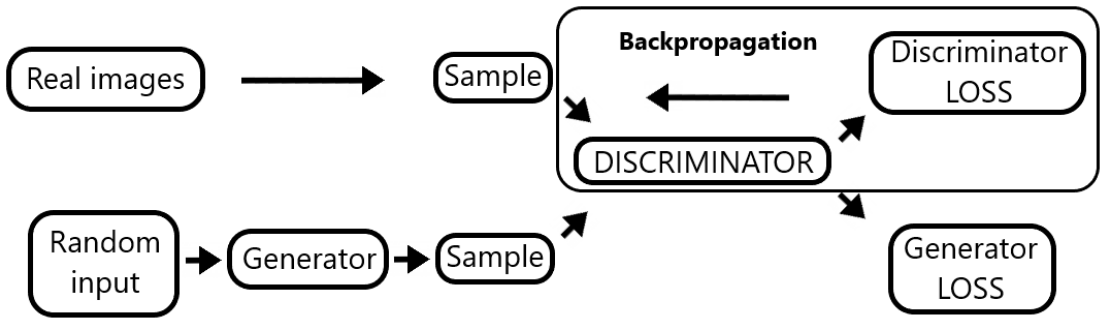


Figure 9: Backpropagation in discriminator training.

The discriminator in a GAN is simply a classifier. It tries to distinguish real data from the data created by the generator. It could use any network architecture

appropriate to the type of data it's classifying. The discriminator's training data comes from two sources:

- Real data instances: such as real pictures of people. The discriminator uses these instances as positive examples during training.
- Fake data instances: created by the generator. The discriminator uses these instances as negative examples during training.

In Figure 9, the two "Sample" boxes represent these two data sources feeding into the discriminator. During discriminator training the generator does not train. Its weights remain constant while it produces examples for the discriminator to train on.

The discriminator connects to two loss functions. During discriminator training, the discriminator ignores the generator loss and just uses the discriminator loss. The generator loss is used during generator training, as described in the next section. During discriminator training:

1. The discriminator classifies both real data and fake data from the generator.
2. The discriminator loss penalizes the discriminator for misclassifying a real instance as fake or a fake instance as real.
3. The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.

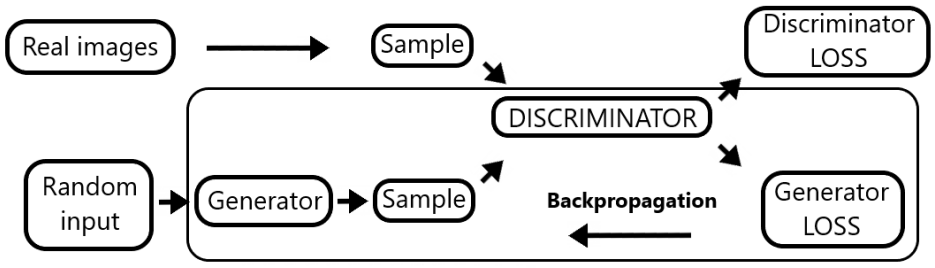


Figure 10: Backpropagation in generator training.

The generator part of a GAN learns to create fake data by incorporating feedback from the discriminator. It learns to make the discriminator classify its output as real. Generator training requires tighter integration between the generator and the discriminator than discriminator training requires. The portion of the GAN that trains the generator includes:

- Random input. as positive examples during training.
- Generator network, which transforms the random input into a data instance.
- Discriminator network, which classifies the generated data.
- Discriminator output.
- Generator loss, which penalizes the generator for failing to fool the discriminator.

In its most basic form, a GAN takes random noise as its input. The generator then transforms this noise into a meaningful output. By introducing noise, the GAN can produce a wide variety of data, sampling from different places in the target distribution. During generator training:

1. Sample random noise.
2. Produce generator output from sampled random noise.
3. Get discriminator "Real" or "Fake" classification as generator output.
4. Calculate loss from discriminator classification.
5. Backpropagate through both the discriminator and generator to obtain gradients.
6. Use gradients to change only the generator weights.

The generator and the discriminator have different training processes. Training a GAN as a whole proceeds in alternating periods, firstly training the discriminator for one or more epochs and then the discriminator. The generator is kept constant during the discriminator training phase. As discriminator training tries to figure out how to distinguish real data from fake, it has to learn how to recognize the generator's flaws. That is a different problem for a deeply trained generator than it is for an untrained generator that produces random output. Similarly, the discriminator is kept constant during the generator training phase. Otherwise the generator would be trying to hit a moving target and might never converge. [12]

Several deep generative model based frameworks are reported below:

2.3.1 AnonymousNet

[13] refers to one of the most powerful framework called AnonymousNet, which tries to address the issues of balance usability, and enhance privacy in a natural and measurable manner using deep generative models.

This framework is composed of 4 main stages:

- Facial semantic extraction powered by a deep Convolutional Neural Network.
- Attribute selection method with regards to privacy metrics such as k-anonymity, l-diverse [14], and t-closeness [15].
- Generative Neural Network for photo-realistic image generator.
- Universal adversarial perturbation to mitigate potential security and privacy threats.

AnonymousNet adopts GoogLeNet [16] for facial attribute extraction. Unlike most other classifiers, GoogLeNet is not trained for one label, but rather is fed with 40 facial attributes at the same time and outputs multiple classification results accordingly. The facial attribute prediction pipeline is composed by labelled images which are first fed into the model. The features are extracted from the fully connected (FC) layer, and then 40 random forest classifiers are trained and facial attributes are subsequently obtained. This output represents the facial attribute distribution over the dataset fed int GoogLeNet and will be used in the attribute estimation of the new images fed into AnonymousNet. This can helpful to determine the attributes with lower distribution in order to enhance the De-Identification.

Afterwards, to obfuscate facial images while preserve visual reality, AnonymusNet adopts a GAN coupled with adversarial perturbation. The GAN is designed as two players, D and G, playing a minmax game with adversarial loss:

$$L_{adv} = E[\log(D(x))] + E[\log(1 - D(G(x)))]$$

where generator G is trained to fool discriminator D, who tries to distinguish real images from adversarial ones. As the face De-Identification task can be categorize as a image-to-image translation problem, the authors customize the GAN model based on StarGAN [17]. The adversarial perturbation consists of adding a Gaussian noise to an image which is generally considered as a simple and effective way to trick deep neural network-based face classifiers which have been showcased to be vulnerable against this attack.

[18] shows there exists a universal adversarial perturbation that can cause images misclassified with high probability by state-of-the-art deep neural networks.

The basic idea is formulated as follows: Suppose a distribution of images in $\mathbb{R}^d$, k is a face classifier that the output given an image x is $k(x)$, i.e. the image in the distribution that match the most.

The universal perturbation vector $v \in \mathbb{R}^d$ that can fool the face classifier $\hat{k}$ should satisfy

$$\| v \| \leq \xi$$

where ξ limits the size of the universal perturbation vector, and

$$P(k(x+v) \neq k(x) \geq (1-\delta))$$

where δ quantifies the failure rate of all the adversarial samples. AnonymusNet introduces a universal perturbation vector as identified through an iterative approach. For each iteration i, it applies DeepFool [18] to identify the minimal perturbation to let k misclassify each input, and update the universal perturbation corresponding to hyperparameter ξ_i to the total perturbation v. It is shown that the algorithm works on a small portion of images sampled from the training dataset, and the universal perturbation generalizes well with respect to the data and the network architectures.

2.3.2 Facial Attribute Transfer Model

The Facial Attribute Transfer Model (FATM) is a method that takes advantage of the recent advances in face attribute transfer models, while maintaining a high visual quality. This method was the starting point for all subsequent face swap tools. [19]

Instead of changing factors of the original faces or synthesizing faces completely, this method use a trained facial attribute transfer model to map non-identity related facial attributes to the face of donors, who are a small number (usually 2 to 3) of consented subjects. Using the donor faces ensure that the natural appearance of the synthesized faces is maintained, while ensuring the identity of the synthesized faces are changed. On the other hand, the FATM blends the donors facial attributes to those of the original faces to diversify the appearance of the synthesized faces.

The input is a RGB image or video frame containing the face of the target. Firstly, FATM runs a face detector and crop each detected face using the bounding boxes. Then, a facial landmark extraction algorithm is applied to the extracted face to locate landmark points corresponding to distinct facial structures such as the tips of eyes, eyebrows, nose, mouth and contour. These landmark points are then matched to the landmark points of a “standard” face, which has a fixed

size with a frontal orientation, with an affine transform. The affine transform is obtained by minimizing the distortion between the two sets of landmark points.

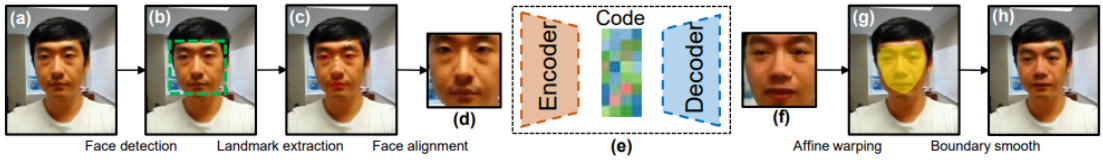


Figure 11: Overall pipeline of FATM De-Id method.

Using this affine transform, the method warps every pixel of the extracted face to the pose of the standard face, and resizes it to have dimension of 64*64 pixels. The rectified face is fed to the FATM, which synthesizes a face based on the donor's identity and the facial expression, head orientation, lighting condition, skin color and other facial characteristics of the target's face. The synthesized face image is resized to the original face, and warped back to the original configuration using the inverse of the same affine transform previously estimated.

After that, the synthesized face is trimmed with a face mask obtained from the landmark points to blend into the surrounding context. The face mask is created from the convex hull of landmarks of the eye brows and the bottom outline of mouth, and 8 interpolated points on both side of the faces to maximally cover the facial area. For instance, from the left side of the face, FATM chooses two extreme landmark points corresponding to the leftmost tip of the eyebrow and leftmost tip of the mouth. A similar procedure is repeated for the right side of the face. As the last step, FATM applies adaptive Gaussian smoothing of the boundary before finally splicing it into the original image to conceal the boundary of splicing. The whole process is automated and runs with minimum manual intervention. [20]

Chapter 3

Face swap tools

In this chapter we describe the general pipeline for what concern performing face swapping over a video file. This process is composed by 3 stages:

1. Data pre-processing: from raw data to a suitable input for the NN employed by the face swap tool.
2. Training stage: the NN is creating an algorithm that can be used to later reconstruct faces as closely as possible to the input images at facial features level and, in the same time, concealing the identity features of subject A.
3. Conversion stage: the algorithm created in the previous stage is used to convert a raw video into a swapped one, replacing the identity features of subject B over subject A.

In the following we detail these aspects, and present a set of tools we selected for comparisons. In Chapter 4 we provide the implementation details adopted to test and validate these models.

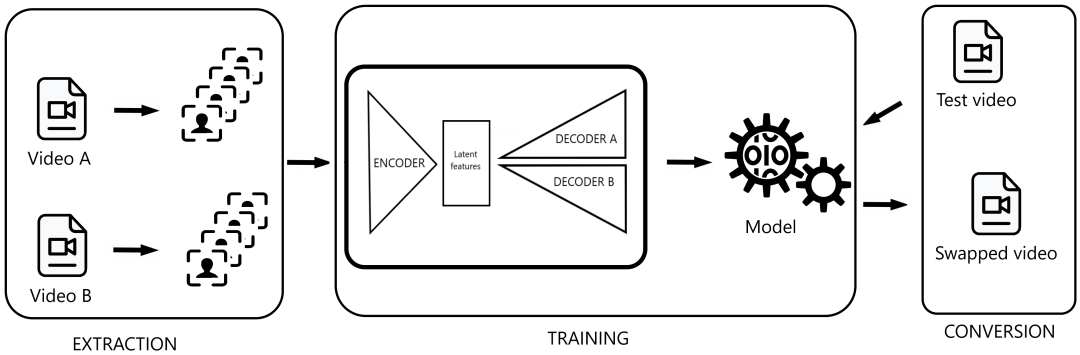


Figure 12: Overall pipeline of face swap process.

3.1 Data pre-processing

At high level, data pre-processing consists of three phases:

- Detection of a face within a frame of a given video.
- Landmark localization within the face and their alignment.
- Mask generation identifying which parts of the final face image correspond to the face and which are background/obstructions.

3.1.1 Detection and alignment

This phase is common in every face swap tool referred in this work. Given a video, a plugin is called to perform face detection on the frame that is under inspection, identify and extract the facial landmarks associated to that face and align the cropped image according the eyes' plane.

For these tasks, the plugins that established themselves are the Single Shot Scale-invariant Face Detector (S3FD) [21], the Multi-task Cascaded Convolutional Networks (MTCCN) [22] and the Face Alignment Network (FAN) [23].

The S3FD and MTCNN are anchor-based object detection methods that detect objects by classifying and regressing a series of pre-set anchors, which are generated by regularly tiling a collection of boxes with different scales and aspect ratios on the image. These anchors are associated with one or several convolutional layers, whose spatial size and stride size determine the position and interval of the anchors, respectively. The anchor-associated layers are convolved to classify and align the corresponding anchors.

The network employed by S3FD and MTCNN uses a cascade structure with three sub-networks; first the image is rescaled to a range of different sizes (called an image pyramid), then the first model (Proposal Network or P-Net) proposes candidate facial regions, the second model (Refine Network or R-Net) filters the bounding boxes, and the third model (Output Network or O-Net) proposes facial landmarks. Comparing with other methods, anchor-based detection methods are more robust in complicated scenes and their speed is invariant to object numbers. The differences among S3FD and MTCCNN regard the complexity level of the P-Net and O-Net: authors in [21] demonstrate that S3FD is more accurate with less false positives and slightly smoother, compared to MTCCN extraction, at the cost of speed of execution.

The FAN plugin is a Cascaded Shape Regression (CSR) based methods, where the face alignment begins with an initial estimate of the landmark locations which is then refined in an iterative manner. The initial shape S_0 is typically an average

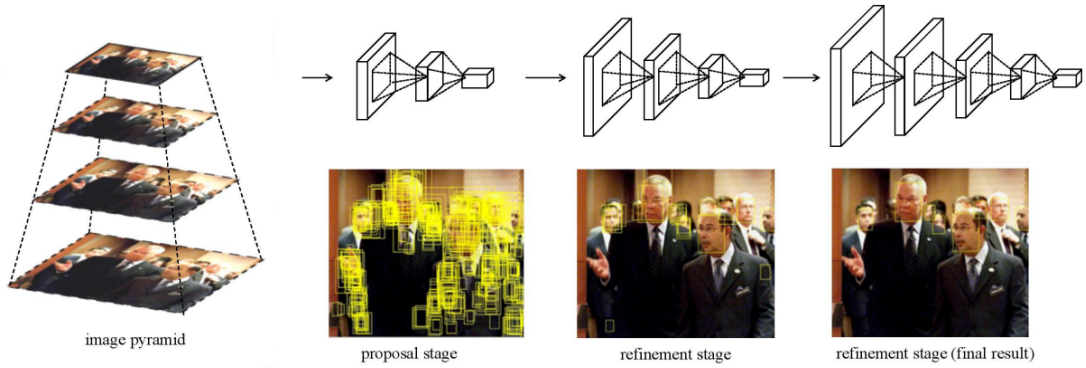


Figure 13: The main framework of CNN-based cascade face detector. It usually contains one proposal stage and several refinement stages. More specifically, the proposal stage usually is a FCN-based binary classifier, which generates a detection score map for each slice of the dense image pyramid. The refinement stages use larger CNNs to remove hard negative examples.

face shape placed in the bounding box returned by the face detector. Each CSR iteration is characterized by the following equation:

$$S_{t+1} = S_t + r_t(\phi(I, S_t))$$

where S_t is the estimate of landmark locations at iteration t, r_t is a regression function which returns the update to S_t given a feature ϕ extracted from image I at the landmark locations. In the case of FAN, it uses SIFT for feature extraction, while regression is performed using a simple linear regressor.

3.1.2 Masker

A masker is a plugin called after the detection and alignment of the given face. Its purpose is to focus the training phase on the face area, forcing the model to provide less importance to the background. This can help the model learn faster, whilst also ensuring that it is not consuming VRAM learning background details that are not important for the swapping. Moreover, the learned mask can be used in the conversion stage, giving the very useful option to preview the converted mask of the swapped subject, making some minor adjustments possible instead of train the model all over again.

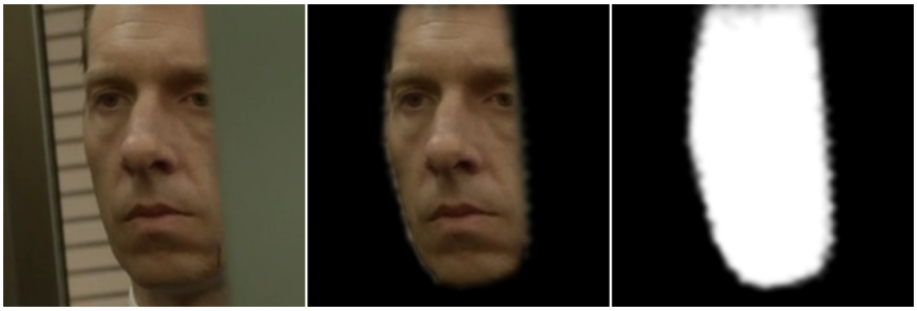
This phase is not common in every face swap tool used in this work, but a briefly description of the maskers used is provided below:

- Components: a mask generated from the 68 landmark point found in the

data pre-processing stage. This is a multi-part mask built around the outside points of the face.

- Extended: one of the main problems with the Components mask is that it covers the face only up to the top of the eyebrows. The Extended mask is based on the components mask, but attempts to go further up the forehead to avoid the "double eyebrow" issue, i.e. when both the source and destination eyebrows appear in the final swap.

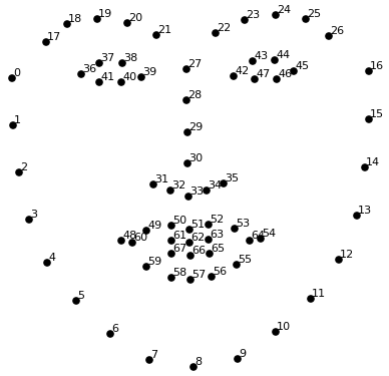
- VGG - Clear: a neural network mask designed to provide smart segmentation of mostly frontal faces clear of obstructions, trained using the VGGFace2 dataset which demonstrated state-of-the-art performance in recognising faces across pose and age. [24]



3.1.3 Output

The previous phases constitute the Extraction, i.e. the first stage meant to be performed in order to create the desired face swapped video. The possible outcomes of this stage are:

- An alignment file, i.e. a JSON/FSA file, and mask for converting the final frames. The alignments file holds information about all the faces found in each frame. Specifically where the face is, where the 68 landmark points are, and also it stores any mask that been extracted for each face.



- A set of faces, and optionally an alignments file and mask, for training the model.

Since this work deals with videos, the Extraction stage is linked with the corresponding frame rate counter of the video. Every framework has an option regarding the interval of frame considered for the task. If the Extraction is performed before the conversion, then the corresponding option will be set on 1, i.e. extract from every frame. If the purpose is training only, then the value has to be set depending on the number of frames per second the input is: for a 25fps video, sane values are between about 12 - 25, i.e. from every half second to a second. Anything less than that can led to too many similar faces, a condition that can adversely affect the training phase.

3.1.4 Sorting and cleaning

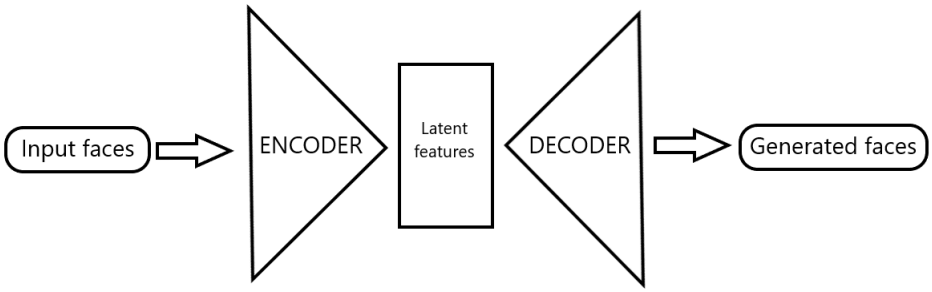
This is an optional phase, required if the input video contains multiple faces. Since the detector and aligner are not infallible, some false positives can occur. This can led to ghost artifacts into the alignment file, forcing the model to replace a face that is misaligned or doesn't exist at all. In order to avoid this, it is good practice to manually check the outcomes of the Extraction stage and, using a distance measure as the histogram-dis or face-cnn [25], sort them.

3.2 Training stage

This section is focused on the Training stage for the model used in a given face swap tool. Since the models used are different in some aspects, this section covers a base overview common to all the tools selected for this work.

3.2.1 Overview

At a high level, training is teaching our neural network (NN) how to recreate a face. Most of the models are largely made up of 2 parts:

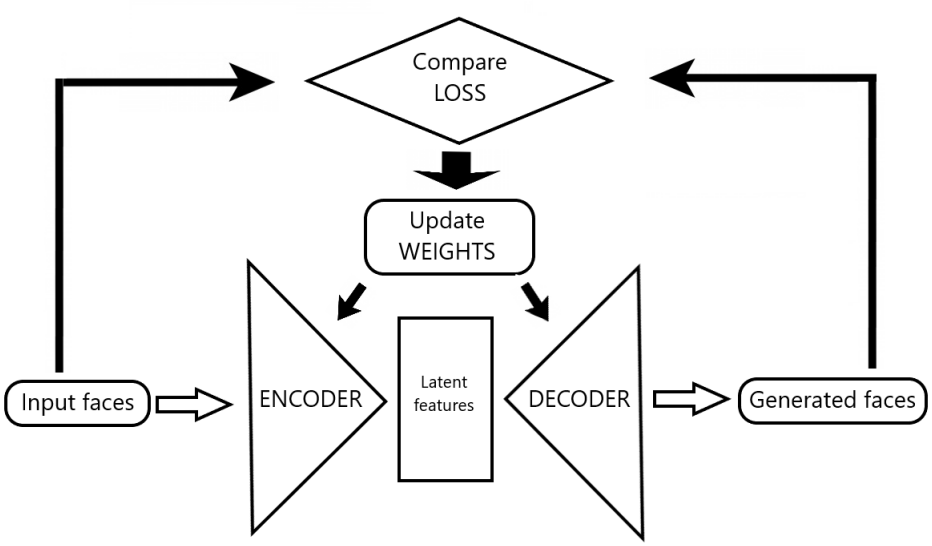


- Encoder: this is a network (FC, CNN, RNN, etc.) that takes the input, and outputs a feature map/vector/tensor. These feature vectors hold the information, the features, that represents the input. It is important to note that it is not learning an exact representation of every face you feed into it, rather, it is trying to create an algorithm that can be used to later reconstruct faces as closely as possible to the input images.

- Decoder: this is a network (usually the same network structure as encoder but in opposite orientation) that takes the feature vector from the encoder, and gives the best closest match to the actual input or intended output.

This structure is called autoencoder, a type of artificial neural network used to learn efficient data codings in an unsupervised manner. [26] [27] In order to evaluate how well the autoencoder is performing encoding and decoding faces, there are two main aspects to consider:

- Loss: for every batch of faces fed into the model, the NN will consider the face it has attempted to recreate by its current encoding and decoding algorithm and compare it to the actual face that was fed in. Based on how well it has done, it will give itself a score (the loss value) and will update its weights accordingly. The loss function is based on computing the delta between the



actual and reconstructed input. The optimizer will try to train both encoder and decoder to lower this reconstruction loss.

- Weights: once the model has evaluated how well it has recreated a face, it updates its weights. In order to do this, the NN uses the backpropagation, an algorithm composed by 2 phases:

  1. Forward phase: in this phase we initialize the weights of the neural network, propagate inputs forward through the network in order to generate the output values, calculate the error defined as

  $$\delta = \frac{\partial C}{\partial net}$$

  where C is the cost function and net is the weighted input in the certain neuron. Afterwards propagate the output back through the network in order to generate the error of all output and hidden neurons.

  2. Backward phase: the weight's output error and input are multiplied to find the gradient of the weight and a certain percentage, defined by learning rate of the weight's gradient is subtracted from the weight.

3.2.2 Face swap case

In our work, the overview given in the previous section is not reflected. Since the objective is to reconstruct a person face starting from someone else's face, the NN has to incorporate two major changes:

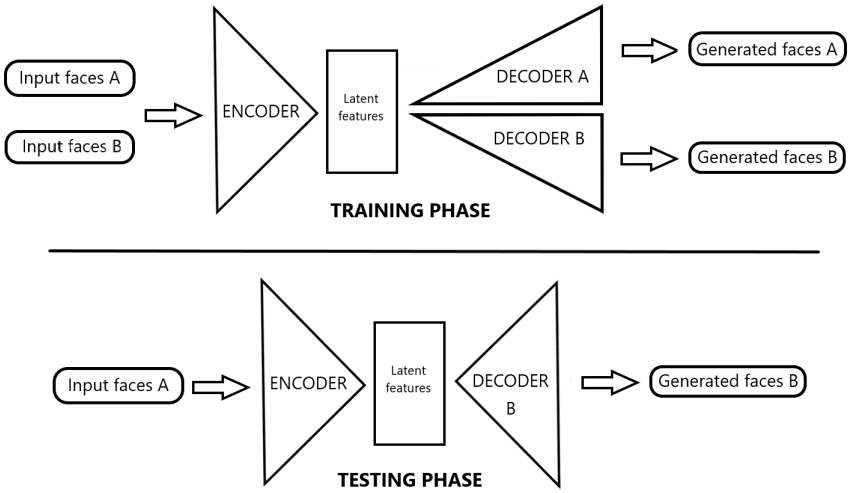


Figure 14: Overview of the network model used in this work. During the Training phase the encoders are shared. In the Testing phase, the decoders are switched.

- Shared Encoder: when we train the model, there are two sets of faces fed into it. The A set, i.e. the original faces that have to be replaced, and the B set, i.e. the faces that have to be placed in a scene. The first step is sharing the Encoder for both the A and B set. In this way the encoder is learning a single algorithm for 2 different people. This is extremely important, as in the final Converting stage the NN has to take the encodings of one face and decode it to another face. The encoder therefore needs to see, and learn, both sets of faces that are required for the swap.

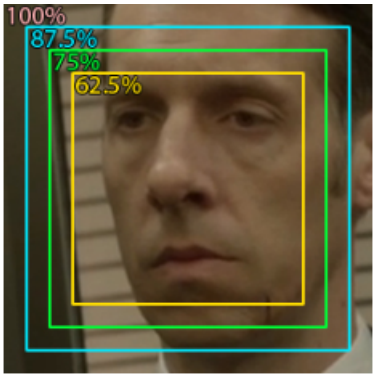
- Switched Decoders: when we train the model, are trained also two decoders. Decoder A is taking the encoding vectors and attempting to recreate Face A. Decoder B is taking the encoding vectors and attempting to recreate Face B. When the Converting stage comes, the decoders are switched around, therefore the model Face A passes through Decoder B. As the Encoder has been trained on both sets of faces, the model will encode the input Face of A, but then attempt to reconstruct it from Decoder B, resulting in a swapped face being output from our model.

3.2.3 Model selection and configuration

The frameworks used in this work can support several models. Since they can differ from each other in terms of complexity and configuration, the detailed explanation of the model chosen for each framework is provided in Chapter 4.

In this section we describe the options that are shared by all the models:

- Coverage: this is the amount of the source image that will be fed into the model. A percentage of the image is cropped from the center by the amount given. The higher the coverage percentage, the more of the face will be fed in.



  Whilst, intuitively, it may seem that higher coverage is always better, this is not actually the case, it is a trade-off. Even though a higher coverage will mean more of the face gets swapped, the input size of the model always remains the same, so the resulting swap will be less detailed, as more information needs to be packed into the same size image. To illustrate this, below is an extreme example of the same image with 62.5% coverage and 100% coverage, both sized to 32px. The 100% coverage image contains far less details than the 62.5% version.

- Masker: the options of the chosen masker (in Section 3.1.2 is provided a description of the available maskers) regards the mask blur which applies a slight blur to the edge of the mask in order to remove the hard edges of the mask and blends it more gradually from face to background, and the

Mask Threshold. The latter doesn't impact alignments based masks as the extended or components since they are binary, i.e. the mask is either "on" or "off". For NN based masks, the mask is not binary and has different levels of opacity. This can lead to the mask being blotchy in some instances. Raising the threshold makes parts of the mask that are near transparent, totally transparent, and parts of the mask that are near solid, totally solid. This can help with poorly calculated masks.

As discussed in the Section 3.2.2 the chosen model has weights which get updated at the end of each iteration. Initialization is the process by which these weights are first set. The default method for initialization is "he_uniform" [28], which draws samples from a truncated normal distribution centered on 0 with

$$StdDev = \sqrt{2/fanIN}$$

where fanIN is the number of input units in the weight tensor. However, there are other initialization methods:

- ICNR Init: this initializer is only applied to upscale layers. Standard initialization can lead to "checkerboard" artefacts in the output images when they get up-scaled within a NN. This initializer seeks to prevent these artefacts. [29]
- CA Init: Convolutional Aware Initialization is applied to all convolutional layers within the model. The premise behind this initializer is that it takes into account the purposes of the convolutional network and initializes weights accordingly, forming orthogonal filters not in the standard convolution space, but in the Fourier space. The reasoning is due to the convolutional theorem which states that convolution in the time domain is element wise multiplication in the frequency domain. This leads to higher accuracy, lower loss and faster convergence at the cost of VRAM consuming. [30]

Regarding the NN, the common configurations are:

- Reflect padding: every models have a notable "gray box" around the edge of the swap area in the final swap. This option changes the type of padding used in the convolutional layers to help mitigate this artefact.
- Loss function: the frameworks used in this work can adopt different loss function. The common ones are Mean Absolute Error (MAE), Binary Cross-Entropy (BCE), Structure Similairty Index Metric (SSIM) and Structural Dissimilarity (DSSIM). [31] [32]

0	0	0	0	0	0	0
0	0	1	2	3	0	0
0	0	4	5	6	0	0
0	0	0	0	0	0	0

6	5	4	5	6	5	4
3	2	1	2	3	2	1
6	5	4	5	6	5	4
3	2	1	2	3	2	1

Figure 15: The differences between using 0-Padding (left) and Reflect-Padding (right)

The MAE is a loss function used for regression. The loss is the mean over the absolute differences between true and predicted values, deviations in either direction from the true value are treated the same way. MAE is not sensitive towards outliers, given several examples with the same input feature values, the optimal prediction will be their median target value. When doing image reconstruction MAE encourages less blurry images compared to Mean Squared Error (MSE). The function can be defined as

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} |y_i - \hat{y}_i|$$

where N is the number of considered points, $\hat{y}$ is the predicted value and y is the true value.

The BCE is a loss function used on problems involving yes/no (binary) decisions. The model tries to decide whether the example belongs to the first or the second class and measures how far away from the true value (which is either 0 or 1) the prediction is. The function can be defined as

$$L(y, \hat{y}) = -(y * \log(\hat{y_i}) + (1 - y) * \log(1 - \hat{y_i}))$$

where $\hat{y}$ is the predicted value and y is the true value.

The SSIM is constructed as an image quality measure with respect to the human perception rather than absolute differences measured by metrics such as the MSE or MAE. Considering a pair of images (x, y) of sizes $m \times n$, SSIM measures three aspects of similarities according to human perception: luminance $l(x, y)$, contrast $c(x, y)$, and structure $s(x, y)$. Those are quantified according to the summary of relative measures including mean, variance, and co-variance measured under sliding windows of size $\delta \times \delta$ with step size of 1 on both horizontal and vertical directions. For each sliding window, SSIM measures base quantity for images x and y respectively. Then each perception subfunction is computed as

$$l(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}$$

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}$$

where (C_1, C_2, C_3) are constants less than 1 to balance potential zero division issue. Usually, $C_3 = \frac{1}{2}C_2$. To enforce independence among those measures, the final SSIM is constructed as the product of those metrics with exponential constant weights (α, β, γ) as

$$SSIM(x,y,\delta) = l(x,y)^\alpha * c(x,y)^\beta * s(x,y)^\gamma$$

and the output of SSIM is a decimal value between $(-1, 1)$ where $SSIM(x,y) = 1$ if and only if $x = y$. The DSSIM metric is derived from SSIM as

$$DSSIM(x,y,\delta) = \frac{1 - SSIM(x,y,\delta)}{2}$$

3.2.4 Data augmentation

Data augmentation consists in performing various manipulations on the input images, since a NN needs to processes as many different images as possible. In order to do this, there are several options:

- Zoom Amount: percentage amount that the face is zoomed in or out before being fed into the NN. Helps the model to deal with misalignments.
- Rotation Range: percentage amount that the face is rotated clockwise or anticlockwise before being fed into the NN. Helps the model to deal with misalignments.
- Shift Range: percentage amount that the face is shifted up/down, left/right before being fed into the NN. Helps the model to deal with misalignments.
- Flip Chance: chance of flipping the face horizontally. Helps create more angles for the NN to learn from.
- Color Lightness: percentage amount that the lightness of the input images are adjusted up and down. Helps to deal with different lighting conditions.
- Color AB: percentage amount that the color is adjusted on the A/B scale of the L*A*B color space. Helps the NN to deal with different color conditions.

- Color CLAHE Chance: percentage chance that the image will have Contrast Limited Adaptive Histogram Equalization applied to it. CLAHE is a contrast method that attempts to localize contrast changes. This helps the NN deal with differing contrast amounts. [33]

3.2.5 Monitoring Training stage

For face swapping tasks, the loss value is effectively meaningless. The value gives an idea of how well the NN thinks it is recreating Face A and how well it is recreating Face B. However, in this work, the NN is creating Face B from the encodings of Face A. It is impossible to get a loss value for this as there are no real-world examples of a swapped face for the NN to compare against. In order to evaluate how well the NN is performing face swap, there is a tool common to all frameworks used in this work, the Preview tool. This starts out as a solid color, or very blurry, but will improve over time as the NN learns how to recreate and swap faces.

Figure 16: Preview of the Trump-Cage swapping. The first column of each submatrix is referred to the source subject. The second column regards the reconstruction performed by the decoder of the original subject and the third column the output of the decoder swapping the destination subject over the original one.

Fine details like eye-glare and teeth will be the last things to come through. Once these are defined, it is normally a good indication that training is nearing completion. If the coverage option is set less than 100%, the Preview tool will display the edges of a red box. These indicate the swap area or the area that the NN has been trained on.

3.3 Conversion stage

The Conversion stage is the final part of the face swap process. This is also the simplest one and can be summarized in these steps:

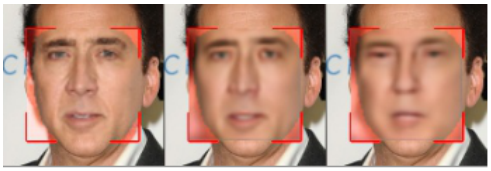


Figure 17: A Cage - Trump swap. The red corners indicate the coverage area.

1. Generate an alignments file for the target video, as described in Section 3.1.3, setting the interval of frame considered on 1, i.e. extract from every frame.
2. Cleanup the alignments file by deleting false positives and misaligned faces.
3. Run the script provided by the given face swap tool in order to create the swapped video using the model trained as described in Section 3.2.2.

3.4 Selected tools

In this section we focus on the pool of open-source face swap tools selected for producing an analytical comparison. We provide a general description below, while the specific configuration for each of them is in Chapter 4.

3.4.1 DFaker

DFaker [1] was the first major face swap tool that came out after FakeApp. This framework was developed by the user dfaker since January 2018 taking inspiration by the FakeApp model and trying to compensate the lack of an adequate pre-processing stage.

At the time that DFaker was presented, FakeApp proposed only the autoencoder-decoder pairing structure, leaving the extraction of the faces required for the training at the user's discretion. In order to address this, DFaker proposed an extraction plugin composed by the S3FD extractor coupled by the dlib-based aligner.

Moreover, the FakeApp's model was strictly defined, without any possible customization, while DFaker supports three different model:

- H64: 64x64 face resolution, which is used in the original FakeApp, but this model in DFaker uses TensorFlow 1.8 DSSIM Loss function, separated mask decoder and better components mask. This model is well suited for straight face-on scenes and can perform on low level hardware, with a minimum VRAM requirement set at 2 GB.

[1] DFaker Github link: `https://github.com/dfaker/df`

- H128: 128x128 face resolution, improved version of the model above which conserves better face details and will perform better with higher resolution videos and close-up shots. Also great for direct face-on scenes and gives the highest resolution and details.
- DF: 128x128 DFaker model, which works with a full-face model, covering a more "full" face which often expands and covers more areas of the cheeks. When using this model, it is recommended not to mix original faces with different lighting conditions. Great for side faces but provides lower resolution and detail. This model requires at least 4 GB pf VRAM.

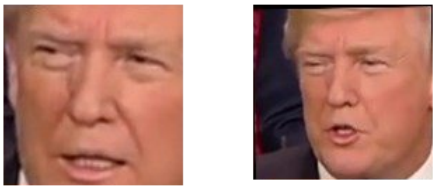


Figure 18: Differences between half-face model (left) and full-face model (right).

3.4.2 DeepFaceLap

DeepFaceLab (DFL) [2] has been in development since June 2018 on GitHub by the user iperov. This tool is equipped with a remarkable plug-in that allowed it to took the lead over the face swap tools of the time, the preview: since DFL has a configurable mask editor, this plug-in shows a windowed preview with the results of the edited mask training plus the conversion with the swap subject, permitting the user to modify the mask model at runtime.

Figure 19: Preview tool on DFL.

Furthermore, DFL was the first tool that perform data augmentation, automatic face extraction utilizing MTCNN and multi-gpu training stage. The latter feature was implemented in order to improve the performance during the training,

[2]DeepFaceLab Github link: `https://github.com/iperov/DeepFaceLab/`

allowing users that don't have an high-end GPU to train the new models that DFL come with:

- LIAEF128: 128x128 face resolution, produces a lower quality module in order to partially fix dissimilar face shape and behaves less aggressive while morphing facial features, though it has issues with tracking eye blinking. It requires at least 5 GB of VRAM.
- SAE & HDSAE: 128x128 and 256x256 face resolution. Styled AutoEncoder (SAE) and High Definition Styled AutoEncoder (SAEHD), new and improved versions of LIAEF model, much more heavy to run but provides noticeably better quality. Features of these models include new encoder that produces more stable and less flickering face image and new decoder that produces cleaner subpixel results, where pixel loss and DSSIM loss are merged together to achieve both training speed and pixel trueness. By default all the networks are initialized with CA weights. SAE requires 8 GB of VRAM, while HDSAE 10+ GB.

3.4.3 Faceswapp

FaceSwap's [3] initial commit was in December 2018 by GitHub user joshua-wu. It is based on a tool called OpenFaceSwap that is now defunct. The key advantage FaceSwap has over other tools is its GUI. The authors have translated all the python script files into a point-and-click interface where a user can select files and options easily. However, the underlying python scripts are easily accessible and is possible to run those individually. This gives the accessibility as well as customization and automation.

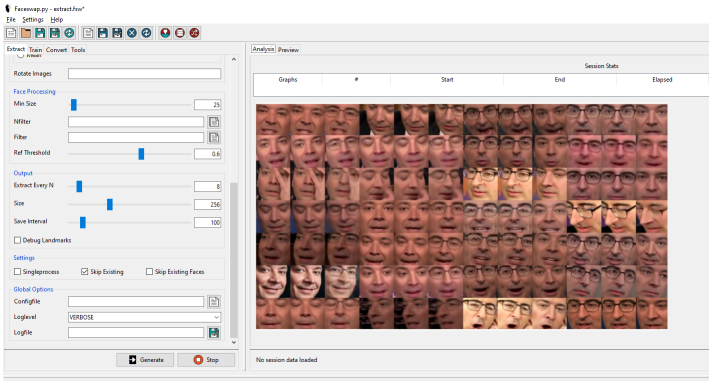


Figure 20: Faceswapp GUI.

[3] Faceswap Github link: `https://github.com/deepfakes/faceswap`

Faceswap includes all the models listed above coupled with new high definition models, making this tool the most complete in complexity terms of all the selected tools, spacing from low to very demanding modelling techniques:

- Unbalanced: 64x64 - 512x512 face resolution, model that can be customized from the input/output size to the complexity of the convolutional layers. It is worth noting that this model puts more emphasis on the B Decoder, so that reversing a swap, i.e. swapping B>A rather than A>B, will lead to less satisfactory results. This is why this model is called Unbalanced.

- Realface: 64x64 - 256x256 face resolution, the successor to the Unbalanced model. It takes learnings from that model and SAEHD, whilst looking to develop them further, including a better features detection such as tatoos, piercing and wrinkles. As with the Unbalanced model this model puts more emphasis on the B Decoder. It requires at least 11 GB of VRAM.

- Villain: 256x256 face resolution, it is likely the most detailed model but very VRAM intensive (12+ GB) and can give sub-par color matching when training on limited sources, requiring a much more intensive data pre-processing phase. It is the source of the viral Steve Buscemi/Jennifer Lawrence deepfake. This model does not have any customization options, beyond a low memory variant, i.e. saving VRAM at the cost of a lower batch-size.

Furthermore, FaceSwap has a very different personality than the other tools. The main developers torzdf, bryanlyon and kilroythethird have been active over the entire developing time, adding features and fixing bugs at a rapid pace. They also take effort to document their development and git commits making it fairly easy to follow the progress. The authors also give special attention to keeping deepfake usage ethical. Although they cannot stop anyone from using it in nefarious ways, they at least add a statement to the very beginning of the readme to show that they are thinking about how their tool might be used.

3.4.4 Faceswapp-GAN

Faceswap-GAN [4] (FSGAN) is the most recent of the selected tools. The development started in January 2019 by the user shaoanlu and it is heavily derived from Faceswap, even though it didn't inherit its interface.

Key feature of FSGAN is the addition of adversarial loss [34] in order to improve the reconstruction quality of the face image: the existing autoencoder network inherit from Faceswap acts as a generator in a GAN and the discriminator takes in

[4]Faceswap-GAN Github link: `https://github.com/shaoanlu/faceswap-GAN`

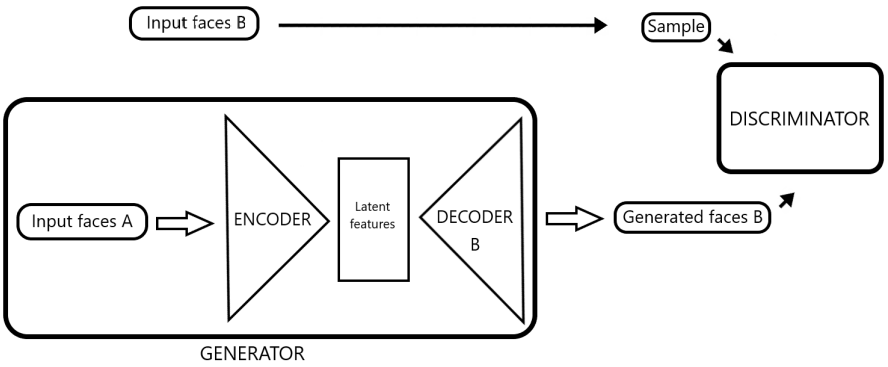


Figure 21: FSGAN architecture overview.

input the output of the autoencoder's decoder, taking inspiration by the structure found in CycleGAN. [35]

Moreover, FSGAN adopts the VGGFace perceptual loss which improves the direction of eyeballs to be more realistic and consistent with input face. It also smoothes out artifacts in the segmentation mask which helps on handling occlusion, eliminating artifacts, and producing natural skin tone, resulting in a higher output quality. The model doesn't allow any type of customization with the exception of the input/output size which spans from 64x64 to 256x256. FSGAN requires at least 6 GB of VRAM.

Figure 22: Effects of perceptual loss. From left to right: input image, mask trained without perceptual loss and mask trained with perceptual loss

Chapter 4

Experiments

This chapter focuses on the De-Identification task performed by the tools described in the previous chapter. This task is divided into two main aspects:

- De-Identification: given a video file, how well each tool conceals the original subject's identity after the face swap process has been performed.
- Facial attributes preservation: after the De-Id process, how well each tool can retain facial expression information from the original video.

The hardware and software setup, tools configuration and the description of the aforementioned aspects are provided in the following sections.

4.1 Experimental setup

4.1.1 Dataset

For our experiments, we use the Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) dataset. RAVDESS [36] is a collection of 24 professional actors (12 female, 12 male), vocalizing two lexically-matched statements in a neutral North American accent.

Figure 23: Frame samples of (from left to right) Actor 1, Actor 2, Actor 3 and Actor 4.

Speech includes calm, happy, sad, angry, fearful, surprise, and disgust expressions, and song (i.e. the previous speech performed in a musical way) contains calm, happy, sad, angry, and fearful emotions. Each expression is produced at two levels of emotional intensity (normal, strong), with an additional neutral expression. All conditions are available in three modality formats: audio-only (16bit, 48kHz .wav), audio-video (720p H.264, AAC 48kHz, .mp4), and video-only (no sound). Since the audio is irrelevant for our purposes, we use video-only file which are organized in

- 208 video files lasting approximately 5 seconds for each actor containing all the possible combinations for the two different statements.
- 1 video file lasting approximately 4 minutes for each actor containing a clip for each expression described above.

Using FFMpeg [1], an open-source multimedia framework which manages audio/video file, we concatenate all the 208 video files into a single one, obtaining approximately 14 minutes video. This video will be used in the training stage, while the single sample video will act as test video for the final swap.

On the other hand, the training set regarding the swap subjects employed for this work consists of a man and a woman with whom we have registered approximately 7 minutes video each, with variations in pose and illumination.

Figure 24: Frame samples of the swap subjects training set. In the first row Giulia, in the second Giorgio.

[1] FFMpeg Github link: `https://github.com/FFmpeg/FFmpeg`

4.1.2 Hardware setup

The extraction and conversion stage are performed on a laptop PC with Intel i7 7700K, 16 GB of RAM and nVidia GTX 1060 GPU. The training stage is performed on a desktop PC with Intel Xeon E3-1200, 64 GB of RAM and nVidia Quadro P6000 GPU.

4.2 Tools configuration

In this section we provide the NN architecture employed by each face swap tool, along with specific options for the training and conversion stage if the tool requires it.

4.2.1 DFaker

DFaker is the simplest of the selected tools. It doesn't allow any type of customization except for the model. For this work we have chosen the DF model, the most complex of the available models. It consists of an encoder network which contains three blocks and accepts face images in a shape of 128 x 128 x 3 as input, and a decoder network which also contains three blocks and its output is the same as the input image, 128 x 128 x 3. The latent vector is 512-dimensional.

We train DFaker model with Adam optimizer [37] with exponential decay rate for the first moment estimates $\beta 1 = 0.5$, exponential decay rate for the second moment estimates $\beta 2 = 0.999$, starting learning rate α=0.0005, with batch size of 64 for 12 hours. At the end of the training stage DFaker registered 347000 iterations.

4.2.2 DeepFaceLab

DFL supports several customizations regarding the chosen model, HDSAE. This model accepts and outputs face images in a shape of 256x256x3, it is initialized with Convolutional Aware weights [30] and the latent vector is 2048-dimensional. It is enabled the option "AE_mem" which allows the NN to use the CPU and consume the available RAM if the VRAM is fully saturated, without stopping the training process, leading to a bigger batch size. DFL adopts the mask components. The key feature of the HDSAE model is the presence of symmetric skip connections [38], which help to back-propagate the gradients to bottom layers and pass image details to the top layers, making training of the end-to-end mapping more easier and effective, and thus achieve performance improvement while the network going deeper.

We train DeepFaceLab model with Adam optimizer [37] with exponential decay rate for the first moment estimates $\beta 1 = 0.9$, exponential decay rate for the second moment estimates $\beta 2 = 0.999$, starting learning rate α=0.001, with batch size of 32 for 12 hours. At the end of the training stage DeepFaceLab registered 256500 iterations.

4.2.3 Faceswap

Faceswap is the most complete of the selected tools. It offers several models, but the Unbalanced and the Villain are the best ones. With an input/output size of 512x512 pixels, we have chosen the Villain model due to its color matching and the extended masker which prevents the "double eyebrow" issue. It has the highest requirement in VRAM terms, and the batch size was set on 8: since this batch size would led to significant amount of time added on the training stage, we have enabled the option "Memory Saving Gradients" [2] (reserved for nVidia GPUs) which can halves the VRAM requirements at a 20% increase in training time. As DeepfaceLab, also Faceswap adopts skip connections between convolutional and de-convolutional layers.

We train Faceswap model with Adam optimizer [37] with exponential decay rate for the first moment estimates $\beta 1 = 0.9$, exponential decay rate for the second moment estimates $\beta 2 = 0.999$, starting learning rate α=0.0001 with batch size of 8 for 12 hours. The hyperparameter λ, i.e. the regularization rate, is gradually increased from $1 * 10^{-7}$ to $2 * 10^{-6}$, in four steps. At the end of the training stage Faceswap registered 165000 iterations. Without this gradual increase, the naturalness of the generated face is diminished.

4.2.4 Faceswap-GAN

FSGAN, as DFaker, doens't consent any customization except the size of input/output, which is set on 256x256, and the dimension of the latent vector, which is set on 2048, the highest possible. As described in Section 3.4.4, FSGAN uses the adversarial loss in order to improve the reconstruction quality of the face image and adopts the VGG - Clear masker in order to benefit of the perceptual loss.

We train Faceswap-GAN model with Adam optimizer [37] with exponential decay rate for the first moment estimates $\beta 1 = 0.5$, exponential decay rate for the second moment estimates $\beta 2 = 0.999$, starting learning rate α=0.00005, with

[2]Memory Saving Gradients Github link: `https://github.com/cybertronai/gradient-checkpointing`

batch size of 16 for 12 hours. At the end of the training stage Faceswap-GAN registered 214500 iterations.

4.3 De-Identification process

In this section we provide a description regarding the De-Id process, how it is performed and its evaluation.

4.3.1 De-Identification method

In the De-Id method, three major phases can be distinguished:

1. Select a specific face swap framework: given the pool of four face swap tools, one of them is selected and is provided in input a single video file from the dataset RADVESS. This video file represents the original point from which a ground truth is defined, while the video file obtained in output from the selected face swap tool is compared against it.

2. Run extract_frames.py: this script has the task of taking in input a video file and randomly extracts frames from it. This step requires the creation of two directories, Known and Unknown: the former will contain one face image of the actor belonging to RADVESS and one face image of the subject used for the swapped video file, the latter will contain all the extracted frames from the original video file and the swapped one.

3. Run face_recognition.py: this script has the task to take in input both the directories previously created and comparing each face image in the Unknown folder against all the face images in the Known folder, outputting a distance measure for each couple of images.

Further information regarding the tool used for the evaluation and its distance measure are provided in the next sub-section.

4.3.2 De-Identification evaluation

In this work, the evaluation regarding the De-Id performed on a video file is performed by face_recognition [3]. This is an API written for Python and command line interface which offers tools for several tasks such as face detection, finding and

[3]face_recognition Github link: `https://github.com/ageitgey/face_recognition`

manipulating facial features in pictures and face recognition. The face recognition tool is based on dlib's face recognition built with deep learning, achieving an accuracy of 99.38% on the Labeled Faces in the Wild benchmark. [39]

The API operates in this way:

1. Encode the input image using the Histogram of Oriented Gradients algorithm (HOG) to create a simplified version of the image. Using this simplified image, the API scans the part of the image that most looks like a generic HOG encoding of a face.

2. Determine the pose of the face image by finding the main landmarks in the face, then warp them to the image so that the eyes and mouth are centered.

3. Deploying a pre-trained CNN based on the FaceNet framework, each face image is embedded in a 128-dimensional code vector.

4. Each code vector is compared against the previous ones in order to find the closest match to a given code vector using the euclidean distance measure.

Since in this work points 1) and 2) are performed by the extraction stage, the focus is on points 3) and 4). As described in Section 4.3.1, once the Known and Unknown directories are made, the tool will provide a distance measure based on the Euclidean distance, defined as

$$D(x, y) = \sqrt{\sum_{i=0}^{128}(x_i - y_i)^2}$$

where x and y are the 128-dimensional code vectors. The lower the measure is, the closer the subjects are, maintaining a threshold set on 0.6 which discriminates between matching/non matching subjects. In order to perform a stricter comparison, lower this threshold would reduce false positive matches at the risk of more false negatives.

In this work are considered different evaluations regard the De-Id process. The next list summarizes them, referring $V_{original}$ and $ID_{original}$ as the original video selected from the RADVESS dataset and the face image of the belonging actor, $V_{swapped}$ and $ID_{swapped}$ as the swapped video and the face image belonging to the swapped actor. C refers to the face recognition tool.

- $C(V_{original}, ID_{original})$: this is a comparison performed in order to confirm the API's state-of-the-art performance. This measurement is expected to be very low, indicating a strong match between the corresponding code vectors.

Figure 25: Comparison between original video and original ID.

- $C(V_{original}, ID_{swapped})$: in this case the comparison is between frames of the original video and the swapped identity face image of the RADVESS' actor. This measurement is expected to be high.

Figure 26: Comparison between original video and swapped ID.

- $C(V_{swapped}, ID_{original})$: this is the core of this work, in fact in this comparison is tested the quality of the swap against the ground truth of the RADVESS' actor face image.

Figure 27: Comparison between swapped video and original ID.

- $C(V_{swapped}, ID_{swapped})$: this comparison acts as the first adding the quality

measure of the swap, useful in order to verify the relationship between the original and swapped data.

Figure 28: Comparison between swapped video and swapped ID.

4.4 Facial attributes preservation process

In this section we cover the other aspect of the trade-off described in Chapter 1: to maintain as much structural information as possible after the face swap process.

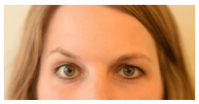
In order to verify this, firstly there is the need of coding the facial attributes of the original and swapped subjects and then comparing them against each other and this task is well suited for the Facial Action Coding System (FACS). [40]

FACS is a system to taxonomize human facial movements by their appearance on the face, based on a system originally developed by a Swedish anatomist named Carl-Herman Hjortsjö. It was later adopted by Paul Ekman and Wallace V. Friesen, and published in 1978. Ekman, Friesen, and Joseph C. Hager published a significant update to FACS in 2002. Movements of individual facial muscles are encoded by FACS, which is capable to identify instant changes in facial appearance. It is a common standard to systematically categorize the physical expression of emotions, and it has proven useful to psychologists and to animators. Due to the subjectivity and time consumption issues, FACS has been established as a computed automated system that detects faces in videos, extracts the geometrical features of the faces, and then produces temporal profiles of each facial movement. These movements are coded into Action Units (AU), i.e. the fundamental actions of individual muscles or groups of muscles.

4.4.1 Facial attributes preservation method

The facial attributes preservation method adopts OpenFace [4], a tool intended for computer vision and machine learning researchers, the affective computing community and people interested in building interactive applications based on facial behavior analysis. OpenFace is the first toolkit capable of facial landmark detection, head pose estimation, facial action unit recognition, and eye-gaze estimation with available source code for both running and training the models. The computer vision algorithms which represent the core of OpenFace demonstrate state-of-the-art results in all of the above mentioned tasks. Furthermore, this tool is capable of real-time performance and is able to run from a simple webcam without any specialist hardware.

Using FACS it is possible to code nearly any anatomically possible facial expression, deconstructing it into the specific AU that produced the expression. It is a common standard to objectively describe facial expressions. OpenFace is able to recognize a subset of AUs, specifically: 1, 2, 4, 5, 6, 7, 9, 10, 12, 14, 15, 17, 20, 23, 25, 26, 28, and 45. A description of each AU along with a reference image is provided below:

AU number	FACS name	Example
1	Inner brow raiser	
2	Outer brow raiser	
4	Brow lowerer	
5	Upper lid raiser	
6	Cheek raiser	
7	Lid tightener	

[4]OpenFace Github link: `https://github.com/TadasBaltrusaitis/OpenFace`

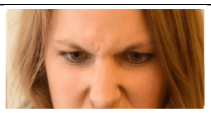
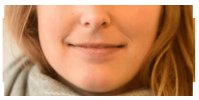
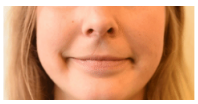
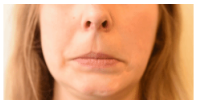
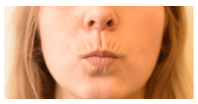
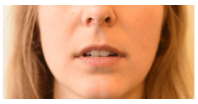
AU number	FACS name	Example
9	Nose wrinkler	
10	Upper lip raiser	
12	Lip corner puller	
14	Dimpler	
15	Lip corner depressor	
17	Chin raiser	
20	Lip stretcher	
23	Lip tightener	
25	Lips part	
28	Lip suck	
45	Blink	

Table 1: Description of each AU considered in this work.

As for the De-Id, this method is composed of three steps:

1. Select a specific face swap framework: given the pool of four face swap tools, one of them is selected and is provided in input a single video file from the RADVESS dataset. This video file represents the original point from which

a ground truth is defined, while the video file obtained in output from the selected face swap tool is compared against it.

2. Run OpenFace: the tool has the task of taking in input a video file and extracting a set of 17 AUs from each frame of the video file, outputting a CSV file where for each frame and for each AU is reported the respective intensity whitin a range from 0 to 5.

3. Run face_att.py: this script has the task of taking in input the CSV files belonging to the original video file and the swapped one, comparing them using two specific measures and outputting the result.

Further information regarding the measures used for the evaluation and the manage of the AUs set are provided in the next sub-section.

4.4.2 Facial attributes preservation evaluation

The evaluation regarding the facial attributes preservation consists in the deploying of the **Pearson Correlation Coefficient** (PCC) and the **Root-Mean-Square Error** (RMSE).

Pearson's correlation coefficient is the statistic test that measures the statistical relationship, or association, between two variables. It is known as the best method of measuring the association between variables of interest because it is based on the method of covariance. It gives information about the magnitude of the association, or correlation, as well as the direction of the relationship. Coefficient values can range from +1 to -1, where +1 indicates a perfect positive relationship, -1 indicates a perfect negative relationship, and a 0 indicates no relationship exists. The PCC is defined as

$$PCC(X,Y) = \frac{\sum_{i=0}^{N}(X - X_m) * (Y - Y_m)}{\sqrt{\sum_{i=0}^{N}(X_i - X_m)^2 * \sum_{i=0}^{N}(Y_i - Y_m)^2}}$$

where x_m is the mean of the vector x and y_m is the mean of the vector y.

The **RMSE** is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are, i.e. how concentrated the data are around the line of best fit. This measure is commonly used in climatology, forecasting, and regression analysis to verify experimental results. When standardized observations and forecasts are used as RMSE inputs, there is a direct relationship with the PCC. For example, if the correlation coefficient is 1, the RMSE will be 0, because all of the points lie on the regression line (and therefore there are no errors). The RSME is defined as

$$RMSE(X,Y) = \sqrt{\frac{\sum_{i=0}^{N}(X_i - Y_i)^2}{n}}$$

where X are the observed-values vector and Y are the predicted-value vector.

4.5 Results

In this section we analyze the results obtained by the De-Id and facial attributes preservation processes. The former's results are presented for each of the selected tools. The latter's results, due to the size of the AUs considered, are listed in a cumulative table.

These tests are performed with same sex pairs. In order to provide further evaluations, we selected 10 actors from the RADVESS dataset (5 male and 5 female) and conducted the De-Id and facial attribute preservation processes with fixed RADVESS' actor and variable swap subject, identifying whether there are sex-related dependencies between them.

In Appendix A are listed the De-Id tables for each framework and in Appendix B the corresponding graphs for each AUs, for each framework.

4.5.1 De-Identification

In Fig. 29, 30, 31 and 32 the box-plots represent the Euclidean distances of the tests conducted as described in Section 4.3.2: the first distribution refers to the case $C(V_{original}, ID_{original})$, the second $C(V_{original}, ID_{swapped})$, the third $C(V_{swapped}, ID_{original})$ and the fourth $C(V_{swapped}, ID_{swapped})$. The script extract_frames.py is set to extract 1000 frames.

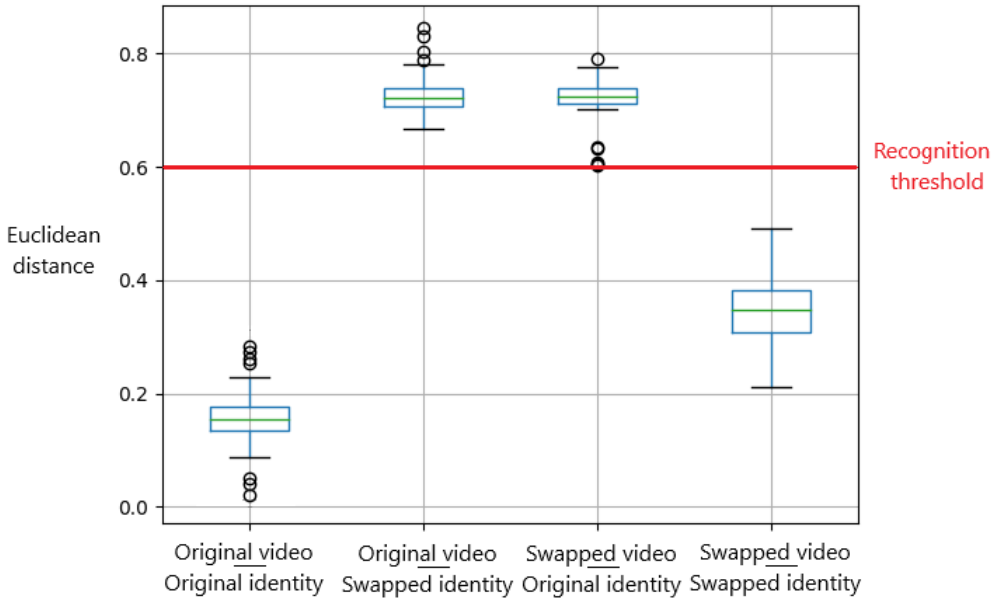


Figure 29: Results of the De-Id process by **DFaker**. The box-plot represents the comparison between (from left to right): $C(V_{original}, ID_{original})$, $C(V_{original}, ID_{swapped})$, $C(V_{swapped}, ID_{original})$, $C(V_{swapped}, ID_{swapped})$. Below the recognition threshold, two frames are classified as matching. The circles refer to outliers.

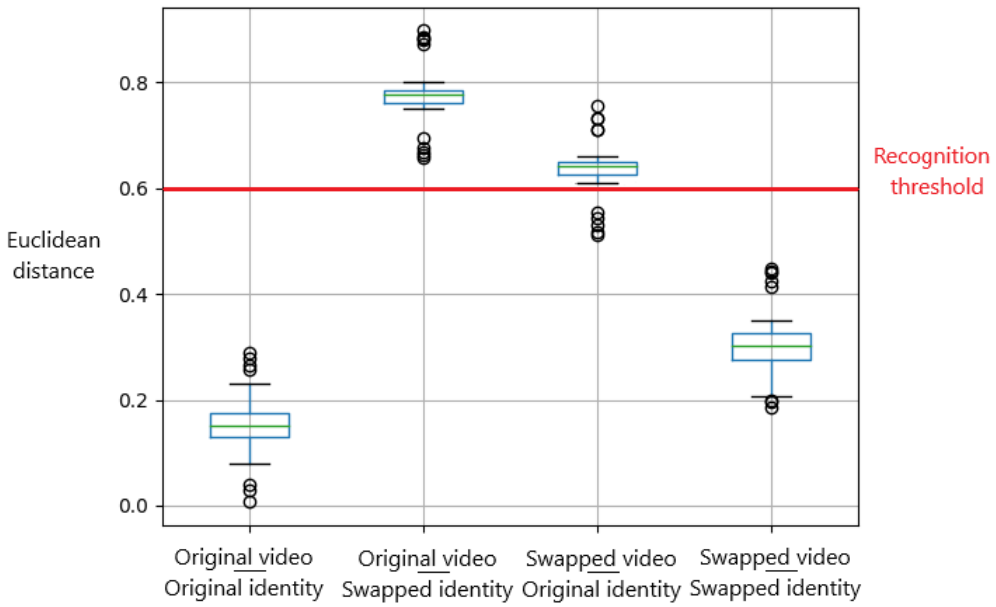


Figure 30: Results of the De-Id process by **DeepFaceLab**. The box-plot represents the comparison between (from left to right): $C(V_{original}, ID_{original})$, $C(V_{original}, ID_{swapped})$, $C(V_{swapped}, ID_{original})$, $C(V_{swapped}, ID_{swapped})$. Below the recognition threshold, two frames are classified as matching. The circles refer to outliers.

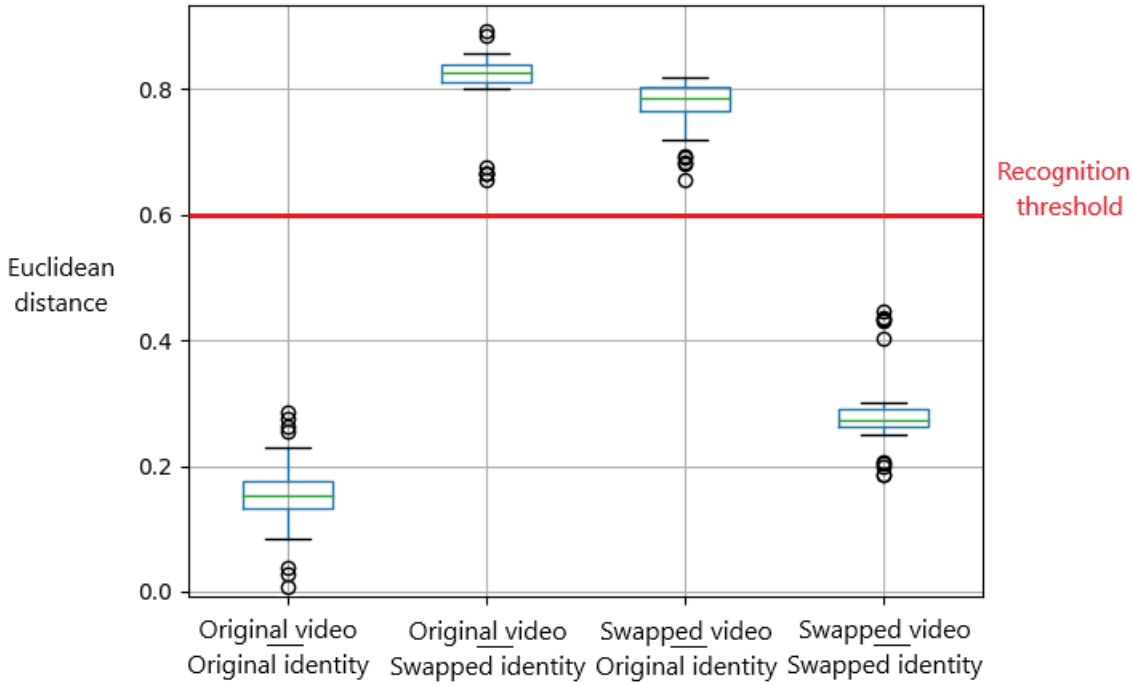


Figure 31: Results of the De-Id process by **Faceswap**. The box-plot represents the comparison between (from left to right): $C(V_{original}, ID_{original})$, $C(V_{original}, ID_{swapped})$, $C(V_{swapped}, ID_{original})$, $C(V_{swapped}, ID_{swapped})$. Below the recognition threshold, two frames are classified as matching. The circles refer to outliers.

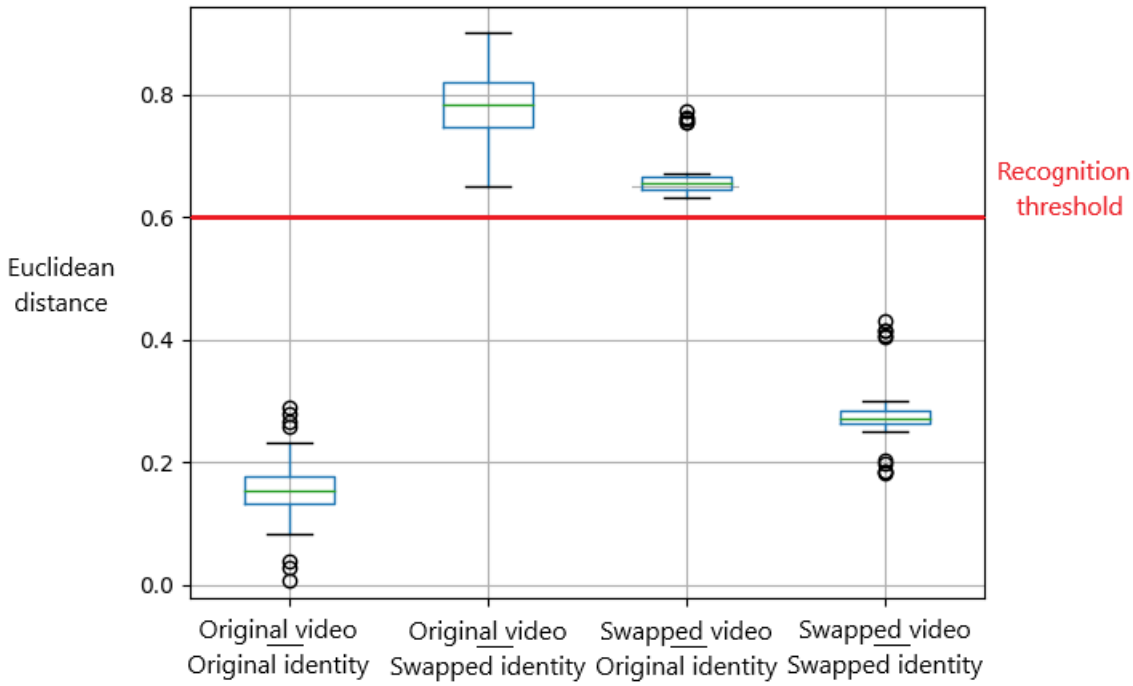


Figure 32: Results of the De-Id process by **Faceswap-GAN**. The box-plot represents the comparison between (from left to right): $C(V_{original}, ID_{original})$, $C(V_{original}, ID_{swapped})$, $C(V_{swapped}, ID_{original})$, $C(V_{swapped}, ID_{swapped})$. Below the recognition threshold, two frames are classified as matching. The circles refer to outliers.

4.5.2 Attributes preservation

The first table below lists the results obtained from the attributes preservation process for each framework. As described in Section 4.4.1, for this work a subset of the AUs is considered. Each framework column is composed of 2 sub-columns: on the left is reported the PCC value, on the right the RMSE value. These values are obtained from the comparison between the AUs recorded from the original video of each actor and the swapped video from each framework.

The second table below lists the average PCC and RMSE value along with their respective standard deviations calculated considering the whole dataset.

Actors	DFaker		DeepFaceLab		Faceswap		Faceswap-GAN	
	PCC	RMSE	PCC	RMSE	PCC	RMSE	PCC	RMSE
AU 1	0.476	0.803	0.432	0.592	**0.78**	**0.385**	0.674	0.466
AU 2	0.241	0.885	0.589	0.517	**0.849**	**0.329**	0.79	0.39
AU 4	0.289	0.999	0.594	0.876	**0.858**	**0.385**	0.804	0.835
AU 5	0.325	0.433	0.6	0.567	**0.89**	**0.332**	0.83	0.409
AU 6	0.276	0.684	0.491	0.84	0.791	0.219	**0.806**	**0.232**
AU 7	0.223	0.962	0.532	0.863	**0.826**	**0.335**	0.752	0.613
AU 9	0.27	0.45	0.469	0.408	**0.778**	**0.283**	0.691	0.329
AU 10	0.308	0.812	0.476	0.996	**0.806**	**0.384**	0.721	0.523
AU 12	0.457	0.897	0.559	0.865	**0.819**	**0.261**	0.753	0.222
AU 14	0.301	0.875	0.419	0.715	0.658	0.41	**0.757**	**0.208**
AU 15	0.125	0.543	0.295	0.442	0.531	0.338	**0.64**	**0.281**
AU 17	0.312	0.874	0.452	0.798	**0.778**	**0.306**	0.689	0.609
AU 20	0.594	0.328	0.324	0.373	**0.651**	**0.289**	0.535	0.327
AU 23	0.659	0.479	0.28	0.298	**0.574**	**0.248**	0.463	0.272
AU 25	0.312	0.839	0.569	6.305	**0.843**	**0.156**	0.776	0.934
AU 28	0.213	0.783	0.449	0.775	**0.778**	**0.393**	0.688	0.593
AU 45	0.154	0.928	0.598	0.91	**0.865**	**0.45**	0.81	0.464

Table 2: Results for each framework of the attributes preservation process.

Framework	PCC		RMSE	
	μ	σ	μ	σ
DFaker	0.287	0.163	0.781	0.252
DeepFaceLab	0.578	0.102	0.587	0.332
Faceswap	**0.782**	**0.083**	**0.196**	**0.172**
Faceswap-GAN	0.699	0.101	0.211	0.221

Table 3: Mean PCC and RMSE values along with their respective standard deviations for each framework.

4.6 Results discussion

In this section we focus on the results obtained by each framework. For what concerns De-Id, DFaker has scored remarkable results, achieving the second place with a mean Euclidean distance of ($\mu \pm \sigma$) 0.768 $\pm$0.147, exceeded only by Faceswap. This performance is impressive considering the substantial differences between the model adopted by the two tools: DFaker uses a 3 blocks autoencoder structure without skip connections and a 512-dimensional latent vector, compared to the 6

blocks autoencoder structure with residual blocks and a 2048-dimensional latent vector used by Faceswap. DFaker's worst results involved actors with narrow faces and/or hair obstructions: actor 3, 11, 17, 20, 24 present these traits and in the extraction stage, the MTCNN extractor fails to locate the exact facial landmarks in the frame that present the side of the face. The model, instead of replacing the swapped identity over the actor, chooses to leave the original identity if the detected landmarks don't respect a pre-fixed aspect ratio between them and therefore lowering considerably the face distance. These actors are represented by the lower-outliers in Figure 28 under the case $C(V_{swapped}, ID_{original})$.

DeepFaceLab (DFL), on the other hand, obtained the worst result in the De-Id process, achieving a mean Euclidean distance of $(\mu \pm \sigma)$ 0.631 $\pm$0.118. DFL adopts the component mask but the coverage area with this option is not settable. The masked area is roughly 55% less wide than with the other tools and this can lead to better attribute preservation at the cost of De-Id: during the conversion stage, the model covers mainly the eyes and the nose, leaving in some cases the mouth-area, eyebrows and cheeks as the reconstructed original ones (Encoder A -> Decoder A) if the DSSIM loss detects discrepancies between the original actor and the swapped identity. This behavior can be observed for the lower-outiliers in Figure 29 which correspond to actors 3, 9, 13, 16, 18 and 21: these actors have a noticeably wider face compared to the subject employed for the swap, forcing DFL to focus the training on the aforementioned features instead of covering the entire face, leading to De-Id issues since several zones act as a slightly blurred-version of the original ones.

Faceswap was the best tool in the De-Id, achieving the first place with a mean Euclidean distance of $(\mu \pm \sigma)$ 0.796 $\pm$0.197. The Villain model is the heaviest in VRAM terms of the tools selected for this work, forcing to lower the batch size up to 8. With this batch size, which is less than half of the one used for the other tools, the model during the training stage detects particular aspects and details of each single face rather than learning generic features and only afterwards refines them. This can lead to blurred areas along the edges of the face shape and around the nose, but these flaws are mitigated due to the output resolution of the model being 512x512. Actor 7, 8 and 14 were the only actors which present some uncertainties during the conversion stage, but this is attributable to the similarity between the actors and the swapped subjects.

Faceswap-GAN (FSGAN) is the most consistant of the tools, confirmed by a mean Euclidean distance of $(\mu \pm \sigma)$ 0.682 $\pm$0.038. The autoencoder model is derived from the one used by DeepFaceLab but in FSGAN is coupled with the adversarial loss, as described in Section 3.4.4. This loss along with the perceptual loss introduced by the VGG-Clear segmentation mask, can compensate the difficulties encountered by DFL while processing wide faces and hard/inclined face

expressions: the masker can isolate single face parts from both subject (forehead, eyes, nose, mouth and cheeks) allowing the model to improve poorly reconstructed areas instead of replacing them with the original ones as DFL does.

For what concerns attribute preservation, it's safe to state that DFaker acts as a naïve method, described in Section 2.1: the input/output resolution along with the latent vector can't manage properly the FHD frames extracted from the original video and this unsuitability reflects the quality of the swapped video, leading to heavily blurred areas and, in case of inclined/side faces, artifacts. This causes on one side the flattening of the AUs as in the case of AU 4, 5, 9, 12 and 15, on the other the rising as in the case of AU 1, 2 and 20, where the registered signal peaks don't correspond to the original video. The mean PCC value of $(\mu \pm \sigma)$ 0.287 $\pm$0.163 and the mean RMSE value of $(\mu \pm \sigma)$ 0.781 $\pm$0.252 confirm this sort of input utility destruction, placing DFaker in the last place. Paradoxically, the best results in this process are the ones obtained when DFaker fails the swap and leaves the original actor's face, allowing OpenFace to detect the true intensity of AUs.

DFL with a mean PCC value of $(\mu \pm \sigma)$ 0.578 $\pm$0.102 and mean RMSE value of $(\mu \pm \sigma)$ 0.587 $\pm$0.332 achieved the middle place in the attribute preservation process, showing the initial purpose of the autoencoder structure: the symmetric skip connections [38], which help to back-propagate the gradients to bottom layers and pass image details to the top layers, along with pixel and DSSIM loss, mitigate the lack of an extended masker, ensuring the reconstruction quality of eye and mouth related AUs. The worst results include the brows related AUs, since the components mask doesn't cover the entire face and actors with sunken or very spaced eyes can lead to small artifacts in the forehead zone. A particular case is actor 2 which presents a prominent face mole under the left eye: the model under the presence of a hard face expression, tries to incorporate it with the eye, causing heavy interference during the AUs detection.

Faceswap reconfirms the first place also in this process, winning with a mean PCC value of $(\mu \pm \sigma)$ 0.782 $\pm$0.083 and a mean RMSE value of $(\mu \pm \sigma)$ 0.396 $\pm$0.172. This superior performance compared to the other tools is achieved through two main aspects: the extended masker covers from the chin up to the forehead section, effectively preventing the "double eyebrow" issue, and the input/output resolution which allows the model to reconstruct in most cases wrinkles around eyes, mouth and cheekbones. With these techniques, even though Faceswap counted the lowest number of iterations with 165000, the model ensures among the best results in terms of attribute preservation, with some uncertainties only for actor 9, 11 and 16 where the beard of the male subject employed for the swap and the narrow face shape of the female subject lead to minor flickers in the mouth zone.

FSGAN performed better in the attribute preservation than in De-Id, achieving a mean PCC value of $(\mu \pm \sigma)$ 0.699 $\pm$0.101 and mean RMSE value of $(\mu \pm \sigma)$ 0.411

± 0.211. These are impressive results, considering the relative simplicity of the model used in FSGAN compared to the one used by Faceswap: the adversarial loss compensates the lack of multiple skip connections and residual blocks within the NN, helping to increase the batch size value. This, however, doesn't lead to poorly reconstructed details in the final swap due to the presence of the VGGFace perceptual loss which improves the direction of eyeballs to be more realistic and consistent with input face. It also smoothes out artifacts in the segmentation mask which helps in handling occlusion, eliminating artifacts, and producing natural skin tone, resulting in higher output quality.

For what concerns the swapping subject switch, the results for both De-Id and attribute preservation processes show that there aren't any sort of sex-related dependencies between the RADVESS' actors and the swap subjects: the gains and losses are attributed to the different shapes and characteristics of the swap subject's face.

Chapter 5

Conclusions

In this work, we evaluated face swap tools under a different perspective than the mere creation of deepfakes: how, and how much each of them can conceal the test subject identity (as measured by face recognition performance), keeping account of the trade off between providing privacy and usability in images (as measured by facial attribute preservation performance on De-Id images). We reviewed four state-of-the-art open source face swap frameworks: DFaker, DeepFaceLab, Faceswap and Faceswap-GAN. Through various experiments, we have shown that the selected tools performed very differently, even though their models are based on the autoencoder architecture.

The best performing framework was Faceswap with its 6 blocks autoencoder along with residual blocks and relative skip connections after each pair of layers, showing a great deal of robustness to the source image as well as for the properties of the image, from which the target face is taken. Faceswap-GAN was an interesting case of study: it used a relatively simple autoencoder but the presence of adversarial loss registered remarkable results, although the twelve hour training stage and low batch size limited its full potential.

In contrast to this, DFaker was one of the tools which achieved among the best result in the De-Id process, but failed under the attribute preservation, implying that its actions are overall input destructive. DeepFaceLab adopts an improved version of the DFaker model and this is confirmed by lower De-Id performance results which indicate far superior attribute preservation.

This work demonstrates that De-Identification is possible only with the most complex, and therefore heaviest in resource terms, face swap frameworks as Faceswap and Faceswap-GAN. DeepFaceLab has shown good results only under specific face shapes and poses, limiting its usefulness to recreational purposes. DFaker was even more stricter than DeepFaceLab, failing at almost every De-Id and attribute preservation comparison.

Moreover, this work highlighted the main drawback of face swap tools: the

model has to be re-trained for every new pair of subjects, limiting their effectiveness under real time usage. In future works we would like to evaluate deep-learning based face swapping tools which act as a modern k-Same-Select: the model is fed with a dataset which contains a wide collection of subjects with large variations in pose, age, illumination, ethnicity as the VGGFace2 dataset. This will produce a cumulative encoding vector but in order to not flatten its values during the training stage due to the dataset size, subdivide it into macro regions from high-level features (as ethnicity) to low-level features (as the nose shape) building a sort of specialized encoder. For a given pair of new subjects, the model will compare the extracted feature with the existing ones, without restart the process all over again.

Appendix A

De-Identification details

A.1 DFaker

Actors	OR - OR	OR - SW	SW - OR	SW - SW
Actor 1	0.187	0.654	0.732	0.415
Actor 2	0.176	0.656	0.665	0.436
Actor 3	0.184	0.664	0.521	0.369
Actor 4	0.179	0.717	0.721	0.351
Actor 5	0.183	0.723	0.719	0.4
Actor 6	0.189	0.749	0.692	0.362
Actor 7	0.18	0.684	0.662	0.428
Actor 8	0.018	0.727	0.727	0.385
Actor 9	0.189	0.743	0.74	0.39
Actor 10	0.171	0.748	0.723	0.38
Actor 11	0.185	0.679	0.597	0.36
Actor 12	0.184	0.74	0.696	0.441
Actor 13	0.178	0.669	0.74	0.395
Actor 14	0.184	0.688	0.664	0.393
Actor 15	0.172	0.719	0.698	0.388
Actor 16	0.17	0.736	0.703	0.438
Actor 17	0.187	0.707	0.537	0.367
Actor 18	0.183	0.708	0.686	0.391
Actor 19	0.265	0.816	0.823	0.411
Actor 20	0.262	0.816	0.593	0.406
Actor 21	0.295	0.795	0.806	0.447
Actor 22	0.1	0.782	0.681	0.376
Actor 23	0.33	0.774	0.665	0.354
Actor 24	0.058	0.757	0.595	0.394

Table 4: Average Euclidean distances worked out over 1000 randomly chosen frames of each actor, considering **DFaker**.

A.2 DeepFaceLab

Actors	OR - OR	OR - SW	SW - OR	SW - SW
Actor 1	0.187	0.666	0.659	0.361
Actor 2	0.176	0.684	0.695	0.375
Actor 3	0.184	0.713	0.69	0.35
Actor 4	0.179	0.667	0.68	0.36
Actor 5	0.183	0.695	0.688	0.352
Actor 6	0.189	0.739	0.68	0.37
Actor 7	0.18	0.656	0.653	0.367
Actor 8	0.018	0.655	0.672	0.419
Actor 9	0.189	0.743	0.657	0.412
Actor 10	0.171	0.698	0.7	0.402
Actor 11	0.185	0.708	0.688	0.387
Actor 12	0.184	0.71	0.667	0.351
Actor 13	0.178	0.748	0.699	0.382
Actor 14	0.184	0.703	0.651	0.354
Actor 15	0.172	0.729	0.696	0.373
Actor 16	0.17	0.748	0.681	0.373
Actor 17	0.187	0.723	0.693	0.362
Actor 18	0.183	0.692	0.679	0.377
Actor 19	0.265	0.801	0.738	0.372
Actor 20	0.262	0.795	0.708	0.378
Actor 21	0.295	0.883	0.722	0.371
Actor 22	0.1	0.794	0.642	0.354
Actor 23	0.33	0.762	0.604	0.355
Actor 24	0.058	0.767	0.619	0.364

Table 5: Average Euclidean distances worked out over 1000 randomly chosen frames of each actor, considering **DeepFaceLab**.

A.3 Faceswap

Actors	OR - OR	OR - SW	SW - OR	SW - SW
Actor 1	0.187	0.777	0.705	0.375
Actor 2	0.176	0.778	0.738	0.351
Actor 3	0.184	0.76	0.717	0.372
Actor 4	0.179	0.802	0.716	0.353
Actor 5	0.183	0.835	0.725	0.377
Actor 6	0.189	0.82	0.708	0.376
Actor 7	0.18	0.828	0.729	0.358
Actor 8	0.018	0.818	0.728	0.369
Actor 9	0.189	0.817	0.717	0.375
Actor 10	0.171	0.831	0.74	0.354
Actor 11	0.185	0.821	0.713	0.365
Actor 12	0.184	0.846	0.717	0.377
Actor 13	0.178	0.748	0.7	0.357
Actor 14	0.184	0.814	0.729	0.366
Actor 15	0.172	0.806	0.713	0.364
Actor 16	0.17	0.83	0.702	0.379
Actor 17	0.187	0.804	0.721	0.369
Actor 18	0.183	0.83	0.717	0.377
Actor 19	0.265	0.829	0.78	0.389
Actor 20	0.262	0.828	0.771	0.375
Actor 21	0.295	0.831	0.769	0.374
Actor 22	0.1	0.832	0.691	0.437
Actor 23	0.33	0.777	0.661	0.428
Actor 24	0.058	0.778	0.694	0.424

Table 6: Average Euclidean distances worked out over 1000 randomly chosen frames of each actor, considering **Faceswap**.

A.4 Faceswap-GAN

Actors	OR - OR	OR - SW	SW - OR	SW - SW
Actor 1	0.187	0.781	0.636	0.361
Actor 2	0.176	0.785	0.629	0.374
Actor 3	0.184	0.761	0.64	0.353
Actor 4	0.179	0.847	0.654	0.351
Actor 5	0.183	0.811	0.676	0.377
Actor 6	0.189	0.803	0.629	0.363
Actor 7	0.18	0.812	0.656	0.357
Actor 8	0.018	0.813	0.665	0.368
Actor 9	0.189	0.824	0.667	0.379
Actor 10	0.171	0.834	0.611	0.352
Actor 11	0.185	0.84	0.602	0.379
Actor 12	0.184	0.83	0.667	0.362
Actor 13	0.178	0.815	0.691	0.373
Actor 14	0.184	0.844	0.626	0.361
Actor 15	0.172	0.811	0.693	0.366
Actor 16	0.17	0.82	0.633	0.379
Actor 17	0.187	0.816	0.685	0.36
Actor 18	0.183	0.833	0.641	0.378
Actor 19	0.265	0.841	0.753	0.397
Actor 20	0.262	0.81	0.759	0.353
Actor 21	0.295	0.807	0.759	0.394
Actor 22	0.1	0.781	0.658	0.41
Actor 23	0.33	0.785	0.657	0.408
Actor 24	0.058	0.761	0.654	0.4

Table 7: Average Euclidean distances worked out over 1000 randomly chosen frames of each actor, considering **Faceswap-GAN**.

Appendix B

Facial attributes preservation details

B.1 DFaker

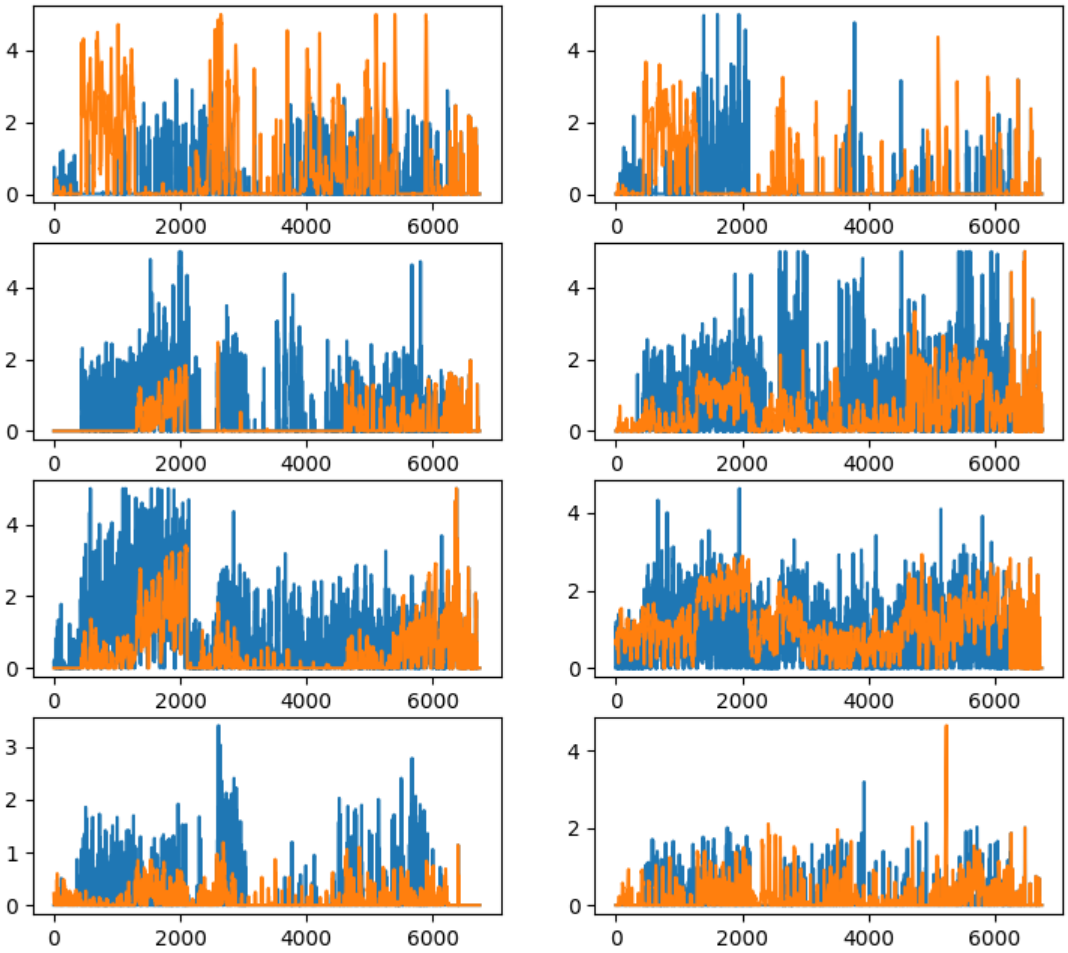


Figure 33: Cumulative and average AUs intensities obtained from original videos (blue distribution) and **DFaker** swapped videos (orange distribution). From left to right, top to bottom: AU 1, AU 2, AU 4, AU 5, AU 6, AU 7, AU 9, AU 10.

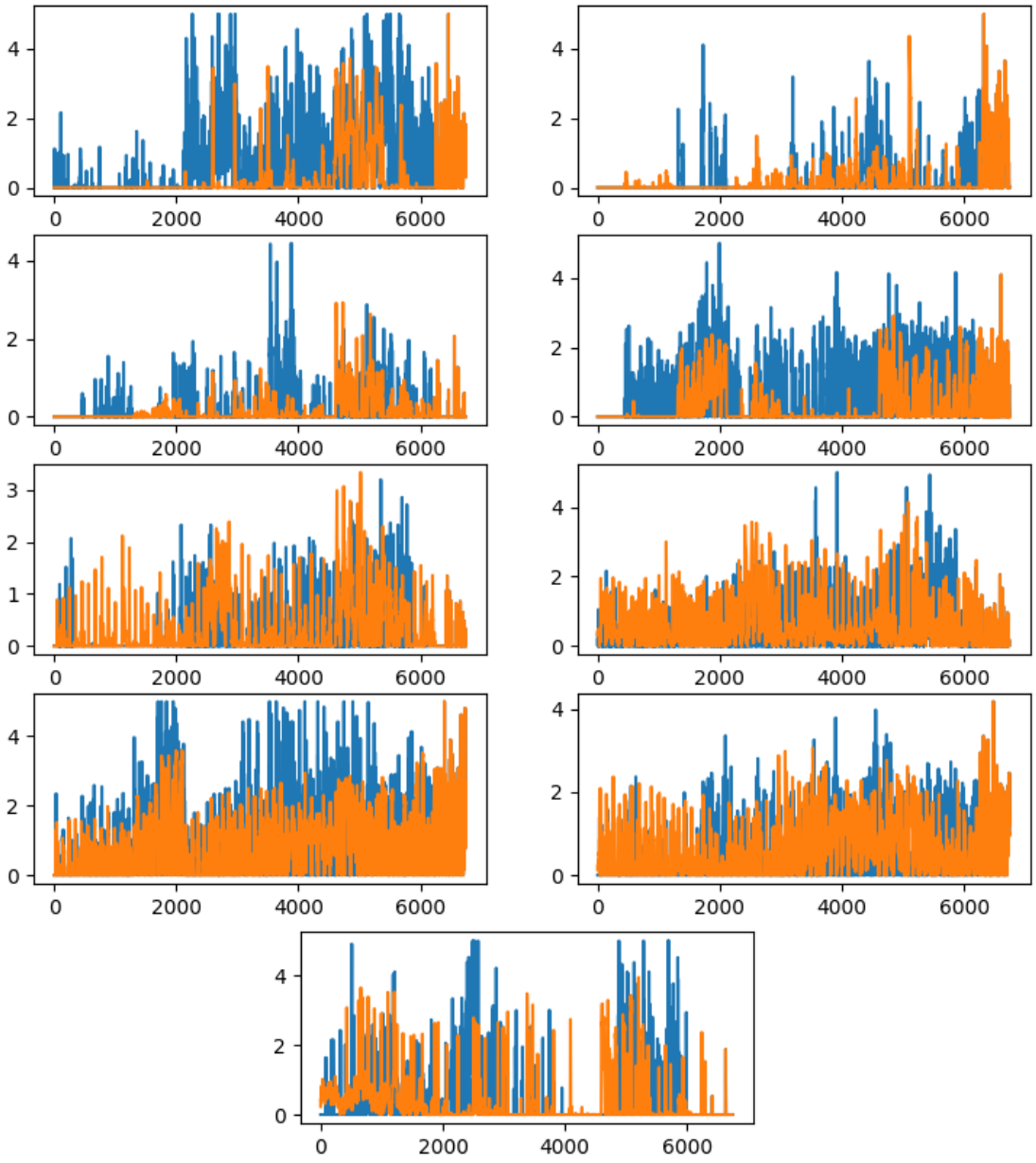


Figure 34: Cumulative and average AUs intensities obtained from original videos (blue distribution) and **DFaker** swapped videos (orange distribution). From left to right, top to bottom: AU 12, AU 14, AU 15, AU 17, AU 20, AU 23, AU 25, AU 28, AU 45.

B.2 DeepFaceLab

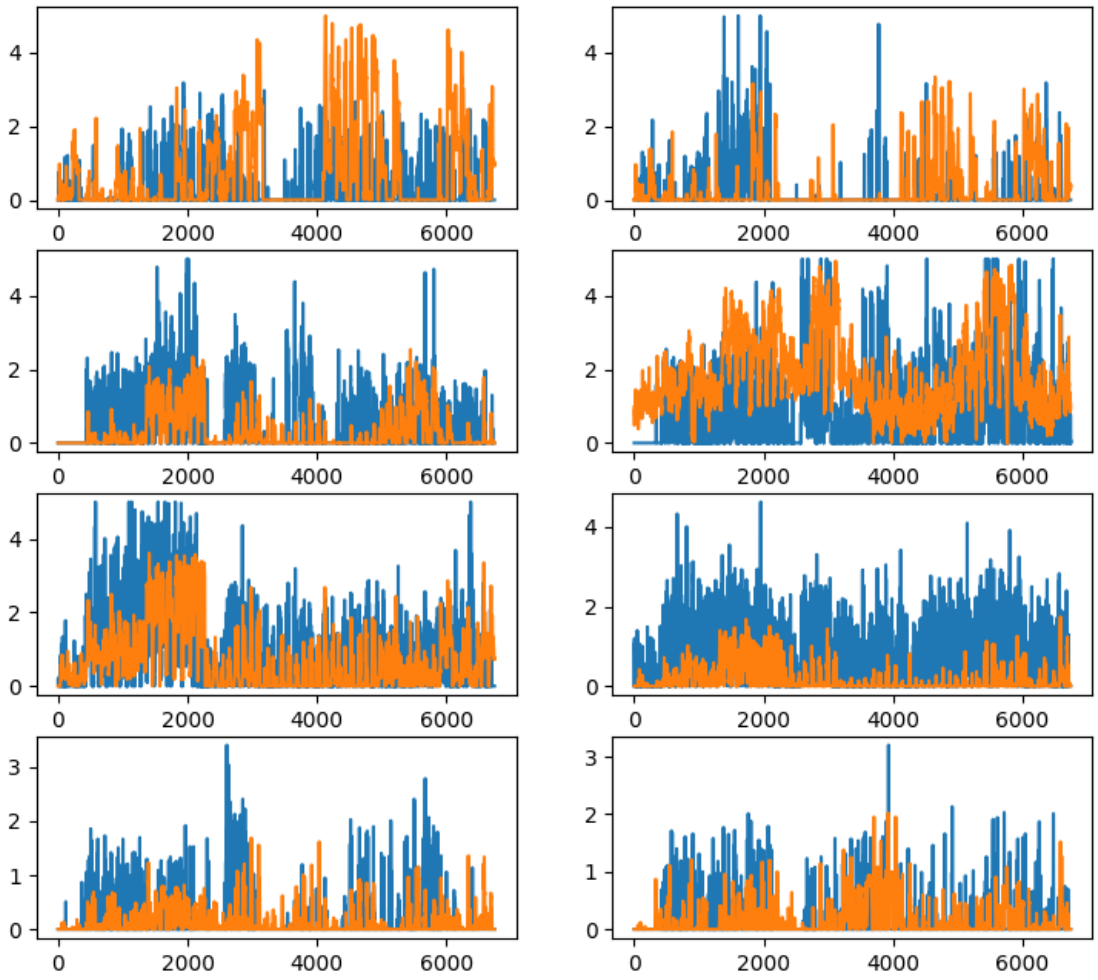


Figure 35: Cumulative and average AUs intensities obtained from original videos (blue distribution) and **DeepFaceLab** swapped videos (orange distribution). From left to right, top to bottom: AU 1, AU 2, AU 4, AU 5, AU 6, AU 7, AU 9, AU 10.

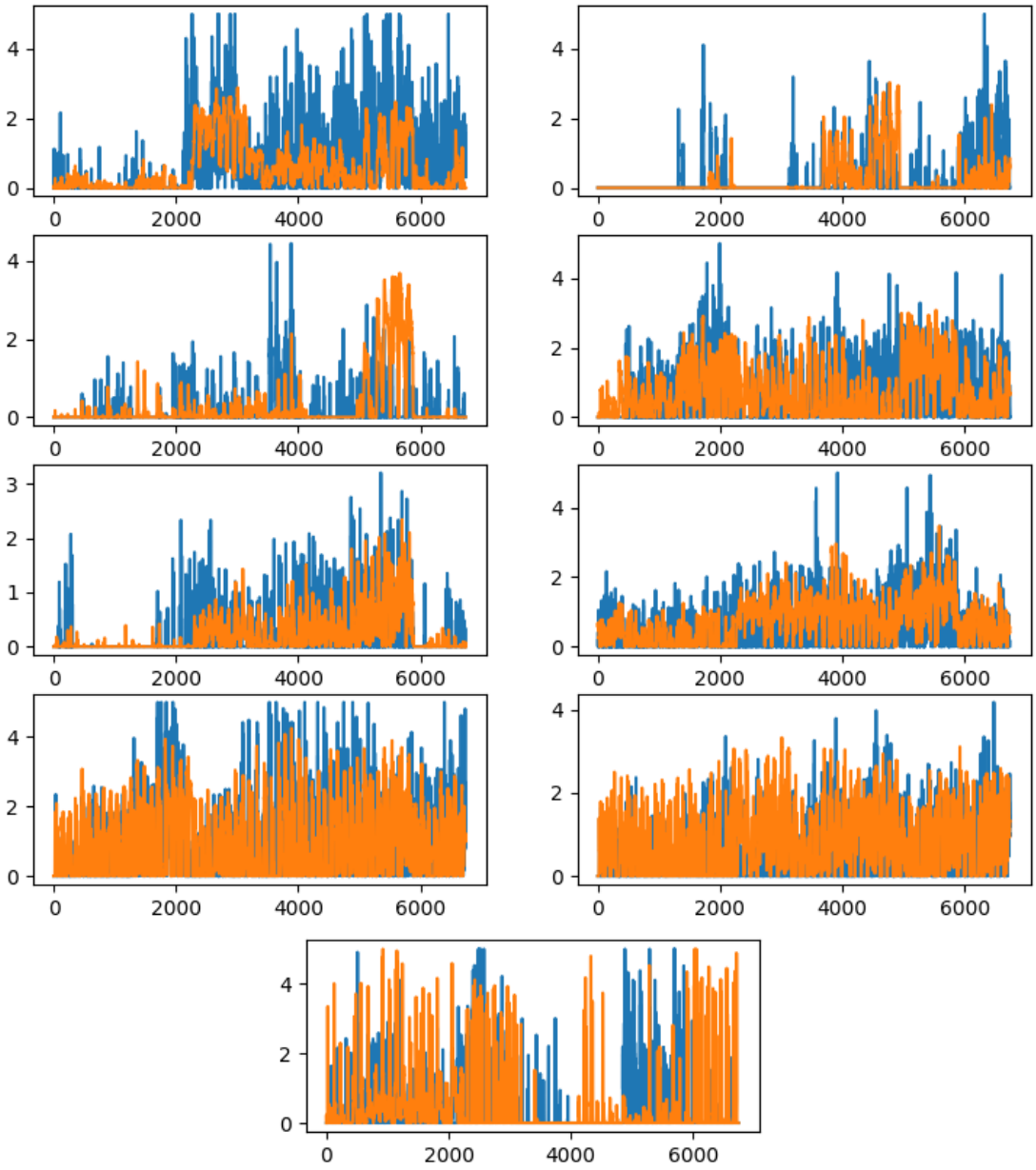


Figure 36: Cumulative and average AUs intensitiess obtained from original videos (blue distribution) and **DeepFaceLab** swapped videos (orange distribution). From left to right, top to bottom: AU 12, AU 14, AU 15, AU 17, AU 20, AU 23, AU 25, AU 28, AU 45.

B.3 Faceswap

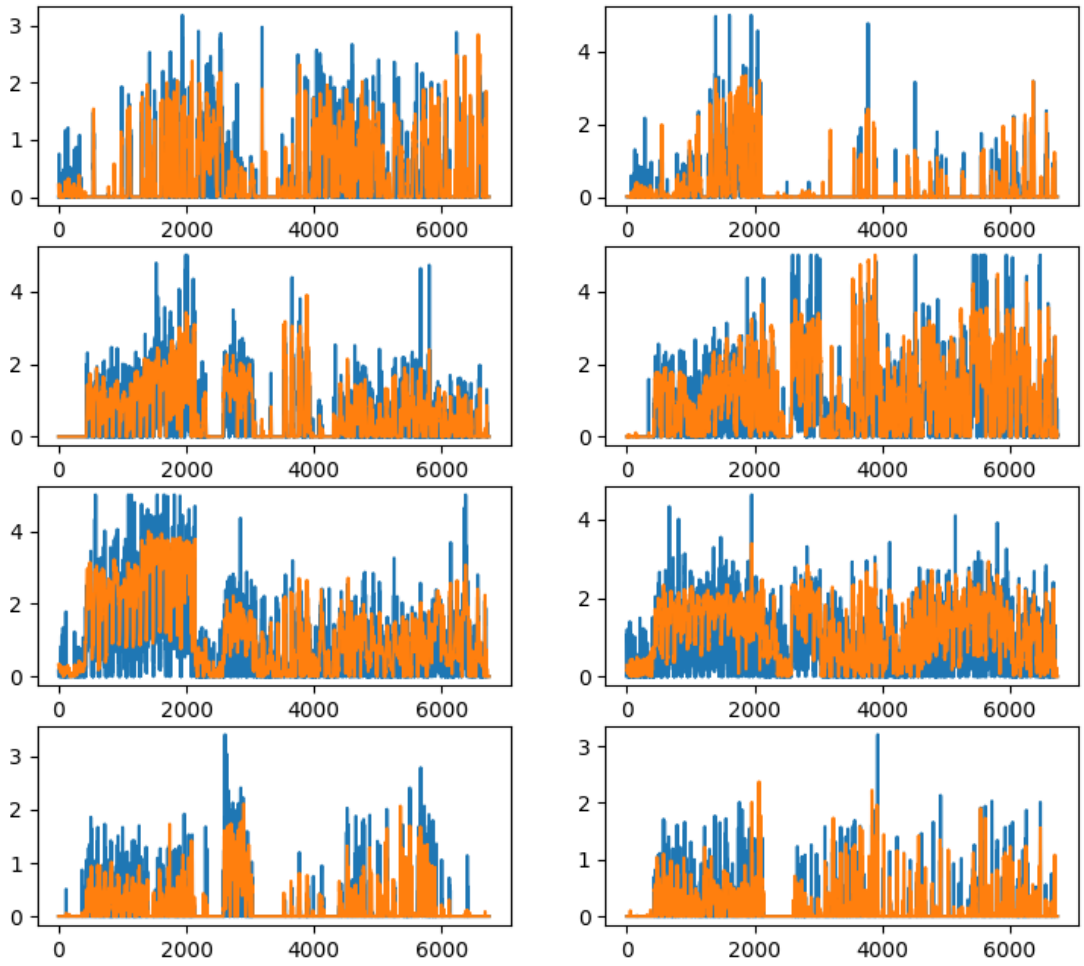


Figure 37: Cumulative and average AUs intensities obtained from original videos (blue distribution) and **Faceswap** swapped videos (orange distribution). From left to right, top to bottom: AU 1, AU 2, AU 4, AU 5, AU 6, AU 7, AU 9, AU 10.

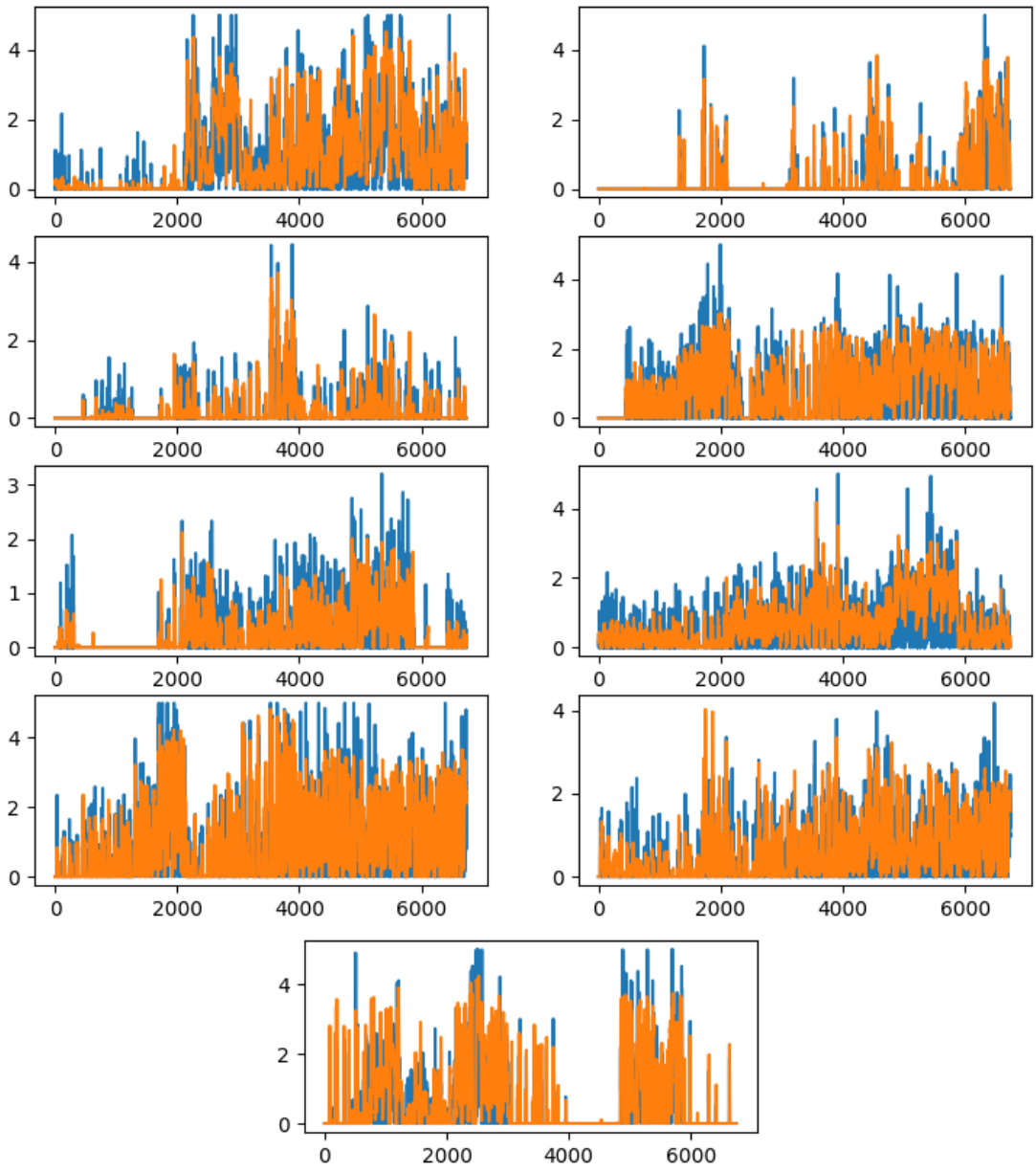


Figure 38: Cumulative and average AUs intensities obtained from original videos (blue distribution) and **Faceswap** swapped videos (orange distribution). From left to right, top to bottom: AU 12, AU 14, AU 15, AU 17, AU 20, AU 23, AU 25, AU 28, AU 45.

B.4 Faceswap-GAN

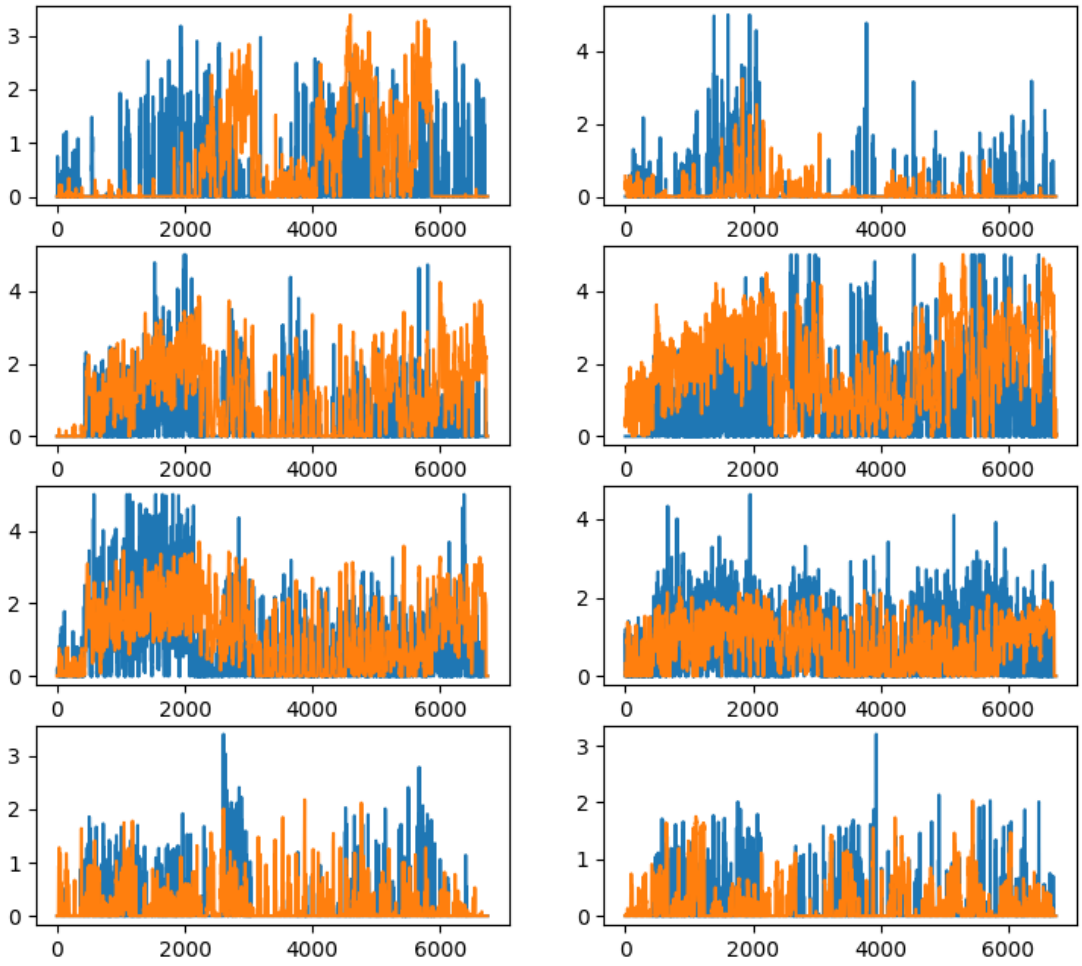


Figure 39: Cumulative and average AUs intensities obtained from original videos (blue distribution) and **Faceswap-GAN** swapped videos (orange distribution). From left to right, top to bottom: AU 1, AU 2, AU 4, AU 5, AU 6, AU 7, AU 9, AU 10.

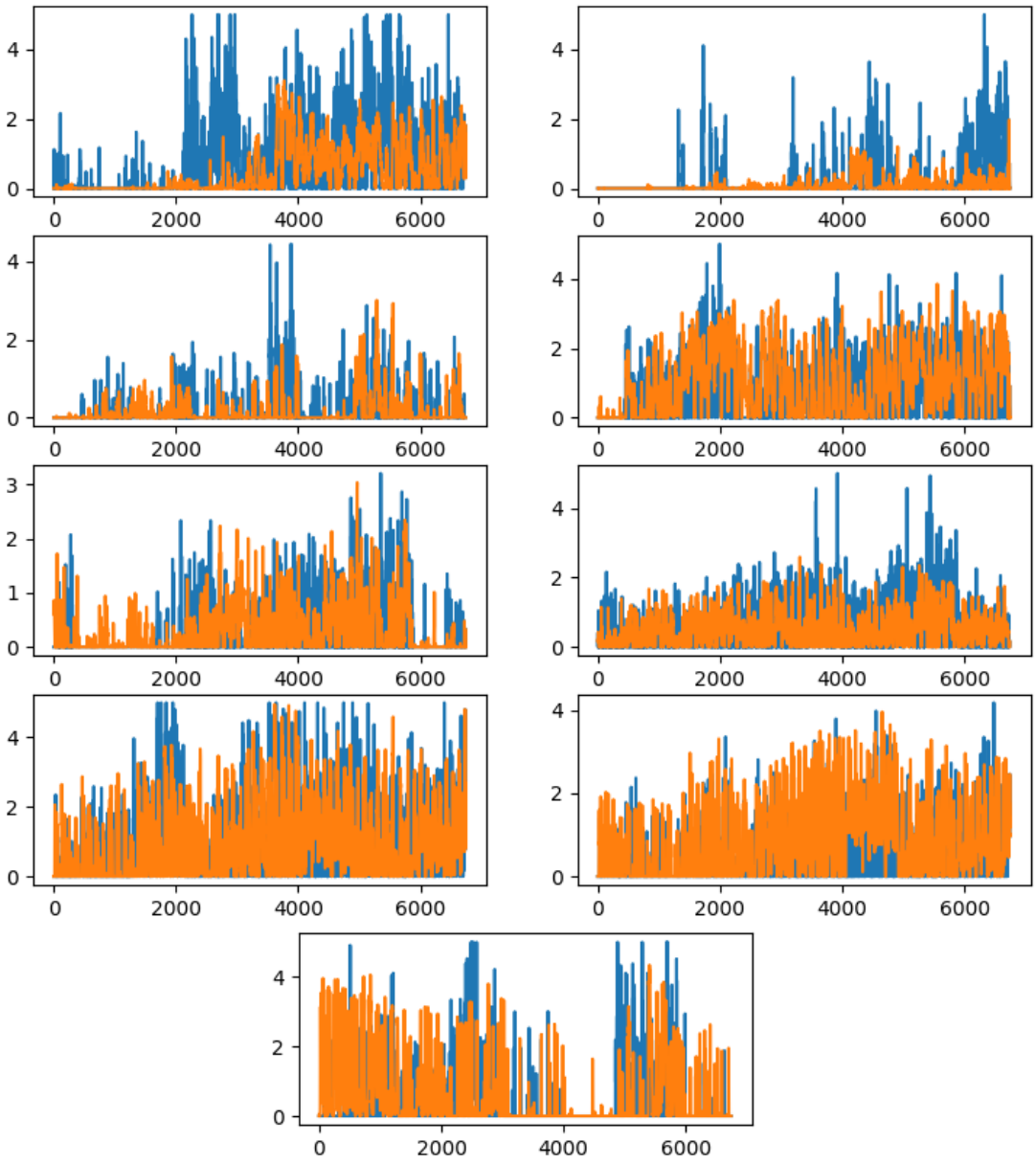


Figure 40: Cumulative and average AUs intensities obtained from original videos (blue distribution) and **Faceswap-GAN** swapped videos (orange distribution). From left to right, top to bottom: AU 12, AU 14, AU 15, AU 17, AU 20, AU 23, AU 25, AU 28, AU 45.

Appendix C

Swapping subjects switch details

Actors	A_{Male} - S_{Male}	A_{Male} - S_{Female}	A_{Female} - S_{Female}	A_{Female} - S_{Male}
Actor 1	0.732	0.719	-	-
Actor 2	-	-	0.665	0.699
Actor 5	0.719	0.741	-	-
Actor 6	-	-	0.692	0.632
Actor 9	0.74	0.698	-	-
Actor 10	-	-	0.723	0.736
Actor 13	0.597	0.715	-	-
Actor 14	-	-	0.664	0.6
Actor 17	0.537	0.514	-	-
Actor 18	-	-	0.686	0.71

Table 8: Euclidean distances for the selected subset of RADVESS' actors considering **DFaker** swapped videos. The columns refer to the actor sex compared to male and female swap subject.

Actors	A_{Male} - S_{Male}	A_{Male} - S_{Female}	A_{Female} - S_{Female}	A_{Female} - S_{Male}
Actor 1	0.659	0.671	-	-
Actor 2	-	-	0.695	0.687
Actor 5	0.688	0.715	-	-
Actor 6	-	-	0.68	0.682
Actor 9	0.657	0.743	-	-
Actor 10	-	-	0.7	0.736
Actor 13	0.699	0.714	-	-
Actor 14	-	-	0.651	0.711
Actor 17	0.693	0.784	-	-
Actor 18	-	-	0.679	0.703

Table 9: Euclidean distances for the selected subset of RADVESS' actors considering **DeepFaceLab** swapped videos. The columns refer to the actor sex compared to male and female swap subject.

Actors	A_{Male} - S_{Male}	A_{Male} - S_{Female}	A_{Female} - S_{Female}	A_{Female} - S_{Male}
Actor 1	0.705	0.724	-	-
Actor 2	-	-	0.738	0.755
Actor 5	0.725	0.712	-	-
Actor 6	-	-	0.708	0.632
Actor 9	0.717	0.720	-	-
Actor 10	-	-	0.74	0.697
Actor 13	0.7	0.721	-	-
Actor 14	-	-	0.729	0.698
Actor 17	0.721	0.734	-	-
Actor 18	-	-	0.717	0.652

Table 10: Euclidean distances for the selected subset of RADVESS' actors considering **Faceswap** swapped videos. The columns refer to the actor sex compared to male and female swap subject.

Actors	A_{Male} - S_{Male}	A_{Male} - S_{Female}	A_{Female} - S_{Female}	A_{Female} - S_{Male}
Actor 1	0.636	0.644	-	-
Actor 2	-	-	0.629	0.618
Actor 5	0.676	0.7	-	-
Actor 6	-	-	0.629	0.632
Actor 9	0.667	0.621	-	-
Actor 10	-	-	0.611	0.588
Actor 13	0.691	0.698	-	-
Actor 14	-	-	0.626	0.716
Actor 17	0.685	0.594	-	-
Actor 18	-	-	0.641	0.673

Table 11: Euclidean distances for the selected subset of RADVESS' actors considering **Faceswap-GAN** swapped videos. The columns refer to the actor sex compared to male and female swap subject.

AUs	**DFaker**		**DeepFaceLab**		**Faceswap**		**Faceswap-GAN**	
	PCC	RMSE	PCC	RMSE	PCC	RMSE	PCC	RMSE
AU 1	+0.012	**-0.009**	**+0.027**	+0.002	+0.008	-0.005	-0.004	+0.041
AU 2	-0.008	+0.017	-0.034	+0.025	**+0.078**	+0.006	+0.017	+0.031
AU 4	+0.001	+0.009	+0.02	-0.015	-0.019	-0.013	**+0.018**	**-0.019**
AU 5	**+0.022**	**-0.016**	+0.01	-0.012	-0.024	+0.018	-0.026	+0.023
AU 6	-0.03	+0.014	-0.011	+0.006	**+0.021**	+0.013	-0.032	**-0.018**
AU 7	+0.013	**-0.018**	**+0.031**	-0.005	-0.012	+0.023	+0.007	+0.014
AU 9	+0.025	-0.005	+0.003	-0.006	+0.006	**-0.014**	**+0.027**	-0.009
AU 10	–0.009	**-0.002**	+0.005	+0.019	**+0.009**	+0.012	-0.021	+0.032
AU 12	-0.004	-0.006	+0.019	+0.022	-0.003	+0.014	-0.02	**-0.021**
AU 14	+0.005	**-0.015**	-0.023	+0.004	**+0.011**	+0.019	-0.005	-0.013
AU 15	+0.012	-0.017	**+0.021**	-0.012	+0.021	+0.023	+0.006	-0.012
AU 17	-0.025	**-0.023**	**+0.02**	+0.024	-0.024	+0.019	-0.016	-0.021
AU 20	**+0.017**	+0.011	+0.022	-0.018	-0.018	-0.043	-0.016	**-0.034**
AU 23	-0.014	-0.004	+0.013	**-0.008**	-0.011	**+0.012**	-0.005	+0.035
AU 25	+0.003	+0.02	-0.013	-0.003	**+0.007**	+0.019	-0.026	**-0.022**
AU 28	+0.002	+0.027	-0.016	**-0.006**	-0.002	+0.012	**+0.031**	+0.028
AU 45	-0.011	**-0.018**	-0.015	-0.002	**+0.023**	+0.017	-0.017	-0.014

Table 12: Differences for each framework of the attributes preservation process for the selected subset of RADVESS' actors. The values are obtained comparing same-sex swapping pairs (i.e. M-M / F-F) and switched-sex swapping pairs (i.e. M-F / F-M). The best gains from the sex-switch swapping subject are in bold.

Bibliography

[1] José Ramón Padilla-López, Alexandros Andre Chaaraoui, and Francisco Flórez-Revuelta. Visual privacy protection methods: A survey. *Expert Systems with Applications*, 42(9):4177–4195, 2015.

[2] Håkon Hukkelås, Rudolf Mester, and Frank Lindseth. Deepprivacy: A generative adversarial network for face anonymization. *arXiv preprint arXiv:1909.04538*, 2019.

[3] Michael Boyle, Christopher Edwards, and Saul Greenberg. The effects of filtered video on awareness and privacy. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 1–10. ACM, 2000.

[4] Ralph Gross, Latanya Sweeney, Jeffrey Cohn, Fernando De la Torre, and Simon Baker. Face de-identification. *Protecting Privacy in Video Surveillance*, pages 129–146, 07 2009.

[5] Chris Stauffer and W Eric L Grimson. Adaptive background mixture models for real-time tracking. In *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, volume 2, pages 246–252. IEEE, 1999.

[6] Kentaro Toyama, John Krumm, Barry Brumitt, and Brian Meyers. Wallflower: Principles and practice of background maintenance. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 1, pages 255–261. IEEE, 1999.

[7] Elaine M Newton, Latanya Sweeney, and Bradley Malin. Preserving privacy by de-identifying face images. *IEEE transactions on Knowledge and Data Engineering*, 17(2):232–243, 2005.

[8] Qianru Sun, Liqian Ma, Seong Joon Oh, Luc Van Gool, Bernt Schiele, and Mario Fritz. Natural and effective obfuscation by head inpainting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5050–5059, 2018.

[9] Ralph Gross, Iain Matthews, Jeffrey Cohn, T Kanade, and S Baker. The cmu multi-pose, illumination, and expression (multi-pie) face database. *CMU Robotics Institute. TR-07-08, Tech. Rep*, 2007.

[10] Ralph Gross, Latanya Sweeney, Fernando De la Torre, and Simon Baker. Model-based face de-identification. In *2006 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'06)*, pages 161–161. IEEE, 2006.

[11] Ralph Gross and Latanya Sweeney. Towards real-world face de-identification. In *2007 First IEEE International Conference on Biometrics: Theory, Applications, and Systems*, pages 1–8. IEEE, 2007.

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[13] Tao Li and Lei Lin. Anonymousnet: Natural face de-identification with measurable privacy. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[14] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3–es, 2007.

[15] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 106–115. IEEE, 2007.

[16] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[17] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8789–8797, 2018.

[18] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.

[19] Alex Liang, Yu Su, and Fangyan Zhang. Afacial recognition-based video encryption approach to prevent fakedeep videos.

[20] Xin Yang, Yuezun Li, Honggang Qi, and Siwei Lyu. Exposing gan-synthesized faces using landmark locations. In *Proceedings of the ACM Workshop on Information Hiding and Multimedia Security*, pages 113–118, 2019.

[21] Shifeng Zhang, Xiangyu Zhu, Zhen Lei, Hailin Shi, Xiaobo Wang, and Stan Z Li. S3fd: Single shot scale-invariant face detector. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 192–201, 2017.

[22] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, Oct 2016.

[23] Adrian Bulat and Georgios Tzimiropoulos. Super-fan: Integrated facial landmark localization and super-resolution of real-world low resolution faces in arbitrary poses with gans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 109–117, 2018.

[24] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. Faceforensics: A large-scale video dataset for forgery detection in human faces. *arXiv preprint arXiv:1803.09179*, 2018.

[25] Sahar Zafar, Fayyaz Ali, Subhash Guriro, Irfan Ali, Asif Khan, and Adnan Zaidi. Facial expression recognition with histogram of oriented gradients using cnn. *Indian Journal of Science and Technology*, 12, 07 2019.

[26] Muhammad Usman, Siddique Latif, and Junaid Qadir. Using deep autoencoders for facial expression recognition. In *2017 13th International Conference on Emerging Technologies (ICET)*, pages 1–6. IEEE, 2017.

[27] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[28] Boris Hanin and David Rolnick. How to start training: The effect of initialization and architecture. In *Advances in Neural Information Processing Systems*, pages 571–581, 2018.

[29] Andrew Aitken, Christian Ledig, Lucas Theis, Jose Caballero, Zehan Wang, and Wenzhe Shi. Checkerboard artifact free sub-pixel convolution: A note on sub-pixel convolution, resize convolution and convolution resize. *arXiv preprint arXiv:1707.02937*, 2017.

[30] Armen Aghajanyan. Convolution aware initialization. *arXiv preprint arXiv:1702.06295*, 2017.

[31] Fatin EM Al-Obaidi. Image quality assessment for defocused blur images. *American Journal of Signal Processing*, 5(3):51–55, 2015.

[32] Yingjing Lu. The level weighted structural similarity loss: A step away from mse. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9989–9990, 2019.

[33] Agung W Setiawan, Tati R Mengko, Oerip S Santoso, and Andriyan B Suksmono. Color retinal image enhancement using clahe. In *International Conference on ICT for Smart Society*, pages 1–3. IEEE, 2013.

[34] Hao-Wen Dong and Yi-Hsuan Yang. Towards a deeper understanding of adversarial losses. *arXiv preprint arXiv:1901.08753*, 2019.

[35] Minjun Li, Haozhi Huang, Lin Ma, Wei Liu, Tong Zhang, and Yugang Jiang. Unsupervised image-to-image translation with stacked cycle-consistent adversarial networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 184–199, 2018.

[36] Steven R Livingstone and Frank A Russo. The ryerson audio-visual database of emotional speech and song (ravdess): A dynamic, multimodal set of facial and vocal expressions in north american english. *PloS one*, 13(5), 2018.

[37] Kingma DP. Ba j. adam: a method for stochastic optimization. In *The international conference on learning representations*, 2015.

[38] Xiaojiao Mao, Chunhua Shen, and Yu-Bin Yang. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. In *Advances in neural information processing systems*, pages 2802–2810, 2016.

[39] Gary B Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. 2008.

[40] Y-I Tian, Takeo Kanade, and Jeffrey F Cohn. Recognizing action units for facial expression analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 23(2):97–115, 2001.